University of Southern Queensland
Faculty of Health, Engineering and Sciences

DOMINATION PROBLEMS IN SOCIAL NETWORKS

A Dissertation submitted by
Guangyuan Wang, MSc

For the award of
Doctor of Philosophy

2014

Abstract

The thesis focuses on domination problems in social networks. Domination problems are one of the classical types of problems in computer science. Domination problems are fundamental and widely studied problems in algorithms and complexity theory. They have been extensively studied and adopted in many real-life applications. In general, a set D of vertices of a simple (no loops or multiple edges), undirected graph $G = (V, E)$ is called *dominating* if each vertex in $V - D$ is adjacent to some vertex in D. The computational problem of computing a dominating set of minimum size is known as *"the dominating set problem"*. The dominating set problem is NP-hard in general graphs.

A social network - the graph of relationships and interactions within a group of individuals - plays a fundamental role as a medium for the spread of information, ideas, and influence among its members. In a social network, people, who have problems such as drinking, smoking and drug use related issues, can have both positive and negative impact on each other and a person can take and move among different roles since they are affected by their peers. As an example, positive impacts of intervention and education programs on a properly selected set of initial individuals can diffuse widely into society via various social contacts: face to face, phone calls, email, social networks and so on. Exploiting the relationships and influences among individuals in social networks might offer considerable benefit to both the economy and society.

In order to deal with social problems, the positive influence dominating set (PIDS) is a typical one to help people to alleviate these social problems. However, existing PIDS algorithms are usually greedy and finding approximation solutions that are inefficient for the growing social networks. By now these proposed algorithms can deal with social problems only in undirected social networks with uniform weight value. To overcome the shortcomings of the existing PIDS model, a novel domination model namely weight positive influence dominating set (WPIDS) is presented. A main contribution of the thesis is that the proposed WPIDS model can be applied in weighted directed social networks. It considers the direction and degree of users' influence in social networks in which the PIDS model does not. The experimental results have revealed that the WPIDS model is more effective than the PIDS model.

At the same time, thanks to the publication of Dijkstra's pioneering paper, a lot of self-stabilizing algorithms for computing minimal dominating sets have been proposed, such as the self-stabilizing algorithms for minimal single dominating sets and minimal k-dominating sets (MKDS). However, for the MKDS problem, so far there is no self-stabilizing algorithm that works in arbitrary graphs. The proposed algorithms for the MKDS either work for tree graphs or find a minimal 2-dominating set. So, in the thesis, for the MKDS problem, two self-stabilizing algorithms are presented that can operate on general graphs. For the weighted dominating set (WDS) problem, most of the proposed algorithms find approximation solutions to a WDS. For the non-uniform WDS problem, there is no self-stabilizing algorithm for the WDS. In the thesis, self-stabilizing algorithms for the minimal weighted dominating set (MWDS) and minimal positive influence dominating set (MPIDS) are presented when operating in any general network. The worst case convergence time of the two algorithms from any arbitrary initial state are also proved. Finally, in order to reduce cost in an education/intervention programme arising from the PIDS problem, two cooperative cost games about PIDS problem are constructed.

Keywords: Social problems, Dominating set, Positive influence dominating set, Weighted positive influence dominating set, K-dominating set, Weighted dominating set, Self-stabilizing algorithm, Daemon, Cooperative cost games, Computing complexity.

Certification of Dissertation

I certify that the ideas, experimental work, results, analysis and conclusions reported in this dissertation are entirely my own effort, except where otherwise acknowledged. I also certify that the work is original and has not been previously submitted for any other award.

Guangyuan Wang, Candidate	Date

ENDORSEMENT

Prof. Hua Wang, Principal supervisor	Date
Dr Ji Zhang, Co-supervisor	Date
Dr Xiaohui Tao, Co-supervisor	Date

Acknowledgments

I would like to thank my supervising Professor Hua Wang for constantly motivating and encouraging me, and also for his invaluable advice during the course of my doctoral studies. I am extremely fortunate to have had the opportunity to work with him over the past few years. In addition, I would like to thank Dr. Xiaohui Tao and Dr. Ji Zhang for their support, feedback, and encouragement.

I would also like to extend my appreciation to the Centre for Systems Biology (CSBi), Department of Mathematics and Computing, Faculty of Science and Research and Higher Degree office of University of Southern Queensland for providing the excellent study environment and financial support. It is my great pleasure to study at the Department of Mathematics and Computing.

I also acknowledge our friends Alan and Mee Wah Roocroft for their help on proof-reading the dissertation. Finally, on a personal note, I would never have made it this far without the continuing support and encouragement from my family; Qinghe Wang, Bingfen Su, and Menglin Qiao.

List of Publications

The following publications were produced during the period of candidature:

1. G. Wang, H. Wang, X. Tao and J. Zhang: Finding Weighted Positive Influence Dominating Set to Make Impact to Negatives - A Study on Social Networks in New Millennium. To appear in ICTs and the Millennium Development Goals - A United Nations Perspective (Ed. by Haur and Tao), 2014, Springer.

2. G. Wang, H. Wang, X. Tao, J. Zhang and X. Yi: Positive influence dominating set games. To appear in the 18th IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD 2014), 2014.

3. G. Wang, H. Wang, X. Tao, J. Zhang and J. Zhang: Minimising k-dominating set in arbitrary network graphs. In proceedings of the 9th International Conference on Advanced Data Mining and Applications (ADMA 2013), pp: 120-132, 2013.

4. G. Wang, H. Wang, X. Tao, J. Zhang and G. Zhu: Finding a weighted positive influence dominating set in e-learning social networks. Internal Journal Computing & Technology, 10(10), pp: 2136-2145, 2013.

5. G. Wang, H. Wang, X. Tao and J. Zhang: A self-stabilizing protocol for minimal weighted dominating sets in arbitrary networks. In proceedings of the 17th IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD 2013), pp: 496-501, 2013.

6. G. Wang, H. Wang, X. Tao and J. Zhang: A self-stabilizing algorithm for finding a minimal positive influence dominating set in social networks. In proceedings of 24th Australasian Database Conference (ADC 2013), pp: 93-99, 2013.

7. G. Wang, H. Wang, X. Tao and J. Zhang: A self-stabilizing algorithm for finding a minimal k-dominating set in general networks. In proceedings of 2012 International Conference on Data and Knowledge Engineering (ICDKE2012), pp: 74-85, 2012.

8. G. Wang, H. Wang, X. Tao and J. Zhang: Positive influence dominating set in e-Learning social networks. In proceedings of the 10th International Conference on Web-based Learning (ICWL 2011), pp: 82-91, 2011.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Graph Domination problems are one of the classical types of problems in computer science. Domination problems are fundamental and widely studied problems in algorithms and complexity theory. They are an important class of combinatorial problems with many practical and theoretical applications. For example, in constructing a cellular phone network, one needs to choose locations for the towers to cover a large region as cheaply as possible. This is a domination problem: the vertices of the graph are the locations that need coverage, the neighborhood of a vertex is the area that a tower at that vertex would cover, and a dominating set is any set of locations at which one could place towers to cover the whole region.

Mathematical study of domination in graphs began around 1960, beginning with Claude Berge [1] in 1962. He introduced the coefficient of external stability, which is now known as the domination number of a graph. Oystein Ore [2] introduced the terms dominating set and domination number in his book on graph theory which was published in 1962. A decade later, Cockayne and Hedetniemi [3] published a survey paper, in which the notation $\gamma(G)$ was first used for the domination number of a graph G. In 1998, a book [4] on domination has been published which lists 1222 papers in this area. Since this book was published, domination in graphs has been studied extensively and several additional research papers have been published on this topic. Recently, domination problems have been extensively studied and adopted in many real-life applications in fields such as optimisation, communication networks and network design, social network theory, computational complexity, and algorithm design. Many facility location, resource allocation, and scheduling problems, are variants of graph domination problems [5–10].

In general, a set D of vertices of a simple (no loops or multiple edges), undirected graph $G = (V, E)$ with vertex set V and edge set E is called *dominating* if each vertex in $V - D$ is adjacent to some vertex in D. That is if every vertex $v \in V$ is either in D or adjacent to some vertex in D. For clarity, we assume that all the graphs in this thesis are simple and undirected unless separately defined. Let us start by formally introducing a prominent graph domination problem:

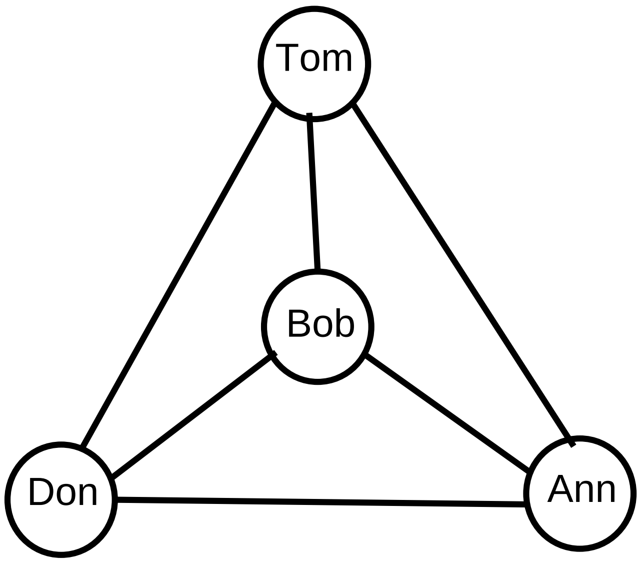

Figure 1.1: An example of a dominating set

Dominating Set Problem[1]
Input: A graph $G = (V, E)$ with vertex set V and edge set E and an integer $k \in N$ (N is the set of natural numbers).
Question: Does there exist a dominating set $D \subseteq V$ in G of size at most k?

We use an illustration to further understand the dominating set (DS). For example, there are four persons in Fig. 1.1 and their relationships constitute a complete graph. We can say anyone of the four persons is a DS of the four persons group. Due to the practical applications of dominating sets, there are many variations of dominating sets such as Total Dominating Set (TDS) [11–14], Weighted Dominating Set (WDS) [15–18], Connected Dominating Set (CDS) [19–24], K-Dominating Set (KDS) [25, 26], Positive Influence Dominating Set (PIDS) [27, 28] and so on.

Recently, online social network has developed significantly in the new millennium. For example, online social network sites like Facebook, MySpace and Twitter are among the most popular sites on the Internet. Some research has been done to understand the properties of online social networks [29–34] and how to effectively utilize social networks to spread ideas and information within a group [27, 28, 35, 36]. Among these research, Wang et. al explored the problem of how to utilize online social networks to help alleviate social problems in the physical world [27, 28]. The social problems include drinking, smoking, drug use and so on. Within the context of the drinking problem, for example, a person can be an abstainer, or an alcoholic. An abstainer has positive impact on his direct friends (called neighbourhoods), but he might turn into an alcoholic and have negative impact on his neighbourhoods if many of his friends are alcoholics and vice versa. Ideally, in an alcohol intervention programme the purpose is to educate all alcoholics, since this will reduce the possibility of the converted alcoholics being influenced by his alcoholic friends who are not selected in the intervention programme. On the other hand, due to the financial limitations in budget, it is too

[1]We will develop algorithms for the dominating set problem in the following Chapters.

expensive to select all the alcoholics in an intervention programme. Therefore, it becomes an important research problem as to how to select a subset of individuals to be part of the programme so that the effect of the intervention programme can spread through the whole group under consideration.

To deal with the above social problems, Wang et. al [27] introduced the notion of the positive influence dominating set (PIDS) and proposed a greedy approximation PIDS selection algorithm in 2009. Recall that $D \subseteq V$ is a *positive influence dominating set* (PIDS) [27,28] if any node $v \in V$ is dominated by at least $\lceil \frac{d(v)}{2} \rceil$ nodes (that is, v has at least $\lceil \frac{d(v)}{2} \rceil$ neighbours) in D where $d(v)$ is the degree of node v. Note that there are two requirements for PIDS:

(i) Every node not in D has at least half of its neighbours in D;

(ii) Every node in D also has at least half of its neighbours in D.

Wang et al. [27] revealed that approximately 60% of the whole group under consideration needs to be selected into the PIDS in order to achieve the goal that every individual in the community has more positive neighbours than negative neighbours. However, the proposed algorithm can deal with social problems only in undirected social networks with uniform weight value. If we consider some key factors, such as the attribute, direction and degree of each person's influence, the size of the solution for selecting a proper subset of the whole group might be smaller and the algorithm might be more effective and economical. Exploiting the relationships and influences among individuals in social networks might offer considerable benefit to both the economy and society.

Following the key factors as we analyzed above, Figure 1.1 is a proper example which illustrates the scenario as discussed above. In Figure 1.1, Bob, Tom, Don and Ann are four equal e-learners in a small group. Any three of them is a PIDS satisfying its definition (requirements (i) and (ii)). But if Bob is a tutor and has strong positive influence on others, only Bob can positively affect (dominate) others.

In this thesis we would like to explore how to select a subset of individuals to be part of the programme so that the effect of the intervention programme can spread through the whole group under consideration. We consider the degree and direction of each user's influence, propose a novel DS named weighted positive influence dominating set (WPIDS) and develop two WPIDS selection algorithms in e-learning environments.

The computational problem of computing a DS of minimum size is known as domination problems. Domination problems are all NP-hard in general graphs [11,37]. Nowadays, it is commonly believed that NP-hard problems cannot be solved in polynomial times. For these problems one remaining hope is to design exact algorithms with good exponential running times [38, 38–44]. The fastest currently known exact algorithms for the DS are around $O(1.5^n)$ where n is the number of nodes [41, 42]. So numbers of researchers study exact exponential algorithms for the domination problem on certain special graph classes [38, 43]. Another research direction is to compute approximation solutions to domination problems.

Research on local algorithms has been thriving again [45–48], probably thanks to emerging applications in ad hoc and sensor networks. To get a virtual backbone (connected DS), Guha, Khuller, and Du et al. [20–23,49–51] presented some two-stage greedy approximation algorithms and interesting results by utilizing the property of wireless networks.

Owing to the rapid development of social network services, domination problems in social networks have become important issues [27,28,36,52]. Eubank et al. [36] proposed a greedy approximation algorithm and proved that the algorithm gives an $1 + O(1)$ approximation with a small constant in $O(1)$ to the DS problem in a power-law graph. Kleinberg et al. [53] studied online social networks, in which relationships can be either positive or negative. Wang et al. introduced a new DS, called PIDS which can help alleviate social problems in the physical world [27,28].

Although domination problems are all NP-hard in general graphs [11,37], which means to find a minimum DS is NP-hard in general graphs, some proposed self-stabilizing algorithms for minimal dominating sets are in polynomial time (steps or rounds) complexity [26,54] or linear time (steps or rounds) complexity [54,55].

Self-stabilization is an optimistic fault tolerance approach for distributed systems. It was introduced by Dijkstra [56, 57]. Self-stabilization can be used to solve a variety of graph theoretic problems such as dominating sets [58, 59], independent sets [54, 59, 60], colorings [61–63], and matchings [64–66] problems in graphs. According to Dijkstra's work, a self-stabilizing system is guaranteed to reach a correct state, in a finite time, regardless of its initial state [67]. Thus, a self-stabilizing system can recover from any transient fault without any external intervention. Self-stabilization is also a non-masking approach, since after the occurrence of a transient fault, the system exhibits temporarily disrupted behaviour for a certain period of time.

A fundamental idea of self-stabilizing algorithms is that the distributed system may be started from an arbitrary global state. After finite time the system reaches a correct global state, called a *legitimate* or *stable* state. An algorithm is *self-stabilizing* if the following two properties hold: *convergence* and *closure*. That is, when the system executes the algorithm,

(1) for any initial illegitimate state it reaches a legitimate state after a finite number of node moves (*convergence*), and

(2) for any legitimate state and for any move allowed by that state, the next state is a legitimate state (*closure*).

However, the current proposed self-stabilizing algorithms for the minimal k-dominating set (MKDS) problem either work for tree graphs (Kamei and Kakugawa [26] or find a minimal 2-dominating set (Huang et al. [55, 68]). There are seldom self-stabilizing algorithms for the MKDS that work in arbitrary graphs. Another shortcoming for the generalized DS (Goddard et al. [58]) is that most of the existing algorithms work for uniform weight cases. The weight value of each node in most of the proposed algorithms is uniform [26,26,54,55,58]. Meanwhile, the polynomial time approximation algorithms for weighted domination problems are all greedy algorithms for finding approximation solutions [16–18]. There are seldom self-stabilizing algorithms for the weighted dominating set (WDS) problem.

In this thesis, we have the motivations to extend the PIDS model which can deal with social problems only in undirected social networks with uniform weight and the minimal 2-dominating set (M2DS) to the general k: the minimal k-dominating set (MKDS). We have proposed a novel DS, namely weight positive influence dominating set (WPIDS) considering the direction and degree of users' influence in weighted directed social networks in which the PIDS model does not. To extend Huang et al.'s work, two self-stabilizing algorithms have been proposed for the MKDS when operating in any general networks. Moreover, two self-stabilizing algorithms for finding a minimal weighted dominating set (MWDS) and an MPIDS under a central daemon in a general graph are presented. Finally, we consider cooperative cost games arising from the PIDS problem which can deal with social problems. We introduce two games, the rigid PIDS game and relaxed PIDS game and focus on the cores of both games.

1.2 Research Problem Statement

Based on the background knowledge and the research gap we discuss above, we list five research problems.

- **Problem A**: How to find a smaller DS in an education/intervention programme to positively dominate the whole group?
- **Problem B**: How to find an MKDS[2] in any general network.
- **Problem C**: How to find an MWDS[3] in any general networks.
- **Problem D**: How to find an MPIDS[4] in any general social network.
- **Problem E**: How to distribute the total cost among the individual players in a "fair" way to ensure that no coalition would have an incentive to split from the grand coalition V, and do better on its own when there is a PIDS.

1.3 Contributions

In this section, we list our models and algorithms which we have solved these five problems we listed above.

First, the research is as to how to effectively select positive e-learners to affect an individual in the e-learning network becomes "positive" if half of its neighbours are "positive" about adopting a product or behaviour. We give both theoretical justification and empirical verification for the two proposed selection algorithms [69, 70]. Second, we are interested in an MKDS in any general networks [71, 72].

[2]Refer to Definition 3.1 in Chapter 3.
[3]Refer to Definition 5.1 in Chapter 5.
[4]Refer to Definition 6.1 in Chapter 6.

Third, we extend Huang et al.'s work and consider the extension problem of M2DS mentioned in [55, 68]. We will solve that extension problem for general k (i.e., for k being an arbitrary positive integer) in general networks. We firstly develop two new self-stabilization algorithms for finding an MKDS in a general network. The first algorithm works under a central daemon [71] and the second one works under a distributed daemon [72]. We discuss the PIDS problem further. We propose a self-stabilizing algorithm for an MPIDS [73] and discuss the core of the PIDS games [74]. The contributions of this thesis are as follows:

1. A new DS named WPIDS and its model are defined. Two WPIDS selection algorithms to solve Problem A are proposed. The model reasonably utilizes its online social network structure to help e-learners to improve their study achievements [69,70]. It overcomes the drawbacks that the PIDS model can deal with social problems only in undirected social networks with uniform weight. The effectiveness of our two WPIDS selection algorithms has been evaluated by simulation experiments. The experimental results show the size of the WPIDS is much smaller than that of the PIDS model.

2. A self-stabilizing algorithm for finding an MKDS under a central daemon in an arbitrary simple connected undirected network graph to solve Problem B is presented. The correctness of the proposed algorithm is verified. The computational complexity of the algorithm is proved that the worst case convergence time of the algorithm from any arbitrary initial state is $O(n^2)$ steps where n is the number of nodes in the network [71].

3. A self-stabilizing algorithm for finding an MKDS under a distributed daemon in an arbitrary connected simple undirected graph to research Problem B further is presented. The correctness of the proposed algorithm is verified; the worst case convergence time of the algorithm from any arbitrary initial state is proved to be $O(n^2)$ steps where n is the number of nodes in the network graph [72].

4. A self-stabilizing protocol for finding an MWDS in an arbitrary network graph under a central daemon to solve Problem C is proposed. The worst case convergence time of the protocol from any arbitrary initial state is proved to be $O(n^2)$ steps where n is the number of nodes in the network [75].

5. A self-stabilizing algorithm for finding an MPIDS in an arbitrary network graph under a central daemon to solve Problem D. The worst case convergence time of the protocol from any arbitrary initial state is proved to be $O(n^2)$ steps where n is the number of nodes in the network [73].

6. Two new game models, the rigid PIDS game and relaxed PIDS game are presented. A relationship between the cores of both games is obtained. The core of the relaxed PIDS game is discussed to solve Problem E [74].

1.4 Structure of the Thesis

The outcomes of about three years worth of research for the PhD thesis has been presented in eight chapters. Briefly the contents of each chapter are as follows:

Chapter 1 - In the first chapter, we have discussed the background of domination problems and listed the research problems or what we can improve for domination problems. We have also listed our contributions or what we have done to solve these research problems.

Chapter 2 - We analyze the problem of how to utilize online social network as a medium to improve users' study achievements in e-learning environments. We have proposed a new DS named WPIDS and two WPIDS selection algorithms to evaluate the effect of educating a subset of the entire target group susceptible to a social problem. The simulation experimental results have revealed that the WPIDS model and selection algorithms are more effective than the PIDS one [27].

Chapter 3 - We extend Huang et al.'s work and consider the extension problem of M2DS mentioned in [55, 68]. We have solved the extension problem for general k (i.e., for k being an arbitrary positive integer) in general networks. We have first developed a new self-stabilization algorithm for finding an MKDS in a general network that works under a central daemon. We have also analyzed the correctness and time complexity of the proposed algorithm, in which the time complexity of their algorithm is not been discussed in [68].

Chapter 4 - We have proposed another new self-stabilization algorithm for finding an MKDS in a general network that works under a distributed daemon. We have also analyzed the correctness and time complexity of the proposed algorithm in this chapter.

Chapter 5 - We have presented a self-stabilizing protocol for finding an MWDS in an arbitrary network graph under a central daemon. We have also analyzed the correctness and time complexity of the proposed algorithm in this chapter.

Chapter 6 - We have first presented a self-stabilizing algorithm for the MPIDS problem, which can find an MPIDS in large social networks without any isolated node. We have also proved that the worst case convergence time of the algorithm from any arbitrary initial state is $O(n^2)$ steps where n is the number of nodes in the network.

Chapter 7 - We investigate cooperative cost games arising from the PIDS problem on social network graphs. We have proposed two new game models, the rigid PIDS game and relaxed PIDS game, and focused on their cores. First, a relationship between their cores of both games is obtained. Next, we have proved that the core of the relaxed PIDS game is non-empty if and only if there is no integrality gap for the relaxation linear programming of the PIDS problem on the graph G.

Chapter 8 - The conclusions of the entire thesis are presented in this chapter. Firstly, an overview of the entire thesis is provided followed by the major conclusions arising from this research. Detailed conclusions based on the specific objectives and the key outcomes are presented. Finally, the key areas proposed for further research are identified.

Chapter 2

Weighted Positive Influence Dominating Set

In order to help alleviate a certain social problem, a variation of dominating set, called positive influence dominating set (PIDS) was introduced by Wang et al. [27]. The PIDS can deal with social problems only in undirected social networks for uniform weight cases [69, 70]. In this chapter, we have motivations to extend the PIDS to overcome its drawbacks. For example, the PIDS in existing research does not take into consideration the attributes, directions and degrees of persons' influence [27, 28]. We will propose a novel dominating set called weighted positive influence dominating set (WPIDS) and two WPIDS selection algorithms in e-learning environments. The WPIDS can deal with social problems in weighted directed social networks. Experimental results demonstrate that the proposed WPIDS and algorithms are more reasonable and effective than those of the PIDS without considering the factors of attribute, direction and weight of users' influence in e-learning environments.

The information in this chapter is based on two published papers [69, 70].

2.1 Introduction

As the Internet becomes widespread, e-learning communities have become more and more popular [76–78]. E-learning is an attractive and efficient way for modern education since e-learning environments are more convenient and source saving to build compared with the traditional learning environments. In such learning environments, almost all the resources are provided through the computers and networks and students can learn at anytime or anywhere. Meanwhile, the interaction and collaboration of tutors and students also play important roles in the e-learning programme. Social interaction within an online framework can help students share experiences and collaborate on relevant topics.

Recently, some research has been done to understand the properties of e-learning. Many educators and researchers have proposed their designs, described their implementation and shared their experiences from different points of view on e-learning environments [77–80]. In fact, the relationship between the e-learning users (e-learners) composes an online social network. In e-learning programmes the tutors and students compose the set of e-learners. There are some different studying groups according to their interests and purposes. The fact is that each e-learner has a different learning ability. For instance, an excellent student, an average student, or a poor student in terms of their academic records. An excellent student has positive influence on his direct friends (outgoing neighbours), but he might turn into a poor student and has negative influence on his outgoing neighbours if he is affected by many of his friends who are poor students, and vice versa. Besides, an e-learner can be an authority such as a tutor who has strong positive influence on others. It is very important to divide groups such that there are plenty of tutors or excellent students in each group to have positive influence to help other students.

On the other hand, due to the financial limitations in budget, it is too expensive to set many tutors in the study programme. Therefore, how to choose a subset of individuals to be part of the programme so that the effect of the intervention program can spread through the whole group under consideration becomes an important research problem. These issues are very intricate and complex problems. In an effort to address this issue, the specific problem we study in this chapter is the following: Given an online e-learning social network and the set of e-learners, we identify a subset of the individuals within the e-learning online social network to participate in an education/intervention programme such that the education/intervention can result in a globally positive impact on the other e-learners.

2.2 Motivations and Contributions

In [27], Wang et al. introduced the notion of the PIDS and proposed a greedy approximation PIDS selection algorithm in 2009. Recall that $D \subseteq V$ is a *positive influence dominating set* (PIDS) [27, 28] if any node i in V is dominated by at least $\lceil \frac{d(i)}{2} \rceil$ nodes (that is, i has at least $\lceil \frac{d(i)}{2} \rceil$ neighbours) in D where $d(i)$ is the degree of node i. Note that there are two requirements for PIDS:

(i) Every node not in D has at least half of its neighbours in D;

(ii) Every node in D has at least half of its neighbours in D.

Wang et al. [27] revealed that approximately 60% of the whole group under consideration needs to be selected into the PIDS to achieve the goal that every individual in the community has more positive neighbours than negative neighbours.

Figure. 2.1 is a proper example of PIDS which illustrates the scenario as discussed above. In Fig. 2.1, Bob, Tom, Don and Ann are four equal e-learners in a small learning group. Any three of them form a PIDS satisfying its definition (requirements (i) and (ii)). If Bob is a tutor and has strong positive influence on others, only Bob can positively affect (dominate) others. Another fact is that the degrees of influence between two persons are not always equal. But the PIDS assumes the degrees of influence among them are uniform [27, 28].

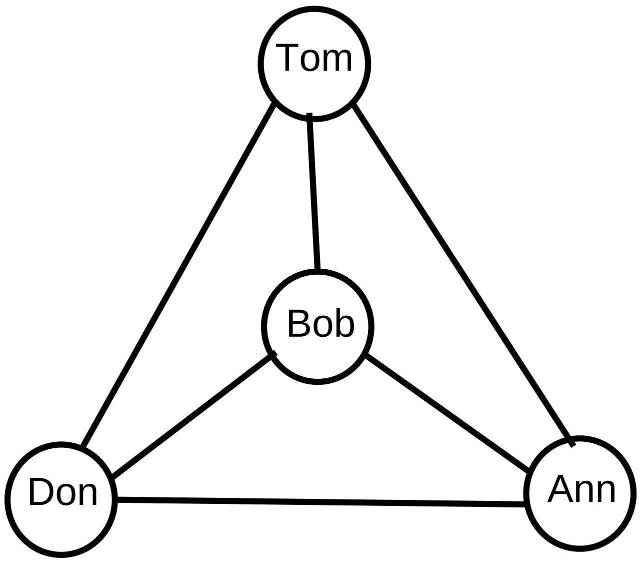

Figure 2.1: An example of a PIDS graph model

As we discussed above, if we consider some key factors in e-learning environments, such as the important persons and degree of each e-learner's influence, the size of the solution for selecting a proper subset of the whole group might be smaller and the algorithm might be more effective and economical. In this chapter we consider the degree and direction of each e-learner's influence in e-learning social networks. We propose a novel dominating set named WPIDS and develop two WPIDS selection algorithms. The effectiveness of our proposed WPIDS model and algorithms are evaluated by experiments.

The main idea of our research is as to how to effectively select positive e-learners to affect an individual in the e-learning social network. An individual becomes a "positive" e-learner if its neighbours' "positive" influence are more than its neighbours' "negative" influence. We give both theoretical justification and empirical verification for the two proposed selection algorithms. Specifically, we prove the feasibility of the two selection algorithms by a theorem. The contributions of our work are as follows:

1. A new dominating set named WPIDS and two WPIDS selection algorithms have been presented. The WPIDS model reasonably utilizes its online social network structure to help e-learningers to improve their study achievements.

2. The effectiveness of the two WPIDS selection algorithms has been evaluated by simulation experiments.

3. The differences between WPIDS and PIDS models and the causes why the WPIDS model is better than PIDS model have been discussed.

2.3 Problem Definition

In this section, we formulate the WPIDS problem arising from the e-learning social networks. We will use the following graph model to illustrate the e-learning social network in the context of the improving study achievement issue: A digraph $G = (V, A, C, W)$ is used to represent the e-learning online social network. V is the set of nodes in which each node is an e-learner in the e-learning social network; A is the set of arcs in which each direct arc represents the existence of a social connection/influence between the two endpoints; C is the compartment vector that saves the compartment of each node. The compartment of a node decides whether it has *positive* or *negative* influence on its outgoing neighbours. For example, for the improving e-learningers' study achievements problem, the compartment of each node is one of the followings: authority (tutor), excellent student, average student, or poor student. A node in the authority or excellent student compartment has positive influence and all nodes in any of the other two compartments have negative influence. W is a set of weight values corresponding to arcs belong to A. Each arc's value is decided by the frequency of the two persons' interactions.

We assume that:

(i) If the total arcs weight of an individual's incoming neighbours has positive impact on him, then the probability that this individual positively impacts others in the social network is high;

(ii) Education/intervention programmes can convert a negative influential individual to a positive influential person;

(iii) There are some authority e-learners (tutors) with no incoming arcs which means that they are positive e-learners without others' influence.

The first assumption comes from an extensive body of evidence suggesting that one of the most powerful predictors of habitual behaviour in individuals is whether an individual has friends who also engage in that behaviour [81–83]. Due to outside competition in terms of personality traits attained from peer influence, the more neighbours/friends exerting positive influence an individual has, the more likely he is to impact others in a positive way. The second assumption comes from the work in [81, 84, 85], where nearly every individual in the feedback intervention programme showed an improving grade in studying. The third assumption comes from the fact that the tutors are authorities in the study programme who cannot be affected by other students' negative influence. With the above three assumptions, the problem is equivalent to selecting a subset of all the e-learners in an e-learning programme such that other e-learners in the social network receive more positive influence than negative influence.

The formal definitions of the WPIDS problem are as follows.

Definition 2.1. *(E-learner Social Network). An e-learner social network is* *a weighted digraph $G = (V, A, C, W)$. V is the set of nodes in which each node is an e-learner in the network; A is the set of arcs between the vertices: $A = \{(u, v)|u, v \in V, u \mapsto v\}$, where $u \mapsto v$ indicates that the e-learner u affects the e-learner v; C is the compartment vector; the compartment of a node decides whether it has positive or negative influence on its outgoing neighbours. W is a set of weight values corresponding to arcs belonging to A. The weight value W of an arc (u, v) is defined as:*

$$\begin{cases} w(u,v) \in [-1, 0) & \text{if the e-learner } u \text{ is a negative e-learner;} \\ w(u,v) \in [0, 1] & \text{if the e-learner } u \text{ is a positive e-learner.} \end{cases}$$

■

Definition 2.2. *(Weighted Positive Influence Dominating Set (WPIDS)). With Definition 2.1, the WPIDS of an e-learner online social network G is defined as a subset $P \subseteq V$ such that any node $u \in V - P$ is positively dominated by P. That is,*

$$\forall u \in V - P, \quad w(u^-) = \sum_{v \in N^-(u)} w(u, v) \geq 0,$$

where $N^-(u) = \{v|(v, u) \in A\}$ is the set of incoming neighbours of node u. ■

The WPIDS problem is to find a so-called minimum WPIDS of G, which minimizes the total number of its vertices. In this chapter, we will propose two selection algorithms for the WPIDS problem and find approximate solutions to the WPIDS problem in large online social networks.

Example 2.1. *An example of the WPIDS is shown in Fig. 2.2. Let node v_2 represent an authority (tutor) and nodes v_1, v_3, v_4 represent negative students, and let $w(v_2, v_1) = w(v_2, v_3) = w(v_2, v_4) = 0.7$, other arcs weight values are -0.3. According to the definition of WPIDS, the total incoming arcs weight values of nodes v_1, v_3, v_4 are 0.1. So the subset $P = \{v_2\}$ is a WPIDS, which shows the key person's influence.*

2.4 Weighted Positive Influence Dominating Set Algorithms

In this section, we present one theorem and two WPIDS selection algorithms for the WPIDS problem formalized in the above section. To do so, we first define a function f as follows.

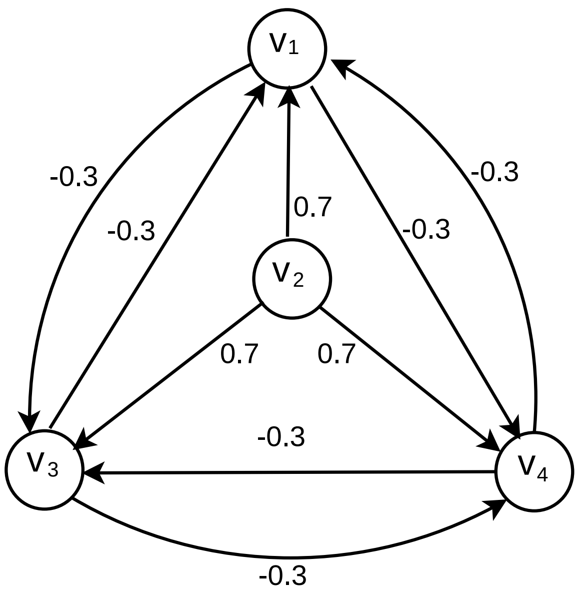

Figure 2.2: An example of a WPIDS graph model with an authority

Definition 2.3. *Consider a weighted digraph $G = (V, A, C, W)$ as an instance of WPIDS. For any vertex subset $P \subseteq V$, we define*

$$f(P) = \sum_{v \in V-P} min\{0, w(n_P^-(v))\},$$

where $w(n_P^-(v)) = \sum_{u \in n_P^-(v)} w(u, v)$, $n_P^-(v) = \{u|(u, v) \in P\}$ denotes the incoming neighbours of $v \in P$. ■

The following theorem states important properties of the function f.

Theorem 2.1.

(1) $f(\emptyset) = 0$.
(2) $\forall v \in V - P, f(P) = 0$, if and only if P is a WPIDS of G.
(3) If $f(P) < 0$, then there exists a vertex $u \in V - P$ such that $f(P \cup \{u\}) > f(P)$.

Proof. Note that $n_\emptyset^-(v) = \emptyset$ for all $v \in V$. Therefore (1) holds.

For (2), we note that $f(P) = 0$ if and only if $0 \leq w(n_P^-(v))$ for every $v \in V - P$ if and only if P is a WPIDS.

To see (3), note that $f(P) < 0$ implies the existence of $v \in V - P$ such that $0 > w(n_P^-(v))$. Let u be an incoming neighbour of v which is not in P and select u into P, then $f(P \cup \{u\}) = \sum_{v \in V} min\{0, w(n_P^-(v)) + w(u, v)\} > f(P)$. ■

Theorem 2.1 is the theoretical analysis to conduct two greedy algorithms for the WPIDS problem formalized in the earlier section. This is very important for running the algorithms in practice.

We define and explain a few terms and definitions used in the description of our algorithms. Let a weighted digraph $G = (V, A, C, W)$ be an instance of WPIDS. Each node of V can have either positive or negative impact on its neighbourhoods. The positive degree of a node v affects an outgoing neighbour u is the positive weight value of $v's$ outgoing arc weight value $w(u, v)$. The same holds for the negative degree. The compartment C of a node decides whether the node is a positive or a negative node. For example, in the e-learning social network, a node in the tutor compartment is a positive node and a node in any other compartment is a negative node. Nodes chosen into the WPIDS are marked positive. Thus an e-learner u is a *positive* e-learner if u is initially a positive node or u is selected into the WPIDS. A PIDS P of a graph G is a subset of nodes in G that any node u in G is dominated by at least $\lceil \frac{d(u)}{2} \rceil$ positive nodes (that is, u has positive neighbours in at least half of its neighbourhood) in P where $d(u)$ is the degree of node u. A WPIDS P of a graph G is a subset of nodes in G such that any node u in $V - P$ is positively dominated by the positive nodes in P (that is, the total influence value of $u's$ incoming neighbours is no less than zero).

We explain the heuristic methods to the two WPIDS selection algorithms as follows:

Algorithm 2.1: Weighted Positive Influence Dominating Set (WPIDS) Selection Algorithm

```
Input: A digraph $G = (V, A, C, W)$ where $V$ is the set of nodes, $A$ is the set of
       arcs that capture the social interaction of the nodes, $C$ is the set of
       nodes that are initially in a positive compartment.
Output: A subset $P \cup C$ of $V$ such that any node $u$ in $V - P \cup C$ has been
        positively dominated by $P \cup C$, i.e., each node $u \in V$, $w(u^-)$ is
        non-negative.
1  let $V' = V - C$ and set the status of nodes in $C$ to POSITIVE;
2  initialize the status of all nodes in $V'$ to NEGATIVE and $P$ to empty;
3  each node $u \in V'$, calculate $w(u^-)$ ;
4  $T$ is the set of nodes in $V'$ that have been positively dominated;
5  let $V' = V' - T$;
6  while not every node in $V'$ has been positively dominated
7      choose the node $u$ with the minimum negative weight value of its total
       outgoing arcs in $V'$ into set $P$, set the status of nodes in $P$ to positive;
       resetting the node $u$ has absolute weight value of outgoing arc weights;
8      update $P = u \cup P$, $V' = V' - T$;
9      update the positive values of each node in $V'$;
10 end of while
11     the nodes in set $V - P \cup C$ are positively dominated in $G$, and the
       positive influence of each node ($w(u^-)$) in $V - P \cup C$ is also calculated.
12 OUTPUT: The nodes in set $P \cup C$ positively dominate set of $V - P \cup C$, and
   the positive value of each node ($w(u^-)$) in $V - P \cup C$ is also calculated.
```

First we need to consider the e-learners (nodes) who are not positively dominated. It is easy to imagine that we can get a more “greedy” algorithm if we select nodes with the biggest outgoing negative influence as dominators into the WPIDS because they have more positive influence on others. Repeat this procedure until all nodes not in the WPIDS are positively dominated by their incoming neighbours. The procedure of this algorithm is presented in Algorithm 2.1.

Considering the fact that the negative e-learners (nodes) who have the highest accumulative negative weights from other neighbours' influence are more easily educated into positive e-learners, we propose a new algorithm. The main idea of WPIDS Selection Algorithm 2.2 is to select the e-learners (nodes) from the negative e-learners group with the highest accumulative weights of incoming arcs into the WPIDS according to the fact that these selected e-learners are more easily educated into positive e-learners. The procedure of this algorithm is presented in Algorithm 2.2.

Algorithm 2.2: Weighted Positive Influence Dominating Set (WPIDS) Selection Algorithm

```
Input:  A digraph $G = (V, A, C, W)$ where $V$ is the set of nodes, $A$ is the set of
        arcs that capture the social interaction of the nodes, $C$ is the set of
        nodes that are initially in a positive compartment.
Output: A subset $P \cup C$ of $V$ such that any node $u$ in $V - P \cup C$ has been
        positively dominated by $P \cup C$, i.e., each node $u \in V$, $w(u^-)$ is
        non-negative.
1  let $V' = V - C$ and set the status of nodes in $C$ to POSITIVE;
2  initialize the status of all nodes in $V'$ to NEGATIVE and $P$ to empty;
3  each node $u \in V'$, calculate $w(u^-)$ ;
4  $T$ is the set of nodes in $V'$ that have been positively dominated;
5  let $V' = V' - T$;
6  while not every node in $V'$ has been positively dominated
7      choose the node $u$ with the maximum negative weight value of its total
       incoming arcs in $V'$ into set $P$, set the status of nodes in $P$ to positive;
       resetting the node $u$ has absolute weight value of outgoing arc weights;
8      update $P = u \cup P$, $V' = V' - T$;
9      update the positive values of each node in $V'$;
10 end of while
11     the nodes in set $V - P \cup C$ are positively dominated in $G$, and the
       positive influence of each node $(w(u^-))$ in $V - P \cup C$ is also calculated.
12 OUTPUT: The nodes in set $P \cup C$ positively dominate set of $V - P \cup C$, and
   the positive value of each node $(w(u^-))$ in $V - P \cup C$ is also calculated.
```

The difference between these two selection algorithms is that Algorithm 2.1 is to select e-learners with the biggest outgoing negative influence as dominators and Algorithm 2.2 is to select e-learners who are easy to be changed into positive e-learners as dominators. The procedures of these two WPIDS selection algorithms are described in Algorithms 2.1 and 2.2 respectively.

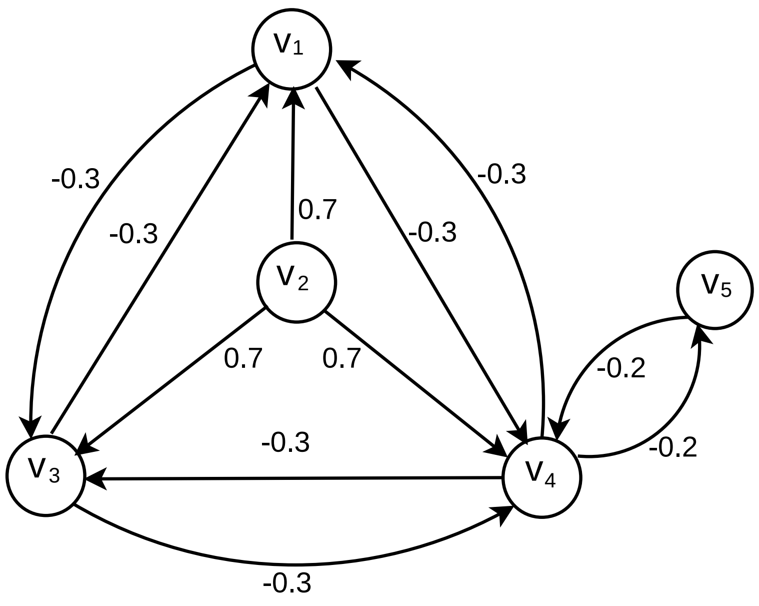

Figure 2.3: An example of a WPIDS selection algorithm graph model

Example 2.2. *Figure. 2.3 shows how to operate our two WPIDS selection algorithms. Figure. 2.3 is almost the same as Fig. 2.2 except one more negative node* v_5 *and let* $w(v_4, v_5) = w(v_5, v_4) = -0.2$*. According to Algorithm 2.1, the nodes* v_1 *and* v_3 *have already been positively dominated by the node* v_2*. So we just consider the nodes* v_4 *and* v_5*. The node* v_4 *has the smallest total outgoing arcs weight value (-0.8) and the node* v_5 *has total outgoing arcs weight values as -0.2, so the node* v_4 *is selected as a positive node. The arc weight* $w(v_4, v_5)$ *becomes 0.2 which can positively dominate the node* v_5*. Consequently, the set* $\{v_2, v_4\}$ *is a WPIDS, which can positively dominate the whole set of nodes. According to Algorithm 2.2, the node* v_4 *has the biggest total ingoing arcs weight value (-0.1) and the node* v_5 *has total ingoing arcs weight values as -0.2, so the node* v_4 *is selected as a positive node. The arc weight* $w(v_4, v_5)$ *becomes 0.2, which can positively dominate the node* v_5*. The set* $\{v_2, v_4\}$ *is a WPIDS of Fig. 2.3.*

2.5 Experimental Evaluations

In order to clearly reveal the effectiveness of our proposed WPIDS methods, we designed a programme to simulate e-learning environment. In this way, we can clearly predefine the ground truth of each e-learner's attribute to test the efficiency of our greedy WPIDS selection algorithms. In this section, we will discuss the experiments of WPIDS selection algorithms on data generated from the programme.

The evaluation is designed to answer the following questions:

a) What is the difference between the size of WPIDS and PIDS and what is the difference between the influences of these two sets?

To answer the first questions, we compare the size of dominating sets [36] between PIDS and WPIDS. Generally speaking, the smaller the size of the dominating set, the more effective the algorithm.

b) How many nodes need to be selected into the WPIDS and how influential these nodes can be?

To answer the second question, we calculate the average total incoming arcs' weight of each node (called positive influence value) [86] to measure influence. The higher the average positive influence value is, the more the influential of the WPIDS can be; and the possibility of the whole community turning into a positive community is higher.

c) what is the difference of the performance of our two greedy WPIDS selection algorithms in the WPIDS problem?

In order to test the third question, we compare the evaluation of these two WPIDS selection algorithms by parameter selection respectively.

2.5.1 Simulation Program Design

The experiment coded was implemented by standard Java and conducted on a Windows XP server with a 2.4GHz Intel Dual Core processor, 4GB of RAM, and 2TB SATA HD RAID (Level 5).

We conducted experimental evaluations of the proposed method on the data set generated from simulations. Two types of e-learners are defined in the simulation including: P Type (positive e-learners or tutors), N Type (neutral e-learners or students). P Type of e-learners have three characteristics as follows.

1) All P Type e-learners have outgoing arcs. They are positive e-learners or tutors without others' influence during the process.

2) All P Type e-learners have more outgoings arcs than N Types e-learners. They are the very active e-learners or instructor presenting distance learning courses.

3) A P Type e-learner has higher total influence degree than an N Type e-learner. N Type e-learners are neutral e-learners or students whose neighbours and influence degree are smaller than P Type e-learners. They are assigned randomly.

In the designed simulation programme, we simulated 300 e-learners and ran the programme 100 cycles. Among these 300 e-learners, in the light of the e-learning course of USQ standard, we used a parameter θ to determine the percentage of positive e-learners. For example, $\theta = 5\%$, 10%, or 15%, would make 15 P Type e-learners, 30 P Type e-learners or 45 P Type e-learners respectively,

2.5.2 Evaluation Metrics

We used the size of dominating sets [36] as one of the parameters to evaluate the effectiveness of WPIDS selection algorithms. The size of WPIDS is the proportion of selected e-learners in the whole e-learners group, i.e.,

$$Size = \frac{|WPIDS|}{|V|}, \tag{2.1}$$

The smaller the size of dominating sets, the more effective the algorithm.

We used the Average Positive Influence Value of e-learners (APIV) [86] as the other evaluation parameter, which implies the possibility of the whole community turning into a positive community. The APIV of e-learners is calculated as:

$$APIV = \frac{\sum_{v \in V-P} w(n^{-}(v))}{|V-P|}, \tag{2.2}$$

where $n^{-}(v) = \{u|(u,v) \in V\}$ denotes the incoming neighbour nodes of v in V.

The high APIV of e-learners reflects the high possibility of the whole community turning into a positive community.

2.5.3 Experimental Results

Table 2.1: Result of the simulation program by WPIDS selection algorithm 2.1

P Type (percentage)	Size of WPIDS (percentage)	APIV
15(5%)	52(17.3%)	0.218
30(10%)	67(22.3%)	0.212
45(15%)	79(26.3%)	0.211

Table 2.1 illustrates the result of the simulation experiments for WPIDS Selection Algorithm 2.1. The size of the e-learning group is 300 e-learners, we performed the simulation experiments with three different settings of the parameter $\theta =$ 5% (15 P Type e-learners), 10% (30 P Type e-learners), and 15% (45 P Type e-learners). The result in Table 2.1 is the average over 100 runs. Applying WPIDS Selection Algorithm 2.1, the experiment results show that when $\theta = 5\%$ the model has the best performance: 17.3% (52 P Type e-learners are selected) of the e-learners (nodes) are selected into the WPIDS (the smallest) and the resulting APIV is 0.218 (the biggest), which means it is more effective to positively dominate the whole group. For the PIDS Selection Algorithm [27], 58.2% of the nodes are selected into the dominating set, which is more than three times as big as our WPIDS Selection Algorithm 2.1. Another observation is that the WPIDS Selection Algorithm 2.1 displays a tendency that the smaller the parameter θ, the better WPIDS can be obtained (for both the two evaluation parameters). Based on experiences, this tendency is not monotonic, there should be a turning point.

Table 2.2 illustrates the result of the simulation programme under similar settings (the same e-learning group size (300 nodes) and the similar parameter θ (5%, 10%, 15%)) by WPIDS Selection Algorithm 2.2. On the contrary, it is clear shown in

Table 2.2: Result of the simulation program by WPIDS selection algorithm 2.2

P Type (percentage)	**Size of WPIDS (percentage)**	**APIV**
15(5%)	76(25.3%)	0.306
30(10%)	74(24.7%)	0.324
45(15%)	68(22.7%)	0.324

Table 2.2 that $\theta = 15\%$ gives the best performance: 22.7% (68 P Type e-learners, the smallest) of the nodes are selected into the dominating set which is also much smaller than 58.2% (PIDS Selection Algorithm [27]), and the resulting APIV is 0.324 (the biggest). From Table 2.2 we can also see a tendency that the bigger the parameter θ, the better result can be achieved (for both the two evaluation parameters).

In summary, from Tables 2.1 and 2.2, we can see that the size of WPIDS by Algorithm 2.1 is smaller than that of Algorithm 2.2 when the parameter θ is small ($\theta = 5\%$ and $\theta = 10\%$). This result meets our conjecture. Because Algorithm 2.1 is firstly to select the persons who have the more influence among the users as positive dominators and it is more "greedy" than Algorithm 2.2. So it may find a WPIDS much smaller and faster than Algorithm 2.2. Another experiment result is that the APIV of Algorithm 2.1 is smaller than that of Algorithm 2.2 respectively. This phenomenon contradicts our conjecture. Heuristically, to select persons with the biggest negative influence into positive ones in Algorithm 2.1 can provide a bigger positive influence than to select persons with the smallest negative influence in Algorithm 2.2. An explanation is that it is more effective to change persons who are easily changed than to change persons who are hard changed in an education/intervention programme. This interesting result needs further investigation.

Table 2.3: Comparison between WPIDS and PIDS selection algorithms

Algorithm (percentage)	**Size of DS (percentage)**	**Average**
WPIDS Selection Algorithm 2.1	26.3%	0.211 (APIV)
WPIDS Selection Algorithm 2.2	22.7%	0.324 (APIV)
PIDS Selection Algorithm	58.2%	22.5 (degree)

Table 2.3 illustrates the different size of dominating sets which are got by running PIDS Selection Algorithm in [27] and our WPIDS Selection Algorithms in online social networks. We set $\theta = 15\%$ which is the same percentage as in Wang et. al's work [27]. The results in Table 2.3 are the average over 100 runs. Note that for each run, the size and average degree are consistent. The greedy WPIDS Selection Algorithms have better performance over the greedy PIDS Selection Algorithm in [27]. Applying WPIDS Selection Algorithm 2.1, 26.3% of the nodes are selected into the dominating set and the resulting APIV is 0.211. Furthermore, applying WPIDS Selection Algorithm 2.2, 22.7% of the nodes are selected into the dominating set and the resulting APIV is 0.324. Both evaluation parameters are obviously improved compared with WPIDS Selection Algorithm 2.1. For PIDS Selection Algorithm, 58.2% of the nodes are selected into the dominating set and the resulting average positive degree is 22.5 [27]. In terms of the comparison in Table 2.3, the probability of positive influencing the whole community

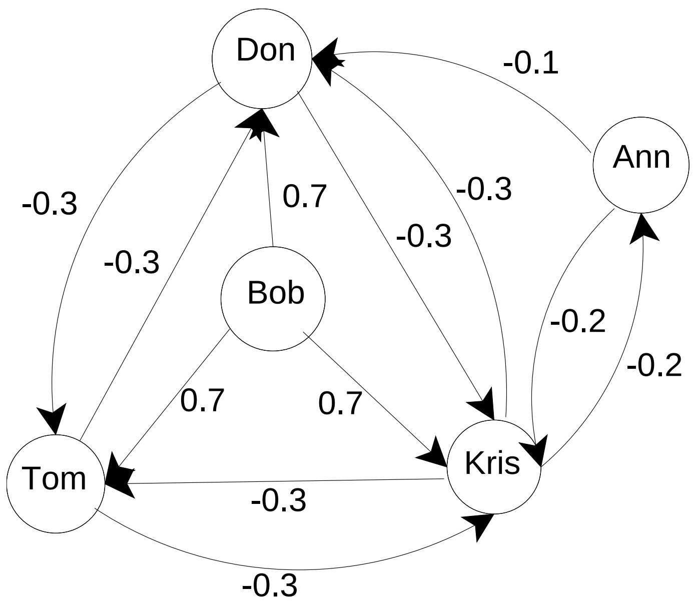

Figure 2.4: A case study of a WPIDS selection algorithm group model

through an intervention programme where participants selection is determined by the WPIDS Selection Algorithm is significantly higher by moderately considering the key persons' positive effect and increasing the participation related cost.

2.6 Case Study

In this section we use a study group Fig. 2.4 to discuss the differences between WPIDS and PIDS selection algorithms. The study case is depicted in Fig. 2.4. The scenario undergraduates are picked from a computer science class[1], where Bob is a tutor and other persons are non-positive students in a small e-learning group. Ann is a quiet and introverted girl who has limited influence on others. In Fig. 2.4, the arcs weight values are the same as in Fig. 2.3 and Ann has -0.1 influence value on Don. According to our WPIDS definition, Bob and Kris are a WPIDS. If we get rid of the arc's weights and directions, Don and Kris are a PIDS according to its definition [27]. The difference between these two solutions is that the solution to WPIDS select Bob as one of the dominators and considers the fact Bob is a tutor which should be as a key person who plays an important role in the programme. But the solution to PIDS does not. Another difference is our graph model is directed. The influence of relationship between e-learners is different, whereas the influence between e-learners in PIDS model is uniform [27].

[1]From Faculty of Science, University of Southern Queensland, Australia. For privacy reasons, the students have been renamed.

2.6.1 Discussions

So from the Case Study of Fig. 2.4, we can see that the WPIDS in our model has some different real meanings from the PIDS [27]. One of our improvements is that we consider the reasonable partition of persons who attended the programme. Either in e-learning programme or drinking (smoking or drug use) intervention strategies and programmes we should consider the authorities' effect such as tutors or correctional officers who have strong positive influence on other persons and without interference from others. In other words, they are the key persons who play important roles in the programme. For example, in the above Case Study Bob is a tutor who should be a key person and as one of the dominators in the programme. But the PIDS model has a different conclusion of the Case Study. Tom's role is the same as Bob's. Tom can be a replacement dominator of Bob in any solution. This does not meet the fact that Bob is a tutor whereas Tom is a student.

Due to Ann's special situation, Kris should be selected into WPIDS to positively influence Ann instead of other persons. This solution shows the importance of effective directions. For example, the influence of relationship between tutors and students can be considered one-way. The fact is that the degree of influence between two persons is different and should be considered. Besides, our definition of the dominating problem is more reasonable than that of PIDS since one person's neighbours' level of influence is the decisive factor instead of the number of one's positive neighbours. So, in order to positively dominate the whole group we can strengthen influence on the key persons instead of increasing the number of positive persons. Through these discussions we can draw the conclusion our WPIDS model is more reasonable and effective than PIDS model [27].

2.7 Summary

In this chapter, we have introduced and studied the problem of how to utilize online social network as a medium to improve e-learners' study performances in e-learning environments. We have proposed an innovative dominating set (WPIDS) and two WPIDS Selection Algorithms (Algorithms 2.1 and 2.2) to evaluate the effect of educating a subset of the entire target group susceptible to an e-learning issue. Our idea is one person's neighbours' level of influence is the decisive factor instead of the number of its positive neighbours. The experimental results have demonstrated that the WPIDS model and selection algorithms are more effective than the PIDS one [27,28], especially to deal with the large social network cases. The main reason is that the WPIDS model considers the key persons who play important roles in the community. So the size of WPIDS is smaller than that of the PIDS one in an online social network. Moreover, the Case Study has further revealed that the WPIDS model and algorithms are more effective than those of the PIDS ones [27].

For the further research work of the WPIDS problem, an interesting research direction is that the influence of each individual in the system will have changed over a period of time. So applying a mathematical model to understand the dynamic change of the WPIDS model is very attractive.

Chapter 3

Self-Stabilization and K-Dominating Set

In this chapter, the extension problem, "To find a self-stabilizing algorithm for the minimal 2-dominating set (M2DS) problem in general networks under the central daemon model" which was studied by Huang et al. [55, 68] is presented. In the past, the existing proposed self-stabilizing algorithms for the minimal k-dominating set (MKDS) either work for tree graphs (Kamei and Kakugawa [26]) or find an M2DS (Huang et al. [55, 68]). We have the motivation to extend the M2DS from 2 to general k (MKDS). We propose a self-stabilizing algorithm for the MKDS under a central daemon model when operating in any general network. We further prove that the worst case convergence time of the algorithm from any arbitrary initial state is $O(n^2)$ steps where n is the number of nodes in the network.

The information in this chapter is based on one published paper [71].

3.1 Introduction

In this chapter, we first discuss a classical approach for distributed systems: *Self-stabilization*, which was introduced by Dijkstra in [56, 57]. Self-stabilization is a paradigm for distributed systems that allows the system to achieve a desired global state, even in the presence of faults [56, 57]. It is an optimistic fault tolerance approach for distributed systems. Before introducing Self-Stabilization we first present the conception of distributed systems.

A *distributed system* consists of a set of loosely connected processes that do not share a common or global memory. Each process has one or more shared registers and possibly some non-shared local variables, the contents of which specify the local state of the process. Local states of all processes in the system at a certain time constitute the global configuration (or, simply, configuration) of the system at that time. The main restriction of a distributed system is that each process in the system can only access the data (i.e., read the shared data) of its neighbours. Since a distributed algorithm is an algorithm that works in a distributed system, it cannot violate this main restriction. Depending on the purpose of a distributed system, a global criterion for the global configuration is defined. Those global configurations satisfying the criterion are called *legitimate configurations*, whereas other global configurations are called *illegitimate configurations*. A goal of a distributed system is that the system should function correctly in spite of intermittent faults. When the system is in a legitimate configuration, the purpose of the system is fulfilled. Ideally the global state of the system should remain in a legitimate state. Often, however, malfunctions or perturbations bring the system to some illegitimate state, and it is desirable that the system be automatically brought back to a legitimate state without the interference of an external agent.

A distributed system can be modeled as a simple undirected graph. The topology of a distributed system can be represented as an undirected graph $G = (V, E)$ (called the system's communication graph) with nodes set V and edges set E, where the nodes represent the processes and the edges represent the interconnections between the processes. Throughout this chapter, we denote by n the number of nodes ($|V| = n$), and by m the number of edges ($|E| = m$) in the graph G. Let $i \in V$ be a node; then $N(i)$, its *open neighbourhood*, denotes the set of nodes to which i is adjacent. Every node $j \in N(i)$ is called a neighbour of node i. We denote by $d(i)$ the number of neighbours of node i, or its degree ($d(i) = |N(i)|$). Throughout this chapter we assume G is connected and $n > 1$.

3.2 Self-Stabilization

Self-stabilization can be used to solve the Dominating Set problem, which is introduced by Dijkstra [56, 57], is the most inclusive approach to fault tolerance in distributed systems. According to Dijkstra's work, a distributed system is *self-stabilizing* if it can start at any possible global configuration and regain consistency in a finite number of steps by itself without any external intervention and remains in a consistent state [56, 57]. After having been neglected for nearly a decade, Dijkstra's work was drawn to public attention by Lamport in his invited address at PODC [87]. Since then, the research on self-stabilizing systems has flourished, and a great number of papers regarding self-stabilizing algorithms have been published. Recently, some self-stabilizing algorithms for Dominating Set, Independent Set, Colorings, and Matchings in graphs have been developed [54,61,64,88].

In a self-stabilizing algorithm, each process maintains its local variables, and can make decisions based on the knowledge of its neighbours' states. A process changes its local state by making a *move* (a change of local state). The algorithm is a set of rules of the form "*if condition part (or guard)*" then "*action part*". The condition part (or guard) is a Boolean function over the states of the process and its neighbours; the action part is an assignment of values to some of the process's shared registers. A process i becomes *privileged* if it's condition is true. When a process becomes privileged, it may execute the corresponding move.

A fundamental idea of self-stabilizing algorithms is that the distributed system may be started from an arbitrary global state. After finite time the system reaches a correct global state, called a *legitimate* or *stable* state. An algorithm is *self-stabilizing* if, when the system executes the algorithm,

(i) for any initial illegitimate state it reaches a legitimate state after a finite number of node moves (convergence), and

(ii) for any legitimate state and for any move allowed by that state, the next state is a legitimate state (closure).

The convergence property ensures that, starting from any incorrect state, the distributed system reaches a correct state. The closure property ensures that, after convergence, the system remains in the set of correct states.

Various execution models have been suggested for developing self-stabilizing algorithms. These models are encapsulated within the notion of a daemon (or scheduler). Daemons are one of the most central yet less understood concepts in self-stabilization. Most of these papers adopt Dijkstra's computational model, which is generally referred to as the central daemon model [56, 89, 90]. Under a *central daemon*, only one process can execute an atomic step at one time. In a *central daemon*, we assume a serial model in which no two processes move simultaneously. If two or more processes are privileged, one cannot predict which one will move next. Multiple protocols exist [58, 91, 92] that provide such a scheduler.

Another popular scheduler is the *distributed daemon*, which selects a subset of the system processes to execute an atomic step at the same time. The *distributed daemon* activates the processors by repeatedly selecting a set of processors and activating them simultaneously to execute a computation step. Each processor executes the next computation step as defined by its state just prior to this activation. Once every processor in the set has finished reading, all the processors write a new state (change state). Only then does the scheduler choose a new set of processors to be activated. Note that no non-activated processor changes its state. Thus if a system is self-stabilizing under the distributed daemon model, then it is self-stabilizing under the central daemon model. The converse, however, is not true (the total Dominating Set algorithm in [58] and the 2-Dominating Set algorithm in [55] are self-stabilizing under the central daemon models, but not under the distributed daemon models).

If a *synchronous daemon* is supposed, then all the system processes will execute an atomic step at the same time. The synchronous daemon plays a very important part in the self-stabilization literature. First introduced by Herman [93] to enable analytical tractability of probabilistic self-stabilizing protocols, it was later used in domination problems [94, 95].

The *stabilization time* is the maximum amount of time it takes for the system to reach a correct state. When estimating the time complexity of a self-stabilizing algorithm, we consider the stabilization time, since a self-stabilizing algorithm is usually a do forever loop and it does not terminate. The time complexity is estimated in terms of *process atomic steps* (or simply *steps*), or in terms of *rounds.* A *round* is usually defined to be the minimum time period in which every process is scheduled to execute an action at least once.

A distributed algorithm is said to be *uniform* if all of the individual processes run the same code. If all processes run the same code except a single process,

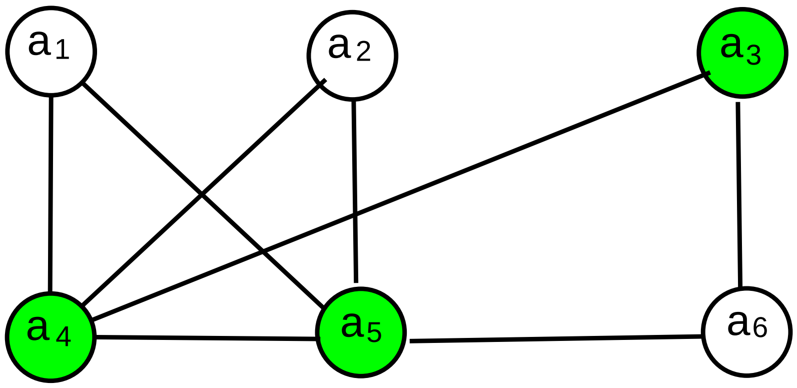

Figure 3.1: A M2DS example

then the algorithm is *semi-uniform*. Some algorithms are designed for *anonymous* systems, where the processes do not have any identification [71,96], whereas other algorithms assume that processes have globally unique identifiers [55,58,68,70,73].

3.3 *K*-Dominating Set Problem

The k-dominating set (KDS) was introduced by Fink and Jacobson in 1985 [97]. Jacobson and Peters showed that the k-domination problem is NP-complete for general graphs [98].

The KDS has many practical applications. For example, to build some fire stations in an urban area with a limited budget we need to ensure that its six areas can receive help from at least k times services from its neighbour areas. Assume there is also a fixed cost for building a fire station in each area. The problem can be interpreted as to determine the number of the fire stations such that the total building cost is minimum among the participating areas under the k times services condition. This problem is equivalent to the problem of finding a minimum weighted KDS among these areas. The following is the formal definition of KDS problem.

Let $G = (V, E)$ be a simple connected undirected graph with vertex set V and edge set E. For a positive integer k, a *k-dominating set* (KDS) of a graph G is a subset $D \subseteq V(G)$ such that every vertex not in D is dominated by at least k vertices in D. The *k-domination number* $\gamma(G)$ of G is the minimum size of a KDS of G. The k-domination problem is to determine a minimum KDS of a graph. The special case when $k = 1$ is the ordinary domination (single domination). On the complexity side of the k-domination problem, Jacobson and Peters showed that the k-domination problem is NP-complete for general graphs [98] and gave linear-time algorithms to compute the $\gamma(G)$ of trees and series-parallel graphs [98].

A KDS D in G is *minimal* if any proper subset of D is not a KDS in G. The so-called minimal KDS problem is to find a minimal KDS (MKDS) in G. Note that an MKDS is minimal not minimum.

For example, without loss of generality, considering the case $k = 2$, a 2-dominating set (2DS) in the urban area as remarked on early, which has 6 locations (a_1, a_2,

..., a_6) and the neighbourhood relations are shown in Fig. 3.1 . We can select the locations $\{a_3,\ a_4,\ a_5\}$, which is a minimal 2DS.

3.4 Motivations and Contributions

Graph algorithms have natural applications in networks and distributed systems, since a distributed system can be modeled with an undirected graph. Due to Dijkstra's pioneering work [56,57], some self-stabilizing algorithms for graph problems have been proposed in the literature, such as the self-stabilizing algorithms for Dominating Set [25, 58, 68], Independent Set and Matching in graphs [54, 88]. Dominating Set and related problems are considered to be of central importance in combinatorial optimization and have been the object of much research. Due to the NP-complete of domination problems [37], researchers have developed some self-stabilizing algorithms for finding a minimal dominating set (MDS).

A self-stabilizing algorithm for the maximal independent set problem can be viewed as a self-stabilizing algorithm for the MDS problem since any maximal independent set in a graph is a MDS in that graph. Hedetniemi et al. [54] presented two *uniform* algorithms (a distributed algorithm is said to be *uniform* if all of the individual processes run the same code) for the DS and the MDS problems. The algorithms all work for any connected graph and assume a *central daemon* (only one process can execute an atomic step at one time). Goddard et al. [88] proposed another uniform algorithm for finding an MDS in an arbitrary graph under a *distributed daemon*, which selects a subset of the system processes to execute an atomic step at the same time.

Alternatively, some self-stabilized algorithms have been proposed in the multiple domination case. Kamei and Kakugawa [26] presented two self-stabilizing algorithms for the MKDS problem in a tree graph. The first uniform algorithm works under a central daemon and the second algorithm works under a distributed daemon. Huang et al. [55, 68] presented two self-stabilizing algorithms to find an M2DS in an arbitrary graph. The first algorithm works under a central daemon with linear time complexity [55] and the second algorithm works under a distributed daemon [68] respectively. For a more detailed discussion of self-stabilizing algorithms for the DS problem, refer to a survey paper [99].

However, so far, there is no algorithm for the MKDS problem in an arbitrary graph that works under a central daemon. The proposed algorithms for the MKDS problem either work for tree graphs (Kamei and Kakugawa [26]) or find an M2DS (Huang et al. [55,68]). In this chapter, we are considering the extension problem of M2DS just mentioned in [55,68]. We will solve that extension problem for general k − MKDS (i.e., for k being an arbitrary positive integer) in general networks.

The following are the contributions.

1. A self-stabilizing algorithm is presented for finding an MKDS under a central daemon in an arbitrary connected simple undirected graph.

2. The correctness of the proposed algorithm is verified; the worst case convergence time of the algorithm from any arbitrary initial state is proved to be $O(n^2)$ steps where n is the number of nodes in the network graph.

3.5 Minimal K-Dominating Set Algorithm

The distributed system in consideration has a general underlying topology, and can be modeled by a connected simple undirected graph $G = (V, E)$, with each node $i \in V$ representing a processor in the system and each edge $(i, j) \in E$ representing the bidirectional link connecting processors i and j. It is assumed that the number of all processors in G is denoted by n. Assume now that for each processor $i \in V$, the set $N(i)$ represents its *open neighbourhood*, denotes the set of processors to which i is adjacent. $d(i)$ represents the number of neighbours of processor i, or its degree $(d(i) = |N(i)|)$.

Definition 3.1. *A subset D of V is a* ***k-dominating set*** *(KDS) in G if each node not in D has at least k neighbours in D.*

A KDS D in G is ***minimal*** *if any proper subset of D is not a KDS in G.*

The so-called "*minimal KDS problem*" (MKDS) is to find an MKDS in G.

The concept of the MKDS problem raises another problem:

Minimal K-Dominating Set

INSTANCE: A connected simple undirected graph $G = (V, E)$ and an arbitrary positive integer k.

QUESTION: How to find an MKDS $D \subseteq V$ such that the D is a KDS of the graph G and minimal, i.e., to construct $D = \{i \in V | x(i) = 1\}$[1] as an MKDS?

In the following, we will propose a self-stabilizing algorithm for finding an MKDS in general networks.

Assume further that each node i in the G has a Boolean flag $x(i)$, whose value is either 0 or 1. At any given time, we will denote with D the current set of nodes i with $x(i) = 1$. That is,

$$x(i) = \begin{cases} 0 & \text{if } i \notin D \\ 1 & \text{if} \in D. \end{cases} \tag{3.1}$$

[1]Refer to Definition of $x(i)$ in Equation 3.1.

Definition 3.2. *$X(i)$ denotes the set $\{j \in N(i) | x(j) = 1\}$, and $|X(i)|$ denotes the cardinality of $X(i)$.*

Note $X(i)$ is the set of $i's$ neighbours in D, that is $N(i) \cap D$.

The minimal k-dominating set algorithm:

```
Algorithm 3.1: Finding a Minimal K-Dominating Set
  Input: A graph $G = (V, E)$, $\forall i \in V$, a Boolean flag $x(i)$, $k$ is a positive
  integer.
  Output: $D = \{i \in V | x(i) = 1\}$.
  R1:  $x(i) = 0 \wedge |X(i)| \leq k - 1 \rightarrow x(i) = 1$
  R2:  $x(i) = 1 \wedge |X(i)| \geq k \rightarrow x(i) = 0$.
```

The Algorithm 3.1 consists of two rules (R1 and R2). Assume k being an arbitrary positive integer. In Algorithm 3.1 each node i has a Boolean variable $x(i)$ indicating membership in the set D that we are trying to construct. The value $x(i) = 1$ indicates that $i \in D$, while the value $x(i) = 0$ indicates that $i \notin D$ as Equation 3.1 describes. The set $X(i)$ is the set of $i's$ neighbours in D and $|X(i)|$ denotes the cardinality of $X(i)$. The variable $|X(i)|$ is designed to count $N(i) \cap D$. Thus, a node i is privileged if it holds R1 or R2, and then executes the corresponding move. An example to illustrate the execution of Algorithm 3.1 is shown in Fig. 3.2 and Fig. 3.3.

R1 says that a node i should enter D if it has fewer than k neighbours in D. R2 says that a node i should leave D if it is dominated by more than k neighbours in D.

3.5.1 An Illustration of a Minimal 3-Dominating Set

Example 3.1. *The example in Fig. 3.2 and Fig. 3.3 is to illustrate the execution of Algorithm 3.1. Without loss of generality, we consider the case $k = 3$ and use a minimal 3-dominating set example to illustrate the execution of Algorithm 3.1. Note that in each configuration, the shaded nodes represent privileged nodes, whereas the shaded node with a darkened circle represents the privileged node selected by the central daemon to make a move.*

In the last configuration (Fig. 3.4), which is a legitimate configuration of Fig. 3.2 and Fig. 3.3, we can see a minimal 3-Dominating Set $D = \{a, c, d, e, g, h\}$ can be identified (the shaded nodes).

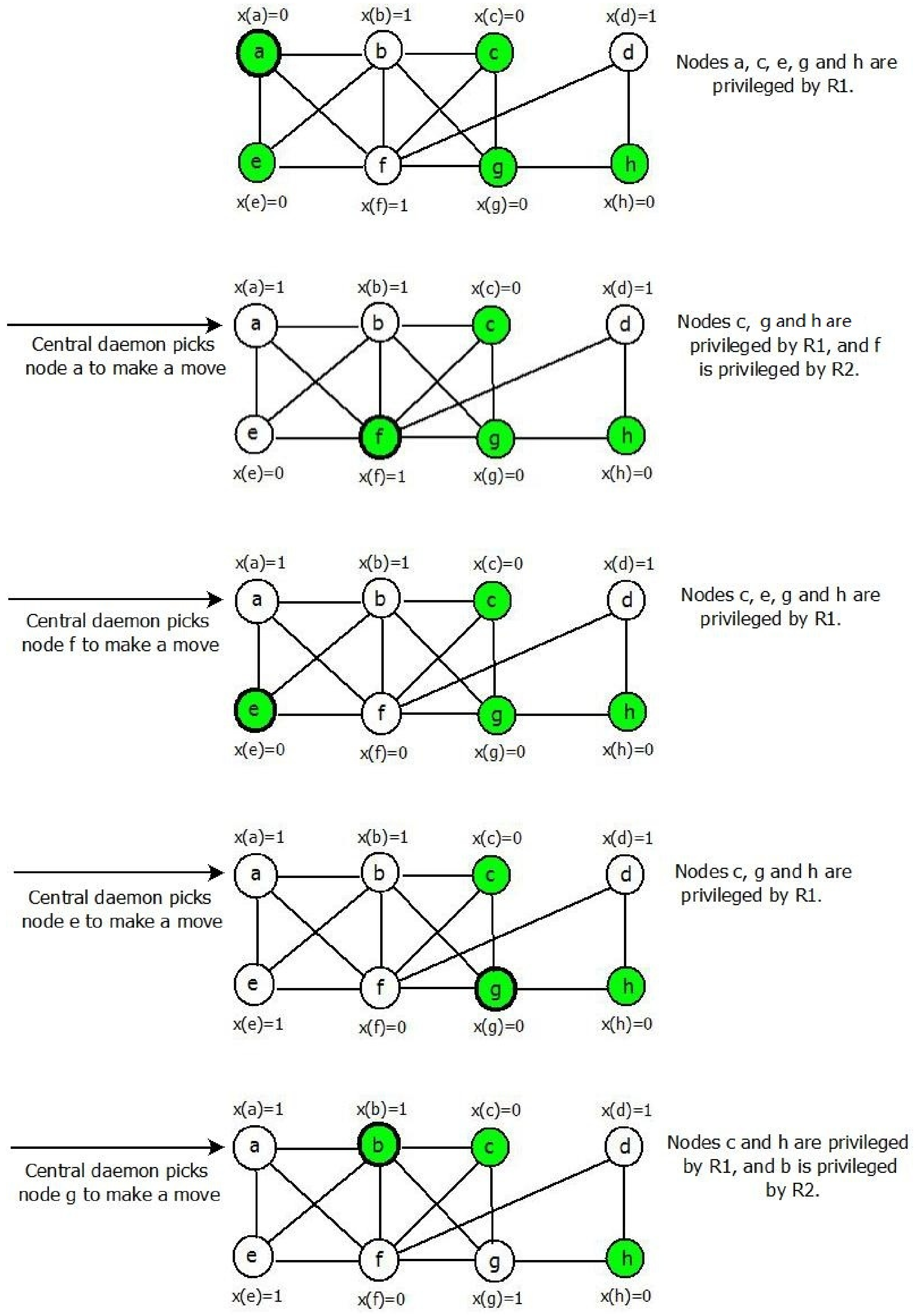
x(a)=0
x(b)=1
x(c)=0
x(d)=1
a
b
c
d
e
f
g
h
x(e)=0
x(f)=1
x(g)=0
x(h)=0
Nodes a, c, e, g and h are privileged by R1.
Central daemon picks node a to make a move
x(a)=1
x(b)=1
x(c)=0
x(d)=1
x(e)=0
x(f)=1
x(g)=0
x(h)=0
Nodes c, g and h are privileged by R1, and f is privileged by R2.
Central daemon picks node f to make a move
x(a)=1
x(b)=1
x(c)=0
x(d)=1
x(e)=0
x(f)=0
x(g)=0
x(h)=0
Nodes c, e, g and h are privileged by R1.
Central daemon picks node e to make a move
x(a)=1
x(b)=1
x(c)=0
x(d)=1
x(e)=1
x(f)=0
x(g)=0
x(h)=0
Nodes c, g and h are privileged by R1.
Central daemon picks node g to make a move
x(a)=1
x(b)=1
x(c)=0
x(d)=1
x(e)=1
x(f)=0
x(g)=1
x(h)=0
Nodes c and h are privileged by R1, and b is privileged by R2.

Figure 3.2: A M3DS example to illustrate the execution of Algorithm 3.1.

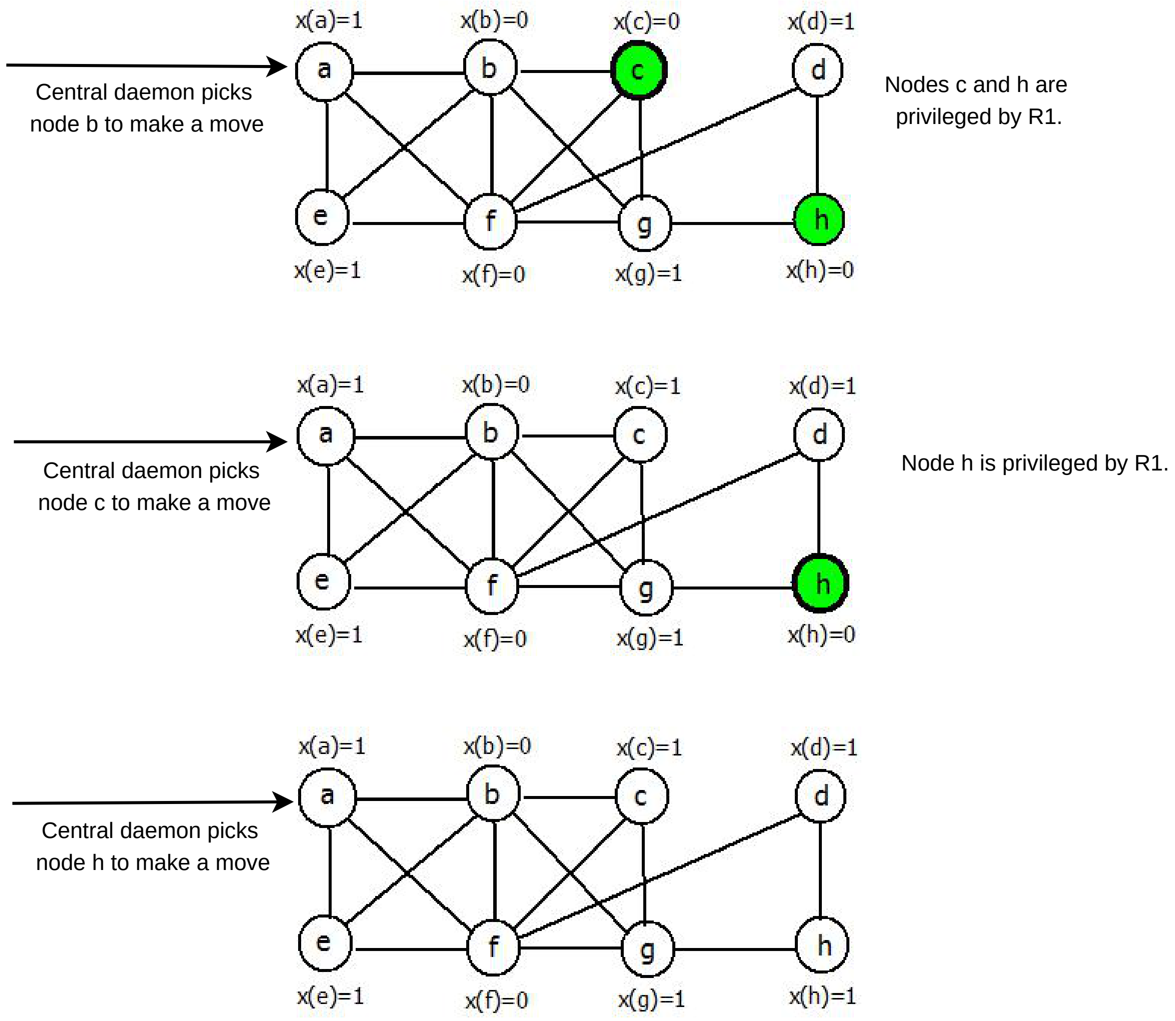
x(a)=1
x(b)=0
x(c)=0
x(d)=1
Central daemon picks node b to make a move
Nodes c and h are privileged by R1.
x(e)=1
x(f)=0
x(g)=1
x(h)=0
x(a)=1
x(b)=0
x(c)=1
x(d)=1
Central daemon picks node c to make a move
Node h is privileged by R1.
x(e)=1
x(f)=0
x(g)=1
x(h)=0
x(a)=1
x(b)=0
x(c)=1
x(d)=1
Central daemon picks node h to make a move
x(e)=1
x(f)=0
x(g)=1
x(h)=1

Figure 3.3: A M3DS example to illustrate the execution of Algorithm 3.1.

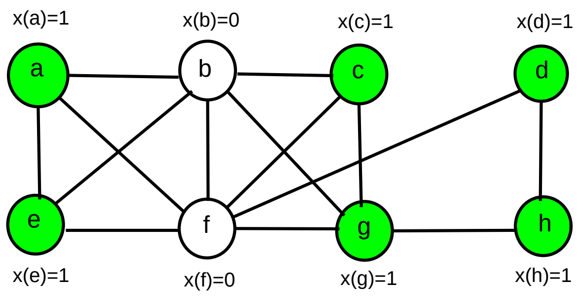

Figure 3.4: The M3DS result of Figs. 3.2 and 3.3 by running Algorithm 3.1.

The legitimate configurations are defined to be all those configurations in which

$$\forall i \in V[(x(i) = 0 \wedge |X(i)| \geq k) \vee (x(i) = 1 \wedge |X(i)| \leq k - 1)].$$

It is obvious that the system is in a legitimate configuration if and only if no node in the system is privileged. The following Lemma clarifies that in any legitimate configuration, an MKDS $D = \{i \in V | x(i) = 1\}$ can be identified.

Lemma 3.1. *If Algorithm 3.1 stabilizes (no node in the system is privileged), then the set* $D = \{i \in V | x(i) = 1\}$ *is an MKDS.*

Proof: Suppose no node in G is privileged. For any $i \notin D$, since $x(i) = 0$ and i is not privileged by R1, we have $|X(i)| \geq k$. Thus i has at least k neighbours in D and thus, D is a KDS in G. We now claim D is minimal as well. If D is not minimal, then there is a proper subset D' of D such that D' is also a KDS. Let $j \in D - D'$, then $x(j) = 1$. Since D' is also a KDS and $j \notin D'$, j has at least k neighbours in D' in view of the definition of KDS. It follows that j has at least k neighbours in D, i.e., $|X(j)| \geq k$. With $x(j) = 1$ and $|X(j)| \geq k$, j is privileged by R2, which causes a contradiction. Therefore D is an MKDS. ■

3.6 Convergence of the K-Dominating Set Algorithm 3.1

In this section, we show the convergence of Algorithm 3.1. We will provide some preliminaries and a correctness proof of convergence for Algorithm 3.1. We consider a distributed system whose topology is represented as a simple connected undirected graph $G = (V, E)$, where the nodes represent the processes and the edges represent the interconnections between the processes. We denote by δ the minimum degree of G ($\delta = min\{d(i)|i \in V\}$) and Δ the maximum degree of G ($\Delta = max\{d(i)|i \in V\}$) respectively.

In a distributed system, under a central daemon model, the behaviour of the system under the action of the algorithm can be described by an execution sequence $\Gamma = (\gamma_1, \gamma_2, ...)$ in which for any $i \geq 1$, γ_i represents a global configuration. For any configuration γ_i, let γ_{i+1} be any configuration that follows γ_i which is obtained after exactly one process in the system makes the i^{th} move. Then, we denote this transition relation by $\gamma_{i+1} \rightarrow \gamma_i$.

Definition 3.3. *Let a sequence* $\Gamma = (\gamma_1, \gamma_2, ...)$ *be a set of all configurations, and an algorithm E is self-stabilizing with respect to* $\Lambda \subseteq \Gamma$ *if and only if the following two conditions hold:*

a) Convergence: Starting from any arbitrary configuration, configuration eventually becomes one in Λ*, and*

b) Closure: For any configuration $\lambda \in \Lambda$*, any configuration* γ *that follows* λ *is also in* Λ.

Each $\lambda \in \Lambda$ is called a legitimate configuration. And Λ is called a set of legitimate configurations.

We now prove the convergence of Algorithm 3.1. Note that the term process represents the term node.

Theorem 3.1. *Algorithm 3.1 always stabilizes, and finds a minimal k-dominating set (MKDS).*

Proof: In light of Lemma 3.1 we see that if Algorithm 3.1 is stabilizing it always finds an MKDS, which satisfies the closure. We need to prove stabilization (convergence). We divide the distributed system (graph) into two cases: $\delta < k$ and $\delta \geq k$.

Case 1: $\delta < k$. Suppose that there exists an infinite computation E starting from γ_0 that does not reach a legitimate configuration. First, we consider the process $i \in V$ (P_i) with the degree δ. Because of $\delta < k$, a guarded command that i may execute is only R1. By the definitions of R1 and KDS, the P_i can execute R1 at most once in an infinite computation. Because computation E is infinite and the processes in G are finite, there must be at least one process , say P_j $(d(j) \geq k)$, which is executed infinitely. By the definition of Algorithm 3.1, a process alternates executing R1 and R2. After P_j executes a guarded command once, it is no longer privileged until $|X(j)|$ changes by an execution of at least one of the neighbours. Thus, there must be a neighbour, say P_k, of P_j which is also executed infinitely. By following such a dependency, we eventually reach the process i. This implies that the P_i must be executed infinitely. This is a contradiction.

Case 2: $\delta \geq k$. Suppose that there exists an infinite computation E starting from γ_0 that does not reach a legitimate configuration. First, we consider the process $i \in V$ (P_i) with the degree Δ. Because of $\Delta \geq k$, by the definitions of R1, R2 and KDS, a guarded command that the P_i may execute is R1 or R2. By the definition of Algorithm 3.1, after P_i executes a guarded command once, it is no longer privileged until $|X(i)|$ changes by an execution of at least one of the neighbours. Thus, the P_i execute R1 and R2 at most Δ times in an infinite computation. Since the processes in G are finite (The number of all processes is n). This is a contradiction.

Therefore, there is no infinite computation, and eventually computation terminates in a configuration in which no process is privileged. ■

Theorem 3.2. *Algorithm 3.1 produces an MKDS and stabilizes in $O(n^2)$ steps.*

Proof: From Lemma 3.1 and Theorem 3.1, we see that Algorithm 3.1 produces an MKDS. We need to prove only Algorithm 3.1 stabilizes in $O(n^2)$ steps. By Theorem 3.1, each node changes its x-value at most Δ times in a finite computation. Considering the graph G is a simple connected undirected graph, the upper bound of the Δ is $(n-1)$, the upper bound of moves is $n(n-1)$. Therefore, the stabilization time of Algorithm 3.1 is $O(n^2)$ steps. ■

3.7 Related Work and Algorithm Comparison

In this section, we discuss the existing self-stabilizing algorithms for the DS. We also compare our algorithm with the related work based on the main ideas. The algorithms presented in this section are summarized in Table 3.1.

Hedetniemi et al. [54] presented two uniform algorithms (all of the individual processes run the same code) for the DS and the MDS problems. The algorithms work for any connected graph and assume a central daemon (only one process can execute an atomic step at one time). The main idea of the first algorithm is to partition the set of nodes into two disjoint sets, such that each set is dominating. The algorithm for the DS problem stabilizes in linear time ($O(n)$ steps) under a central daemon. The second algorithm calculates an MDS. The main idea of this algorithm is that it allows a node to join the set S, if it has no neighbour in S. On the other hand, a node that is already a member of S, and has a neighbour that is also a member of S, will leave the set if all its neighbours are not pointing to it. Thus, after stabilization the set S will be an MDS. The algorithm for the MDS problem stabilizes in $O(n^2)$ steps under a central daemon.

Goddard et al. [58] gave a self-stabilizing algorithm working on the minimal total dominating set (MTDS) problem. A set is said to be a *total dominating set* if every node is adjacent to a member of it. Goddard et al. assume globally unique identifiers for the nodes and a central daemon in [58]. The algorithm uses a mechanism of pointers similar to the one used by the previous algorithm. So, a node i will point to its neighbour having the minimum identifier if i has no neighbour in the set S under construction. On the other hand, if a node i has more than one neighbour in the set then i will point to null; otherwise i will point to its unique neighbour that is a member of the set S. The algorithm allows a node to join the set S if some neighbour is pointing to it, and to leave the set S otherwise. So after stabilization, the set S will become an MTDS. The algorithm works under a central daemon for any general graphs.

Recently, Goddard et al. [88] proposed another uniform self-stabilizing algorithm for finding an MDS in an arbitrary graph under a distributed daemon (a distributed daemon selects a subset of the system processes to execute an atomic step at the same time). The main idea of the algorithm is that it uses a Boolean variable to determine whether a node is a member of the MDS or not, and an integer to count a node's neighbours that are members of the MDS. The algorithm allows an undominated node that has smaller identifier than any undominated neighbour to join the set under construction. On the other hand, a node leaves this latter set if it is not the unique dominator of itself nor any of its neighbours. The algorithm stabilizes in $O(n)$ steps.

Alternatively, some self-stabilizing algorithms have been proposed in the k-domination case. Kamei and Kakugawa [26] presented two uniform algorithms for the MKDS problem in a tree graph. The first algorithm allows a node to join the set S under construction if it has fewer than k neighbours in S, and to leave the set S if it has more than k neighbours in S. The first algorithm works for a central daemon. Based on this idea, in the second algorithm, a node having more than k neighbours in the set S under construction will first make a request to leave

S, and then leaves the set S only if its identifier is the smallest among all the neighbours requesting to leave S. So, after stabilization the set S will become an MKDS. The second algorithm works under a distributed daemon. The time complexity of the two algorithms are both $O(n^2)$ steps.

Huang et al. [55] presented a self-stabilizing algorithm to find an M2DS in an arbitrary graph. The algorithm allows a node to join the set under construction if it has fewer than 2 neighbours in S, and to leave the set S if it has more than 2 neighbours in S. The algorithm works under a central daemon with linear time complexity. Huang et al. also [68] presented another self-stabilizing algorithm to find an M2DS in an arbitrary graph. The algorithm assumes globally unique identifiers for the nodes and works under a distributed daemon. The algorithm allows a node to join the set under construction if it is dominated by fewer than two nodes and none of its neighbours having smaller identifier is in the same situation. Also, a node may leave the set under construction if it is dominated by more than two nodes, and all of its neighbours are either in the set under construction or dominated by more than two nodes.

Table 3.1: Self-stabilizing algorithms for the dominating set

Reference	**Output**	**Topology**	**Self-Stabilizing**	**Daemon**	**Complexity**
Hedetniemi et al. [54]-1	DS	Arbitrary	Yes	Central	$O(n)$ steps
Hedetniemi et al. [54]-2	MDS	Arbitrary	Yes	Central	$O(n^2)$ steps
Goddard et al. [58]	MTDS	Arbitrary	Yes	Central	
Goddard et al. [88]	MDS	Arbitrary	Yes	Distributed	$O(n)$ steps
Kamei et al. [26]-1	MKDS	Tree	Yes	Central	$O(n^2)$ steps
Kamei et al. [26]-2	MKDS	Tree	Yes	Distributed	$O(n^2)$ steps
Huang et al. [55]	M2DS	Arbitrary	Yes	Central	$O(n)$ steps
Huang et al. [68]	M2DS	Arbitrary	Yes	Distributed	
Wang et al. [71]	MKDS	Arbitrary	Yes	Central	$O(n^2)$ steps

We have presented a uniform self-stabilizing algorithm for finding an MKDS that works in general graphs under a central daemon [71]. We use a Boolean flag $x(i)$ indicating whether the node i is in the constructed set D or not and an integer variable $X(i)$ for counting $i's$ neighbours in D. The algorithm allows a node i to join the set D (the value $x(i) = 1$) under construction if it is dominated by fewer than k nodes in D (R1). Also, a node i may leave the set D under construction if it is dominated by more than k nodes (R2). The time complexity of our algorithm in general graphs is $O(n^2)$ steps.

The algorithms we compared in this section are summarized in Table 3.1. As we can see, the first four self-stabilizing algorithms are for single domination (1-dominating set or total dominating set); and the algorithms for the KDS problem by Kamei et al [26] work for a tree graph; the algorithms by Huang et al. [55, 68] are finding an M2DS. And the self-stabilizing algorithm for the MKDS problem in [71] works under a central daemon. Our Algorithm is the first work using a central daemon approach to discuss the MKDS problem in general networks.

3.8 Summary

In this chapter, we have successfully solved the problem, “To find a self-stabilizing algorithm for the MKDS problem in general networks, under the central daemon model”, extended from 2 to general k. Since the MKDS problem is more general than the M2DS problem and is much too complicated to handle. Kamei and Kakugawa have been the first to tackle the problem, restricted to tree networks only [26]. Huang et al. [55,68] discussed the case $k = 2$, the M2DS in any general network.

In the next chapter, we still study the extension problem further, “To find a self-stabilizing algorithm for the MKDS problem in general networks”. We will develop a self-stabilizing algorithm for the MKDS problem under a distributed daemon in an arbitrary network.

Chapter 4

A Self-Stabilizing Algorithm for K-Domination

In this chapter, we continue discussing the minimal k-dominating set (MKDS) problem we have studied in Chapter 3. In Chapter 3, we have presented self-stabilizing algorithm for the MKDS problem in arbitrary graphs under a central daemon [71]. But so far, there is no algorithm for the MKDS problem in arbitrary graphs that works under a distributed daemon. In this chapter, we propose a self-stabilizing algorithm for the MKDS under a distributed daemon model when operating in any general network. We further prove that the worst case convergence time of the proposed algorithm from any arbitrary initial state is $O(n^2)$ steps where n is the number of nodes in the network.

The information in this chapter is based on a published paper [72].

4.1 Introduction

4.1.1 K-Dominating Set Problem

The k-dominating set (KDS), which was introduced by Fink and Jacobson in 1985 [97], is that for a positive integer k, a *k-dominating set* (KDS) of a graph G is a subset $D \subseteq V(G)$ such that every vertex not in D is dominated by at least k vertices in D. The KDS has many real applications. For example, applications in wireless networks and social networks, a person needs to receive help from at least k times services from its neighbours. This problem is equivalent to the problem of finding a minimum KDS on the networks. The following is the formal definition of the KDS problem.

Let $G = (V, E)$ be a connected simple undirected graph in which each node $i \in V$ represents a person and each edge $(i, j) \in E$ represents the bidirectional link connecting nodes i and j. Let k be an arbitrary positive integer. A subset D of V is a *k-dominating set* in G if each node not in D has at least k neighbours in D. The so-called *"k-dominating set problem"* is to find a KDS of minimum size in G. The *k-dominating number* $\gamma(G)$ of G is the minimum size of a KDS of G. On the complexity side of the KDS problem, Jacobson and Peters showed that the KDS problem is NP-complete for general graphs [98] and gave linear-time algorithms to compute the $\gamma(G)$ of trees and series-parallel graphs [98].

A KDS D in G is *minimal* if any proper subset of D is not a KDS in G. Note that a minimal k-dominating set (MKDS) is minimal not minimum. A k-dominating set for $k = 1$, i.e., a 1-dominating set, is just an ordinary dominating set.

The topology of a distributed system can be represented as an undirected graph $G = (V, E)$ (called the system's communication graph), where the nodes represent the processes and the edges represent the interconnections between the processes. In this chapter, we denote by n the number of nodes ($|V| = n$), and by m the number of edges ($|E| = m$) in the graph G. Let $i \in V$ be a node; then $N(i)$, its *open neighbourhood*, denotes the set of nodes to which i is adjacent. Every node $j \in N(i)$ is called a neighbour of node i. We denote by $d(i)$ the number of neighbours of node i, or its degree ($d(i) = |N(i)|$).

4.1.2 Self-Stabilizing Algorithm

The concept of Self-Stabilization, introduced by Dijkstra [56, 57] is considered to be a very general technique to design a system to tolerate arbitrary transient faults. A distributed system is *self-stabilizing* if it can start at any possible global configuration and regain consistency in a finite number of steps by itself without any external intervention and remains in a consistent state [67]. By this property, self-stabilizing algorithms tolerate any kind and any finite number of transient faults.

The state of a fault-free system is defined by a local predicate based on the states of the nodes in the distributed algorithm. Each node is based on local knowledge: its own state and the states of its immediate neighbours. The objective is to achieve some global objective a predicate defined on the states of all the nodes in the network – based on local actions where individual nodes have no global knowledge about the network. Refer to Chapter 3 for a general overview of the paradigm of self-stabilization and its requirements.

4.2 Related Work

There are many self-stabilizing algorithms for finding a dominating set of a network, including the single dominating set and the multiple dominating set ($k > 1$).

Hedetniemi et al. [54] presented two uniform algorithms (all of the individual processes run the same code) for the dominating set (DS) and the minimal dominating set (MDS) problems. The algorithms work for any connected graph and assume a central daemon (only one process can execute an atomic step at one time). The main idea of the first algorithm is to partition the set of nodes into two disjoint sets, such that each set is dominating. The algorithm for the DS problem stabilizes in linear time ($O(n)$ steps) under a central daemon. The second algorithm calculates an MDS. The main idea of this algorithm is that it allows a node to join the set S, if it has no neighbour in S. On the other hand, a node that is already a member of S, and has a neighbour that is also a member of S, will leave the set if all its neighbours are not pointing to it. Thus, after stabilization the set S will be an MDS. The algorithm for the MDS problem stabilizes in $O(n^2)$ steps under a central daemon.

Goddard et al. [58] gave a self-stabilizing algorithm working on the minimal total dominating set (MTDS) problem. A set is said to be a *total dominating set* if every node is adjacent to a member of it. The authors assume globally unique identifiers for the nodes and a central daemon in [58]. The algorithm uses a mechanism of pointers similar to the one used by the previous algorithm. So, a node i will point to its neighbour having the minimum identifier if i has no neighbour in the set S under construction. On the other hand, if a node i has more than one neighbour in the set then i will point to null; otherwise i will point to its unique neighbour that is a member of the set S. The algorithm allows a node to join the set S if some neighbour is pointing to it, and to leave the set S otherwise. So after stabilization, the set S will become an MTDS.

Recently, Goddard et al. [88] proposed another uniform self-stabilizing algorithm for finding an MDS in an arbitrary graph under a distributed daemon (a distributed daemon selects a subset of the system processes to execute an atomic step at the same time). The main idea of their algorithm is that it uses a Boolean variable to determine whether a node is a member of the MDS or not, and an integer to count a node's neighbours that are members of the MDS. The algorithm allows an undominated node that has a smaller identifier than any undominated neighbour to join the set under construction. On the other hand, a node leaves this latter set if it is not the unique dominator of itself nor any of its neighbours. The algorithm stabilizes in $O(n)$ steps.

Meanwhile, some self-stabilizing algorithms have been proposed in the k-domination case. Kamei and Kakugawa [26] presented two uniform algorithms for the minimal k-dominating set (MKDS) problem in a tree graph. The first algorithm allows a node to join the set under construction S if it has fewer than k neighbours in S, and to leave the set S if it has more than k neighbours in S. The first algorithm works for a central daemon. Based on this idea, in the second algorithm, a node having more than k neighbours in the set S under construction will first make a request to leave S, and then leaves the set S only if its identifier is the smallest among all the neighbours requesting to leave S. So, after stabilization the set S will become an MKDS. The second algorithm works under a distributed daemon. The time complexity of the two algorithms are both $O(n^2)$ steps.

Huang et al. [55] presented a self-stabilizing algorithm to find a minimal 2-dominating set (M2DS) in an arbitrary graph. The algorithm allows a node to join the set S under construction if it has fewer than 2 neighbours in S, and to leave the set S if it has more than 2 neighbours in S. The algorithm works under a central daemon, with linear time complexity.

Huang et al. also [68] presented another self-stabilizing algorithm to find an M2DS in an arbitrary graph. The algorithm assumes globally unique identifiers for the nodes and works under a distributed daemon. The algorithm allows a node to join the set under construction if it is dominated by fewer than two nodes and none of its neighbours having a smaller identifier is in the same situation. Also, a node may leave the set under construction if it is dominated by more than two nodes, and all of its neighbours are either in the set under construction or dominated by more than two nodes.

In Chapter 3, we have presented a self-stabilizing algorithm for finding an MKDS that works in general graphs under a central daemon [71]. We used a Boolean flag $x(i)$ indicating whether the node i is in the constructed set D or not and an integer variable $X(i)$ for counting $i's$ neighbours in D. The algorithm allows a node i to join the set D (the value $x(i) = 1$) under construction if it is dominated by fewer than k nodes in D (R1). Also, a node i may leave the set D under construction if it is dominated by more than k nodes (R2). The time complexity of our algorithm in general graphs is $O(n^2)$ steps.

4.3 Motivations and contributions

The self-stabilizing algorithms we dicussed in Section Related Work all work on the dominating set problems. However, there is no self-stabilizing algorithm for the MKDS problem in arbitrary graphs that works under a distributed daemon. The proposed algorithms for the MKDS either work for tree graphs (Kamei and Kakugawa [26]) or find an M2DS (Huang et al. [55, 68]). We have solved the MKDS problem under a central daemon which operates in general graphs in Chapter 3.

In this chapter, we have motivations to extend Huang et al.'s work and consider the extension problem of the M2DS [55, 68] from 2 to general k: the MKDS. We are interested in the work about developing a self-stabilizing algorithm for finding an MKDS in a general network that works under a distributed daemon. We also analyze the correctness and time complexity of the proposed algorithm, in which the time complexity of the algorithm has not been discussed in [68]. The contributions are as follows:

1. A self-stabilizing algorithm for finding an MKDS under a distributed daemon in an arbitrary connected simple undirected graph is presented.

2. The correctness of the proposed algorithm is verified. The worst case convergence time of the algorithm from any arbitrary initial state is proved to be $O(n^2)$ steps where n is the number of nodes in the network graph.

4.4 K-Dominating Set Algorithm

In this section, the self-stabilizing algorithm for solving the MKDS problem will be presented.

4.4.1 Formal Definition of the MKDS Problem

The distributed system under consideration has a general underlying topology, and can be modeled by a connected simple undirected graph $G = (V, E)$, with each node $i \in V$ representing a processor in the system and each edge $(i, j) \in E$ representing the bidirectional link connecting processors i and j. It is assumed that the number of all processors in G is denoted by n. Assume now that for each processor $i \in V$, the set $N(i)$ represents its *open neighbourhood*, denotes the set of processors to which i is adjacent. $d(i)$ represents the number of neighbours of processor i, or its degree ($d(i) = |N(i)|$). Throughout this chapter, we denote by δ the minimum degree of G ($\delta = min\{d(i)|i \in V\}$) and Δ the maximum degree of G ($\Delta = max\{d(i)|i \in V\}$) respectively. It is assumed that

(1) each processor i in the system has a unique identity,

(2) each processor i maintains two shared registers, d_i and p_i,

(3) $N(i)$ denotes the set of all open neighbours of i and $L(i) = \{j \in N(i)|j < i\}$,

(4) the value of d_i is taken from $\{0, 1\}$,

(5) $D(i) = \{j \in N(i)|d_j = 1\}$, and $|D(i)|$ is the cardinality of $D(i)$, and

(6) the value of p_i is always $\emptyset$, $\{i\}$ or $D(i)$.

A Boolean variable d_i indicates membership in the set D that we are trying to construct. The value $d_i = 1$ indicates that $i \in D$, while the value $d_i = 0$ indicates that $i \notin D$.

The concept of the MKDS problem gives rise to the following problem:

Minimal K-Dominating Set

INSTANCE: A connected simple undirected graph $G = (V, E)$ with nodes set V and edges set E, given an arbitrary positive integer k.

QUESTION: How to find an MKDS $D \subseteq V$ such that the D is a KDS of the graph G and minimal, i.e., to construct $D = \{i \in V|d_i = 1\}$ is an MKDS.

4.4.2 Proposed Algorithm

The Algorithm 4.1 is shown below which consists of five rules. Assume k being an arbitrary positive integer. It should also be reiterated that the computational model assumed in the system is the distributed daemon model.

$R1$ tries to ensure that a node $i \notin D$ which has less than k neighbours in D should enter D, if its pointer points to itself and its neighbours which have smaller IDs than it has are all dominated at least k times. $R2$ says that a node i is dominated

Algorithm 4.1: Finding A Minimal K-Dominating Set

```
**Input:** A graph $G = (V, E)$, $\forall i \in V$, a Boolean flag $d_i$ and a set of pointers $p_i$.
**Output:** $D = \{i \in V | d_i = 1\}$.
**R1:** $d_i = 0 \wedge |D(i)| < k \wedge p_i = \{i\} \wedge \forall j \in \{m \in L(i) | d_m = 0\}$,
        $p_j \neq \{j\} \rightarrow d_i := 1$
**R2:** $d_i = 1 \wedge |D(i)| \geq k \wedge \forall j \in N(i) - D(i),\ p_j = \emptyset \rightarrow d_i := 0$
**R3:** $d_i = 0 \wedge |D(i)| < k \wedge p_i \neq \{i\} \rightarrow p_i = \{i\}$
**R4:** $d_i = 0 \wedge |D(i)| = k \wedge p_i \neq D(i) \rightarrow p_i = D(i)$
**R5:** $d_i = 0 \wedge |D(i)| > k \wedge p_i \neq \emptyset \rightarrow p_i = \emptyset$.
```

at least k times by D should leave D if as far as it can tell all of its neighbours not in D are dominated at least k times. That is, the node i is redundant for its neighbours not in D since they are dominated at least k times by $D - \{i\}$. $R3$, $R4$ and $R5$ mean that the nodes not in D should reset their pointers according to the numbers of their neighbours in D. An example to illustrate the execution of Algorithm 4.1 is shown in Fig. 4.1.

Example 4.1. *The example in Fig. 4.1 is to illustrate the execution of Algorithm 4.1. Without loss of generality, we consider the case* $k = 2$ *and use a minimal 2-dominating set example to illustrate the execution of Algorithm 4.1. Note that in each configuration, the shaded nodes represent privileged nodes.*

In the first subgraph of Fig. 4.1, we set $d_1 = d_3 = 1$, *other nodes' d-values are 0, that means* $D = \{1,\ 3\}$. *We further set* $p_1 = p_5 = \{1\}$, $p_3 = p_6 = \{3\}$, $p_2 = \{2\}$, $p_4 = \{1,\ 3\}$, *just as the arrows point in the first subgraph. According to the* Rules *of Algorithm 4.1, after a series of moves and resets, the system reaches a legitimate state. As the last subgraph of Fig. 4.1 shows, which is a legitimate configuration, we can see a minimal 2-dominating set* $D = \{1,\ 2,\ 3,\ 6\}$ *can be identified.*

Note that the 2-dominating set $D = \{1,\ 2,\ 3,\ 6\}$ is *minimal* not *minimum.* The set $D = \{3,\ 4,\ 5\}$ or $D = \{4,\ 5,\ 6\}$ is an M2DS.

4.5 The Stabilization Time of Algorithm 4.1

The legitimate configurations are defined to be all those configurations in which no node in the system is privileged. The following theorem clarifies that in any legitimate configuration, a minimal k-dominating set can be identified (*closure*).

Theorem 4.1. *If Algorithm 4.1 stabilizes,, then the set* $D = \{i \in V | d_i = 1\}$ *is an MKDS.*

Proof. It is obvious that the system is in a legitimate configuration if and only if no node in the system is privileged.

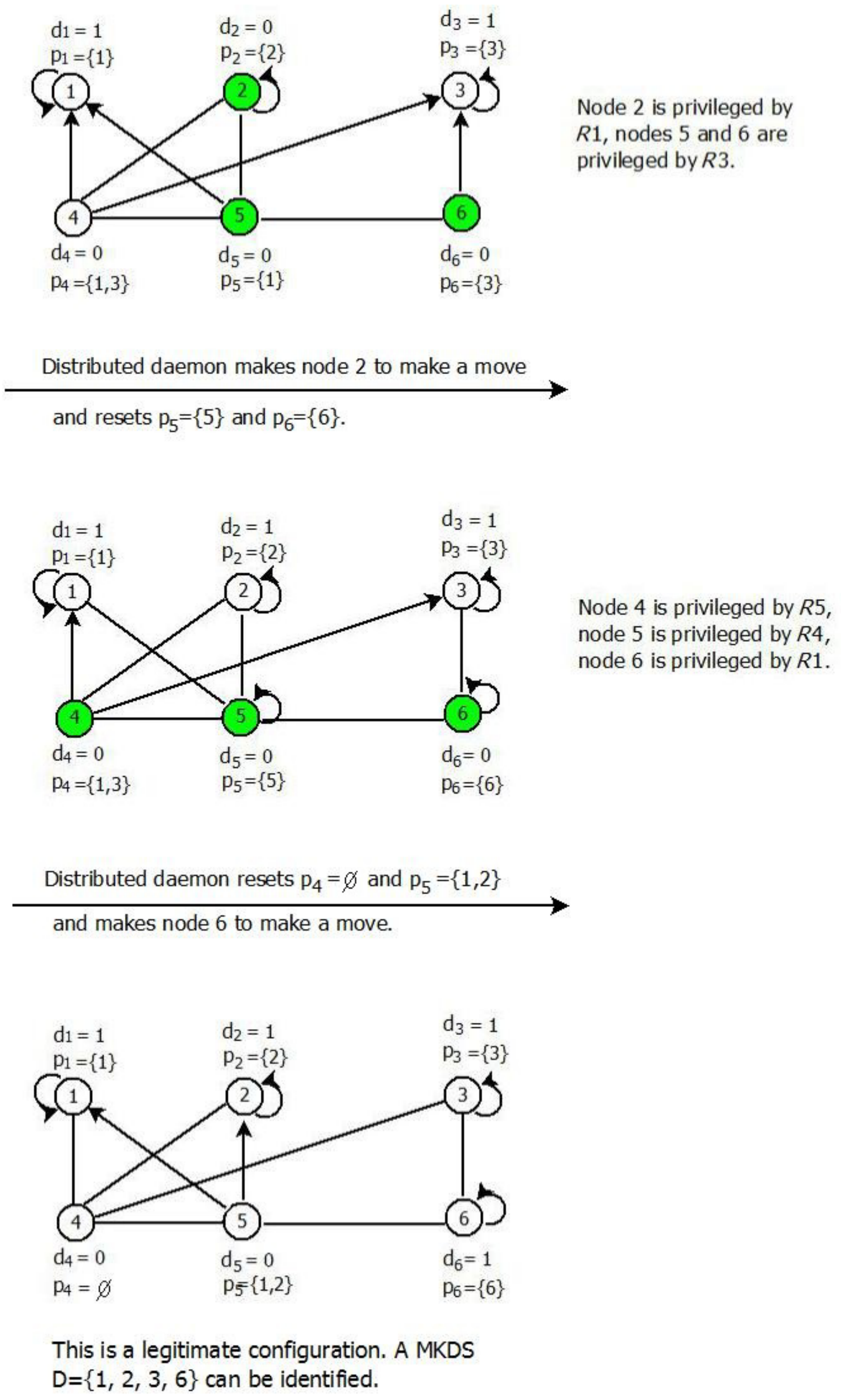
d1 = 1
p1 ={1}
d2 = 0
p2 ={2}
d3 = 1
p3 ={3}
d4 = 0
p4 ={1,3}
d5 = 0
p5={1}
d6= 0
p6={3}
Node 2 is privileged by R1, nodes 5 and 6 are privileged by R3.
Distributed daemon makes node 2 to make a move and resets p5={5} and p6={6}.
d1 = 1
p1 ={1}
d2 = 1
p2 ={2}
d3 = 1
p3 ={3}
d4 = 0
p4 ={1,3}
d5 = 0
p5={5}
d6= 0
p6={6}
Node 4 is privileged by R5, node 5 is privileged by R4, node 6 is privileged by R1.
Distributed daemon resets p4 =∅ and p5 ={1,2} and makes node 6 to make a move.
d1 = 1
p1 ={1}
d2 = 1
p2 ={2}
d3 = 1
p3 ={3}
d4 = 0
p4 = ∅
d5 = 0
p5={1,2}
d6= 1
p6={6}
This is a legitimate configuration. A MKDS D={1, 2, 3, 6} can be identified.

Figure 4.1: A M2DS example to illustrate the execution of Algorithm 4.1.

(1) Suppose D is not a KDS, then there exists a node $i \in V - D$ such that i has at most $k-1$ neighbours in D, i.e., $d_i = 0$ and $|D(i)| < k$.

Claim. For any node i in the system when it reaches a legitimate configuration, if $d_i = 0$ and $|D(i)| < k$, then there exists a node $j \in L(i)$ (thus $j < i$) such that $d_j = 0$ and $|D(j)| < k$.

Proof of Claim. Since $d_i = 0$, $|D(i)| < k$, and i is not privileged by $R3$, we have $p_i = \{i\}$. Then since i is not privileged by $R1$, i.e., $\forall j \in \{m \in L(i) | d_m = 0\}, p_j \neq \{j\}$ cannot hold. Hence there exists a node $j_0 \in L(i)$ such that $d_{j_0} = 0$ and $p_{j_0} = \{j_0\}$. If $|D(j_0)| = k$, then since $d_{j_0} = 0$ and $p_{j_0} = \{j_0\} \neq D(j_0)$, j_0 is privileged by $R4$, which causes a contradiction. If $|D(j_0))| > k$, then since $d_{j_0} = 0$ and $p_{j_0} = \{j_0\} \neq \emptyset$, j_0 is privileged by $R5$, which causes a contradiction. Hence $|D(j_0))| < k$ and the claim is proved. □

By applying the above claim to node i, we get a node $i_1 \in L(i)$ such that $d_{i_1} = 0$ and $|D(i_1)| < k$. Then, by applying the claim to node i_1, we get a node $i_2 \in L(i_1)$ such that $d_{i_2} = 0$ and $|D(i_2)| < k$. In this way, we eventually get infinitely many nodes $i_1, i_2, i_3, \ldots$ such that $i > i_1 > i_2 > i_3 > \cdots$. However, this causes a contradiction because the system has only a finite number of nodes. Therefore, D must be a KDS.

(2) Suppose D is not an MKDS, then there exists a node $i \in D$ such that $D - \{i\}$ is a KDS. Since $i \notin D - \{i\}$ and $D - \{i\}$ is a KDS, i has at least k neighbours in $D - \{i\}$ and thus $|D(i)| \geq k$. If $N(i) - D(i) = \emptyset$, then, since $d_i = 1$ and $|D(i)| \geq k$, i is privileged by $R2$, which causes a contradiction. Hence $N(i) - D(i) \neq \emptyset$. Let j be an arbitrary node in $N(i) - D(i)$. Since $j \notin D - \{i\}$ ($d_j = 0$) and $D - \{i\}$ is a KDS, j has at least k neighbours in $D - \{i\}$. Thus j has at least $k+1$ neighbours in D, i.e., $|D(j)| > k$. Since $d_j = 0$, $|D(j)| > k$ and j cannot be privileged by $R5$, we have $p_j = \emptyset$. Hence the condition [$\forall j \in N(i) - D(i), p_j = \emptyset$] holds. Since $d_i = 1$, $|D(i)| \geq k$ and [$\forall j \in N(i) - D(i), p_j = \emptyset$] hold, node i is privileged by $R2$, which causes a contradiction. Hence D is an MKDS. ■

In the following, we prove the convergence of Algorithm 4.1. We will first provide two Lemmas. Based on these two Lemmas, we will prove the convergence time of Algorithm 4.1.

Lemma 4.1. *If d_i changes from 0 to 1, then d_i will not change again.*

Proof: Since d_i changes from 0 to 1 by $R1$ at time t, so the condition $d_i = 0 \wedge |D(i)| < k \wedge p_i = \{i\} \wedge \forall j \in \{m \in L(i) | d_m = 0\},\ p_j \neq \{j\}$ holds. By $R1$, only the node with larger ID than $i's$ is able to enter the set D. Suppose at the time t', the node i leaves the set D by $R2$, thus the condition $d_i = 1 \wedge |D(i)| \geq k \wedge \forall j \in N(i) - D(i),\ p_j = \emptyset$ holds. So there exists a node $j \in N(i) - D(i)$ with lager ID than $i's$ entering the set D. Then the node j satisfies $R1$, we can get $p_j = \{j\}$, but if the node i leaves the set D it satisfies the *R2*, and we have $p_j = \emptyset$, which causes a contradiction. ■

Lemma 4.2. *A node can make at most two membership moves.*

Proof: If a nodes first membership move is by $R1$, by Lemma 4.1, it will not make a membership move again. If its first membership move is R2, then any next membership move must be by $R1$, after which, it cannot make another membership move. ■

Now we will prove Algorithm 4.1 always stabilizes (*convergence*).

Theorem 4.2. *Algorithm 4.1 produces an MKDS and stabilizes in $O(n^2)$ steps.*

Proof: In light of Theorem 4.1 we need only prove stabilization (*convergence*). By Lemma 4.2, each node will change its d-value at most twice. Therefore, there can be at most $2n$ changes of d-values on all nodes during the whole time. If there is no change in d-value of any node in a time-step, then the time-step involves only changes in p-values. The change in a p-value is determined only by d-values and its neighbours in the set D according to the $R3$, $R4$ or $R5$. Consider the processor $i \in V$ with the largest degree Δ, a guarded command the p_i may execute is $R3$, $R4$ or $R5$. By definition of the Algorithm 4.1, after p_i executes a guarded command once, it is no longer privileged until $|D(i)|$ changes by an execution of at least one of the neighbours. Thus, the p_i executes $R3$, $R4$ or $R5$ at most 2Δ times in an infinite computation. So, the upper bound of the execution time is $2(\Delta + 1)n$ time-steps. Considering the graph G is a connected simple undirected graph, the upper bound of the Δ is $(n - 1)$, therefore, Algorithm 4.1 produces an MKDS and the stabilization time of Algorithm 4.1 is $O(n^2)$ steps. ■

4.6 Algorithm Comparison

In this section, we collect and discuss the existing self-stabilizing algorithms for the dominating set (DS) respectively. The self-stabilizing algorithms presented in this chapter are summarized in Table 4.1. The basic ideas of the first nine self-stabilizing algorithms for the DS are discussed in Related Work (refer to Section 4.3). We also introduce the basic idea of the proposed algorithm presented in this chapter.

Table 4.1: Algorithms for the dominating set

Reference	**Output**	**Topology**	**Self-stabilizing**	**Daemon**	**Complexity**
Hedetniemi et al. [54]-1	DS	Arbitrary	Yes	Central	$O(n)$ steps
Hedetniemi et al. [54]-2	MDS	Arbitrary	Yes	Central	$O(n^2)$ steps
Goddard et al. [58]	MTDS	Arbitrary	Yes	Central	
Goddard et al. [88]	MDS	Arbitrary	Yes	Distributed	$O(n)$ steps
Kamei et al. [26]-1	MKDS	Tree	Yes	Central	$O(n^2)$ steps
Kamei et al. [26]-2	MKDS	Tree	Yes	Distributed	$O(n^2)$ steps
Huang et al. [55]	M2DS	Arbitrary	Yes	Central	$O(n)$ steps
Huang et al. [68]	M2DS	Arbitrary	Yes	Distributed	
Wang et al. [71]	MKDS	Arbitrary	Yes	Central	$O(n^2)$ steps
Algorithm 4.1 [72]	MKDS	Arbitrary	Yes	Distributed	$O(n^2)$ steps

The main idea of our algorithm is that it allows a node i to join the set D (the value $d_i = 1$) under construction if its pointer points to itself and its neighbours which have smaller IDs than it has are all dominated at least k times ($R1$). Also, a node i may leave the set under construction the set D if as far as it can tell all of its neighbours not in D are dominated at least k times by $D - \{i\}$. The time complexity of our algorithm in an arbitrary system graph is $O(n^2)$ steps.

In this chapter we have designed a self- stabilizing algorithm for the MKDS (Algorithm 4.1) under a distributed daemon. The *distributed daemon* activates the processors by repeatedly selecting a set of processors and activating them simultaneously to execute a computation step. Each processor executes the next computation step as defined by its state just prior to this activation. Once every processor in the set has finished reading, all the processors write a new state (change state). Only then does the scheduler choose a new set of processors to be activated. Note that no non-activated processor changes its state. The *central daemon* is a special case of the distributed daemon in which the set of activated processors consists of exactly one processor. Thus if a system is self-stabilizing under the distributed daemon model, then it is self-stabilizing under the central daemon model. The converse, however, is not true (the total dominating set algorithm in [58] and the 2-dominating set algorithm in [55] are self-stabilizing under the central daemon models, but not under the distributed daemon models).

The algorithms we compared in this section are summarized in Table 4.1. As we can see, the self-stabilizing algorithms for MDS of Hedetniemi et al. [54] and Goddard et al. [58,88] all work for the single domination (the 1-dominating set or the total dominating set). On the other hand, the other last six algorithms have been proposed in the multiple domination case [26,55,68,71,72]. The algorithms for the KDS problem by Kamei et al [26] work for a tree graph; the algorithms by Huang et al. [55, 68] find an M2DS in general graphs. The self-stabilizing algorithm for the MKDS in [71] works under a central daemon in general graphs. Our Algorithm 4.1 is the first work using a distributed daemon approach to discuss the MKDS problem in general graphs.

4.7 Summary

In this chapter, we have successfully solved the extension problem, "To find a self-stabilizing algorithm for the MKDS problem in general networks, under a distributed daemon model", for the general k. Since the MKDS problem is more general than the M2DS t problem, it is much too complicated to handle. Kamei and Kakugawa have been the first to tackle the above extension problem, although their results in [26] are restricted to tree networks only. Huang et al. [55, 68] discussed the case $k = 2$, M2DS in any general network.

An immediate extension of this work is to find if it is possible to enhance the stabilization time to $O(nlgn)$ steps. Another future research topic is to attempt to find a proper upper bound size of an MKDS in an arbitrary network.

Chapter 5

A Self-Stabilizing Algorithm for Weighted Domination

In this chapter, we focus on a classical domination problem - the weighted domination problem, which has been used in many practical and theoretical application fields. The existing self-stabilizing algorithms for the dominating set (DS) all work for uniform weight cases. There are seldom self-stabilizing algorithms working for the weighted dominating set (WDS). So we present a self-stabilizing algorithm for finding a minimal weighted dominating set (MWDS) under a central daemon in a general graph. We further prove that the worst case convergence time of the proposed algorithm from any arbitrary initial state is $O(n^2)$ steps where n is the number of nodes in the graph.

The information in this chapter is based on one published paper [75].

5.1 Introduction

Graph domination problems are important combinatorial problems with many practical and theoretical applications. They model many relevant problems in fields such as communication networks and network design, social network theory, computational complexity, and algorithm design. Many facility location, resource allocation, and scheduling problems, are variants of graph domination problems. For example, an application of fire stations, suppose a city has decided to build some fire stations, which must serve all of the people in the city. Assume in Toowoomba city, the fire stations are to be located in areas such as Kearneys Spring, Middle Ridge, Harristown and Newtown so that every area is a neighbour of an area which has a fire station to help it. Since each area has a request target, if too few fire stations are set up to rescue people if there is a fire, there is a high likelihood that in some areas people cannot get rescued in time due to the distance involved. On the other hand, due to a fixed cost of building a fire station there are financial limitations in budget. Hence, it is too expensive to build a fire station in each area. Therefore, how to choose a subset of all the areas to build the fire stations so that each area of the city can be rescued in time becomes a significant issue of the city plan.

From the above example, we can model these four areas into a simple network with 4 nodes. The fire station location problem can be formulated as a weighted domination problem in a graph. Formally, a subset $D \subseteq V$ of a graph $G = (V, E)$ is a *dominating set* (DS) if every vertex in $V - D$ is adjacent to some vertex in D. Let $G = (V, E, W)$ be a simple undirected weighted graph with a weight function $W = \{w_1, w_2, ..., w_n\}$. w_i is the weight associated with vertex i for $i \in \{1, 2, ..., n\}$, which can represents the fire-fighting capacity of each node i (also can be regarded as the construction fees). Each node also has a weight target $t(i)$ (help request). A *weighted dominating set* (WDS) D in G is a subset of V such that for all $i \in V$, the sum $\sum_{j \in N(i) \cap D} w(j) \geq t(i)$, where the set $N(i)$ represents its *open neighbourhood*, denotes the set of nodes to which i is adjacent. Our purpose is to find a minimal assignment of the total values that satisfy the constraints.

In this chapter, we will design a self-stabilizing algorithm for the WDS problem which works under a central daemon. Next, we discuss some basic ideas of Dijkstra's central daemon model.

5.2 Dijkstra's Central Daemon Model

Dijkstra's central daemon model of computation [56, 57, 89] of an algorithm in a distributed system has the following features:

The algorithm running on each processor consists of one or more rules. Each rule is of the form

condition part $\rightarrow$ *action part.*

The *condition part* (or *guard*) is a Boolean function over the states of the processor and its neighbours; the action part is an assignment of values to some of the processors shared registers. If the condition part of a rule in a processor is evaluated as true, we say that the processor is *privileged* to execute the action part (or to *make a move*).

In the initial configuration, if none of the processors is privileged, then the system is deadlocked. Otherwise, if a privileged processor exists, the *central daemon* in the system will randomly select exactly one among all the privileged processors to make a move, in a single atomic step. The local state of the selected processor thus changes and in the meantime results in the change of the global configuration of the system. The system will then repeat the above process to change global configurations as long as it does not encounter any deadlock situation. Thus, the behaviour of the system under the action of the algorithm can be described by *executions* (or a move). It is required that the executions are finite and no node is privileged in the last configuration.

Under this computational model, a great number of papers regarding self-stabilizing algorithms have been published. For example, self-stabilizing algorithms for dominating sets, independent sets, colorings, and matchings in graphs have been developed [54, 61–66, 88].

5.3 Motivations and Contributions

Although domination problems are all NP-hard in general graphs [11, 37], which means to find a minimum dominating set is NP-hard in general graphs, some proposed self-stabilizing algorithms for the minimal dominating set are in polynomial time (steps or rounds) complexity [26, 54] or linear time (steps or rounds) complexity [54, 55]. However, the proposed self-stabilizing algorithms for the minimal k-dominating set (MKDS) (Kamei and Kakugawa [26]) and the minimal generalized dominating set (Goddard et al. [58]) all work for uniform weight cases. The weight of each node in most of the proposed algorithms was uniform [26, 26, 54, 55, 58]. On the other hand, the polynomial time approximation algorithms for the weighted domination problems are all greedy algorithms for finding approximation solutions [16–18]. There are seldom self-stabilizing algorithms for the WDS.

In this chapter, we are interested in a minimal weighted dominating set (MWDS) raised from some facility location problems. We first propose a self-stabilizing algorithm for the MWDS under a central daemon model when operating in any general network, which is a theoretical contribution to Distributed Computing.

The following are our contributions.

1. A self-stabilizing protocol for finding an MWDS in an arbitrary network graph under a central daemon is presented.
2. The worst case convergence time of the protocol from any arbitrary initial state is proved to be $O(n^2)$ steps where n is the number of nodes in the network.

To the best of our knowledge, this is the first work using a self-stabilizing algorithm to find an MWDS.

5.4 Weighted Dominating Set

Let $G = (V, E, W)$ be a connected simple undirected weighted graph that models a distributed system, with each node $i \in V$ representing a process in the system, each edge $(i, j) \in E$ representing the bidirectional link connecting processes i and j, and a nonnegative weight $W = \{w_1, w_2, ..., w_n\}$ (the weight function) denotes the set of node weights. w_i is the weight associated with node i for $i \in \{1, 2, ..., n\}$. We denote $w(S) = \sum_{i \in S} w(i)$ if S is a subset of V. We denote by n the number of nodes ($|V| = n$), and by m the number of edges ($|E| = m$) in the graph G. For each node $i \in V$, the set $N(i)$ represents its *open neighbourhood*, denotes the set of nodes to which i is adjacent. $d(i)$ represents the number of neighbours of node i, or its degree ($d(i) = |N(i)|$). Throughout this chapter, we denote by Δ the maximum degree of G ($\Delta = max\{deg(i)|i \in V\}$). Our algorithm requires that every node has a unique ID. Sometimes i interchangeably denotes a node or the node's ID. Assume there is a total ordering on the IDs. Assume now that each node has a weight value target $t(i)$ such that $t(i) \leq w(N(i))$, indicates that there must be a WDS D such that any node $i \in V$ can be dominated by D. Given these assumptions the definition of a WDS $D \subseteq V$ is as follows:

Definition 5.1. *A subset $D \subseteq V$ is said to be a* ***weighted dominating set*** *(WDS) if for every node $i \in V$,*

$$\sum_{j \in N(i) \cap D} w(j) \geq t(i). \tag{5.1}$$

A WDS D is a *minimal* weight dominating set (MWDS) if any proper subset of D is not a WDS in G.

The concept of the MWDS problem gives rise to the following problem:

Minimal Weighted Dominating Set

INSTANCE: A connected simple undirected weighted graph $G = (V, E, W)$ and a weight value target $t(i)$ such that $t(i) \leq w(N(i))$.

QUESTION: How to find an MWDS $D \subseteq V$ such that the D is a WDS of the graph G and minimal.

In this chapter, we propose a self-stabilizing algorithm under a central daemon for finding an MWDS of a general graph. In our algorithm, each node i has two variables: a set of pointers $P(i)$ and a Boolean flag $x(i)$. If $P(i) = \{j\}$, then we say that i points to j, written $i \to j$. We allow $P(i)$ to contain i and its cardinality is bounded by $N(i)$. Each node also has a Boolean flag $x(i)$. At any given time, we will denote with D the current set of nodes i with $x(i) = 1$, otherwise $x(i) = 0$. That is,

$$x(i) = \begin{cases} 0 & \text{if } i \notin D \\ 1 & \text{if} \in D. \end{cases} \tag{5.2}$$

At a given time, assume $\sum_{j \in N(i) \cap D} w(j) < t(i)$. Then since $t(i) \leq w(N(i))$, there is at least one member in $N(i) - D$.

Definition 5.2. *Let M_i denote the unique set in $N(i) - D$ having the smallest IDs such that:*

$$M_i = Min\{S | w((D \cap N(i)) \cup S) \geq t(i), S \subseteq (N(i) - D)\}.$$

Note that this set depends on $N(i)$ and D.

Definition 5.3. *A set of pointers $Q(i)$ is designed as follows:*

$$Q(i) = \begin{cases} (D \cap N(i)) \cup M_i & \textit{if } w(N(i) \cap D) \leq t(i) \\ \emptyset & \textit{if } w(N(i) \cap D) > t(i). \end{cases} \tag{5.3}$$

Note that the value $Q(i)$ can be computed by i (i.e., it uses only local information).

Definition 5.4. *The Boolean condition $y(i)$ is defined to be 1 if and only if a neighbour of i points to it, otherwise its value is* 0*.*

5.5 Weighted Dominating Set Algorithm

Algorithm 5.1: Finding a Minimal Weighted Dominating Set

```
Input: A graph $G = (V; E; W)$, $\forall i \in V$, a Boolean
flag $x(i)$, a target $t(i)$ and a set of pointers $P(i)$.
 Output: $D = \{i \in V | x(i) = 1\}$.
 R1: if  $x(i) \neq y(i) \vee P(i) \neq Q(i)$
       then  $x(i) = y(i) \wedge P(i) = Q(i)$
```

The Algorithm 5.1 (protocol) which consists of one rule (R1) shows the pseudo-code above. In Algorithm 5.1 each node i has a Boolean variable $x(i)$ indicating membership in the set D that we are trying to construct. The value $x(i) = 1$ indicates that $i \in D$, while the value $x(i) = 0$ indicates that $i \notin D$. The Boolean condition $y(i)$ is defined to be 1 if and only if a neighbour of i points to it. $P(i)$ is a set of pointers. $Q(i)$ is counted by equation (5.3). Thus, a node i is *privileged* if $x(i) \neq y(i)$ or $P(i) \neq Q(i)$. If R1 executes, then it sets $x(i) = y(i)$ and $P(i) = Q(i)$. An example to illustrate the execution of Algorithm 5.1 is shown in Fig. 5.1.

It is obvious that the system is in a legitimate configuration if and only if no node in the system is privileged. The following lemma clarifies that in any legitimate configuration, an MWDS $D = \{i \in V | x(i) = 1\}$ can be identified.

Lemma 5.1. *If Algorithm 5.1 stabilizes then $D = \{i \in V | x(i) = 1\}$ is an MWDS satisfying the definition of WDS (Definition 5.1).*

Proof: Suppose that D satisfies the definition of WDS (Definition 5.1). By contradiction suppose that for some i, $w(N(i) \cap D) < t(i)$. Then $M_i \neq \emptyset$, and since the system is stable there is a node j of $i's$ neighbour such that $j \in Q(i) = P(i)$ and $j \notin D$, then $y(j)$ is 1 but $x(j)$ is 0, a contradiction, so D is a WDS. We now claim D is minimal weight as well. Suppose there exists a subset $D' \subset D$ is a WDS such that $w(D') < w(D)$. For some node $j \in D - D'$, there is a node $i \in N(j)$ that points to it. That means $P(i) = \{..., j\} \neq \emptyset$. Since no node in the system is privileged, we have $P(i) = Q(i)$, and we must have $w(N(i) \cap D) \leq t(i)$. Thus, the removal of j from D will leave $w(N(i) \cap D) < t(i)$ and D is not a WDS (since the weight value of each node is positive). So the WDS D is minimal weight. ■

5.5.1 An Illustration

Example 5.1. *The example in Fig. 5.1 illustrates the execution of Algorithm 5.1. Note that in each configuration, the shaded nodes represent nodes in D (x-value is 1). The privileged node selected by the central daemon according to Algorithm 5.1 to make a move or reset the value of $P(i)$.*

In the first subgraph of Fig. 5.1, we set $w(1) = w(3) = w(5) = 1$ and $w(2) = w(4) = w(6) = 2$. The weight targets are $t(1) = 4,\ t(2) = t(5) = 3,\ t(3) = t(4) = 2$, and $t(6) = 1$. We further set $x(1) = x(3) = 1$, other nodes' x-values are 0, that means $D = \{1,\ 3\}$. $P(1) = P(2) = P(4) = P(5) = \{1\}$, $P(3) = P(6) = \{3\}$, just as the arrows point in the first subgraph. According to the definition of Algorithm 5.1, after a series of moves and resets, the system reaches a legitimate state. As the last subgraph of Fig. 5.1 shows, which is a legitimate configuration, we can see a minimal weighted dominating set $D = \{1,\ 2,\ 3,\ 4\}$ can be identified (the shaded nodes) and the minimal weight of D is 6.

5.6 The Stabilization Time of Algorithm 5.1

In this section, we prove the convergence of Algorithm 5.1. We say that node i *invites* nodes M_i if, at some time t, the total weight of node $i's$ neighbours in D has less than $t(i)$ and then executes the rule, causing $P(i) = (D \cap N(i)) \cup M_i$. For a node to join D, it must either be pointed to from an initial erroneous state or be invited. As the proof of Lemma 5.1, we use the terminology that node i invites node j (with $j = i$ allowed) if at some time $w(N(i) \cap D) < t(i)$, $j \in M_i$, j executes a move. For a node to join D, it must be pointed to from an initial state or be invited.

Definition 5.5. *A move is an in-move if it causes $x(i)$ to become 1, thereby causing a node i to enter D.*

Lemma 5.2. *Let i be a node and suppose that between two moves t and t', there is no in-move by any node $k > i$. Then during this move interval node i can make at most two in-moves.*

Proof: If i is never invited during this interval, then once i leaves D, it cannot re-join. The first in-move made by i may have been because a neighbour node happened to initially point to i. The second in-move made by i must be by invitation. So suppose i is invited by node j, allowing i to make an in-move. Once

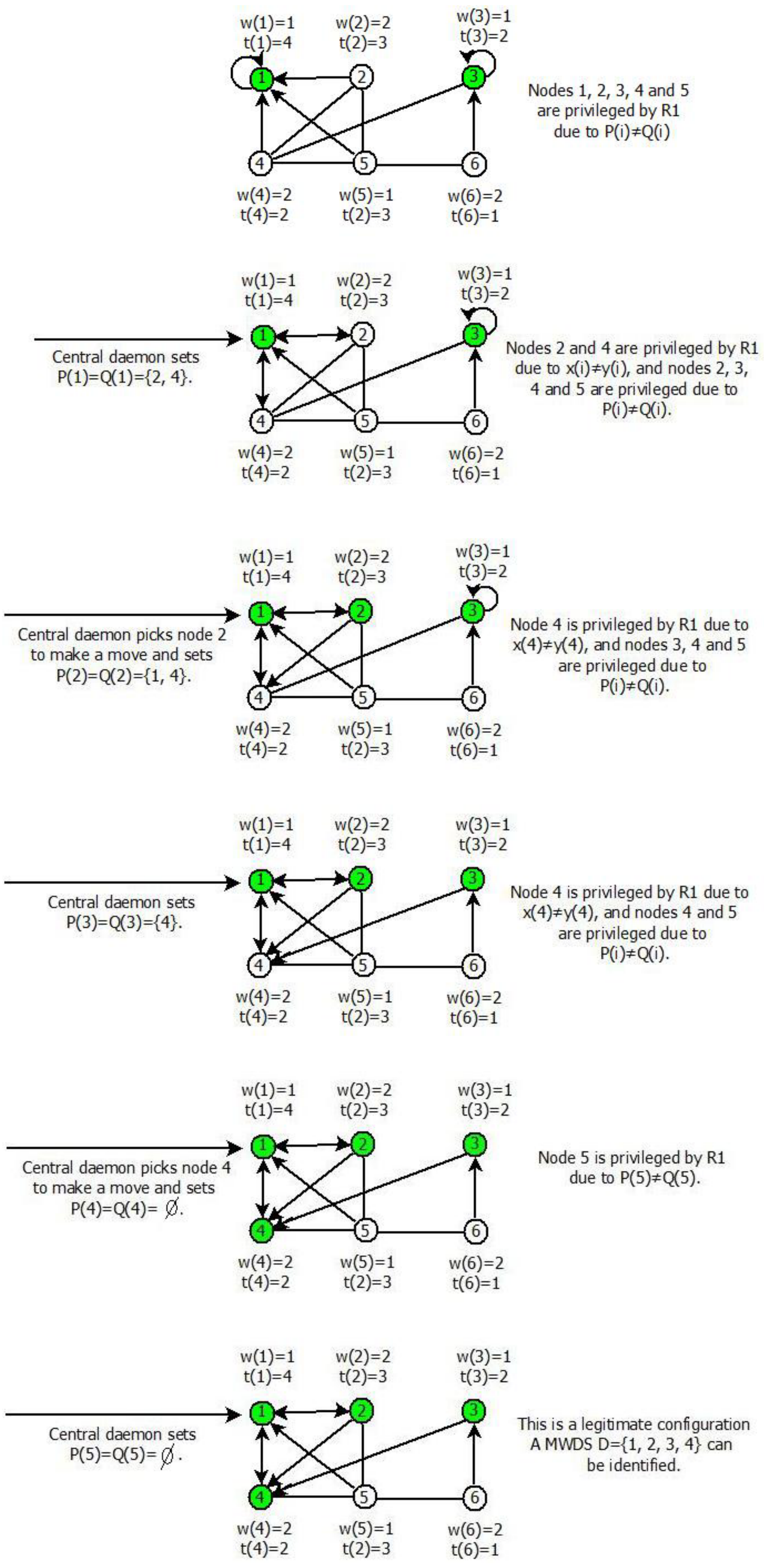
w(1)=1 t(1)=4
w(2)=2 t(2)=3
w(3)=1 t(3)=2
w(4)=2 t(4)=2
w(5)=1 t(2)=3
w(6)=2 t(6)=1
Nodes 1, 2, 3, 4 and 5 are privileged by R1 due to P(i)≠Q(i)
Central daemon sets P(1)=Q(1)={2, 4}.
Nodes 2 and 4 are privileged by R1 due to x(i)≠y(i), and nodes 2, 3, 4 and 5 are privileged due to P(i)≠Q(i).
Central daemon picks node 2 to make a move and sets P(2)=Q(2)={1, 4}.
Node 4 is privileged by R1 due to x(4)≠y(4), and nodes 3, 4 and 5 are privileged due to P(i)≠Q(i).
Central daemon sets P(3)=Q(3)={4}.
Node 4 is privileged by R1 due to x(4)≠y(4), and nodes 4 and 5 are privileged due to P(i)≠Q(i).
Central daemon picks node 4 to make a move and sets P(4)=Q(4)= ∅.
Node 5 is privileged by R1 due to P(5)≠Q(5).
Central daemon sets P(5)=Q(5)= ∅ .
This is a legitimate configuration A MWDS D={1, 2, 3, 4} can be identified.

Figure 5.1: An example to illustrate the execution of Algorithm 5.1.

i enters D it must remain there if j continues pointing to it. And this is ensured, provided $w(N(j) \cap D) \leq t(j)$ throughout. Suppose during the move interval from t to t', $w(N(j) \cap D) = t'(j) < t(j)$. Nodes having weight values larger than that of i do not move during this period, but the nodes with smaller weight values can. during the move interval from t to t', i is among the nodes /node with smallest weight value in $N(j) - D$. Even if all nodes with smaller weight values than that of i were to enter D, we would still have $w(N(j) \cap D) \leq t(j)$. It follows that j will remain pointing to i throughout, and i will remain in D. Hence, $x(i)$ can make at most two in-moves during this move interval. ■

We now prove Algorithm 5.1 stabilizes. Observe that if D remains the same, then every node can execute at most once (to correct its pointer). So it suffices to show that D changes at most a finite number of times.

Theorem 5.1. *Algorithm 5.1 always stabilizes, and finds an MWDS.*

Proof: In light of Lemma 5.1 we see that if Algorithm 5.1 is stabilizing it always finds an MWDS. We need only prove stabilization. It suffices to show that every node makes only a finite number of in-moves. By Lemma 5.2, node n, which has the largest ID, makes at most two in-moves. During each of the move intervals from t to t', the pointer set $P(i)$ makes a finite number of moves since it only use the information of its neighbours. when node n is not making an in-move, using Lemma 5.2 again, node $n-1$ makes at most a finite number of moves and resets. It is easy to show this argument can be repeated, showing that each node can make only a finite number of in-moves and resets during the intervals in which larger weight value nodes are inactive. ■

We provide a correctness proof and a computation of the worst case stabilization time for Algorithm 5.1.

Theorem 5.2. *Algorithm 5.1 produces an MWDS and stabilizes in $O(n^2)$ steps.*

Proof: From Lemma 5.1 and Theorem 5.1, we see that Algorithm 5.1 produces an MWDS. We need only prove Algorithm 5.1 stabilizes in $O(n^2)$ steps. By Lemma 5.2, each node will change its x-value at most twice. Therefore, there can be at most $2n$ changes of x-value on all nodes in all the time. If there is no change in x-value of any node in a time-step, then the time-step involves only changes in $P(i)$-values. The change in a $P(i)$-value is determined only by $Q(i)$-values. Since we are working with the central daemon, there cannot be two consecutive time-steps without a change in x-value or $P(i)$-value. Therefore, there can be at most $n\Delta$ changes of $P(i)$-value on all nodes during the whole time (where Δ is the maximum degree of G). So, the upper bound of the execution time is $(2+\Delta)n$ time-steps. Considering that the graph G is a simple undirected graph, we have $\Delta \leq (n-1)$. Therefore, the stabilization time of Algorithm 5.1 is $O(n^2)$ steps. ■

5.7 Related Work and Comparisons

In this section we present and discuss the existing algorithms for the dominating set (DS), especially for the weighted dominating set (WDS) respectively. We also discuss the basic idea of Algorithm 5.1.

Hedetniemi et al. [54] presented two self-stabilizing algorithms for the DS and the minimal dominating set (MDS) problems. The idea of the first algorithm is to partition the set of nodes into two disjoint sets, such that each set is dominating. To obtain this result, each node has a boolean variable that indicates whether it is in the first or the second set. Then, a node is allowed to change its state if all nodes in its neighbourhood have the same state. In the second algorithm, which calculates an MDS, each node maintains a boolean variable that indicates after stabilization whether the node is in the MDS or not. Also, each node maintains another variable (pointer) that indicates if it is dominated by only one node. So, a node will point to null if it is in the set under construction S or it is dominated by more than one node; otherwise it will point to the unique node that dominates it. The algorithm allows a node to join the set S, if it has no neighbour in S. On the other hand, a node that is already a member of S, and has a neighbour that is also a member of S, will leave the set if all its neighbours are not pointing to it. Thus, after stabilization the set S will be an MDS. The two algorithms work for any connected graph and assume a central daemon.

Turau [59] designed a self-stabilizing algorithm for the MDS problem assuming the nodes to have globally unique identifiers and an arbitrary graph. The idea of his algorithm is that if a node is dominated by exactly one neighbour, then it will point to this neighbour. And if a node is dominated by more than one neighbour, or it is a member of the set under construction, then it will point to null. This algorithm allows a node to join the set under construction if: a node that has no neighbour in the set will first change its state to *WAIT*, then it will change its state to *IN* if it has no neighbour with a lower identifier in *WAIT* state.; however, in order to leave the set under construction a node must have no neighbour pointing to it. Note that this is the first MDS algorithm that stabilizes in linear time under a distributed daemon.

Zou et al. [16] gave the first polynomial time approximation scheme (PTAS) for the MWDS problem on unit disk graphs, in which the sensor nodes are assumed to lie on the Euclidean plane, and there is an edge between two nodes iff their Euclidean distance is no more than one. The definition of the MWDS in their paper is to seek a subset of the vertices of the graph with minimum total weight such that each vertex of the graph is either in the subset or adjacent to some nodes in the subset. For any $\epsilon > 0$, their greedy algorithm can achieve a $(4 + \epsilon)$-approximation for the MWDS based on a polynomial-time dynamic programming.

However, there is no self-stabilizing algorithm for the weighted domination problems in arbitrary graphs that works under a central daemon. Most of the proposed self-stabilizing algorithms all work under the uniform weight [54, 59]. The weight of each node in previous algorithms discussed in this section is all uniform. The best greedy algorithm for the WDS problem is about $(4 + \epsilon)$-approximation for any $\epsilon > 0$. In order to select a minimal weight dominating set (MWDS) in general networks, we present a uniform algorithm (all of the individual processes run the same code) for finding an MWDS that works in arbitrary graphs under a central daemon.

The main idea of Algorithm 5.1 is that we assume globally unique identifiers for the nodes and a central daemon. The algorithm uses a mechanism of pointers to show that a node i will point to its neighbours having the smallest weight values if i has been dominated less than $t(i)$ weight value by the set D when constructing it. On the other hand, if a node i has been dominated more than $t(i)$ weight value by the set D then $P(i)$ will point to an empty set; otherwise $P(i)$ will point to its unique neighbours that are members of the set D. The algorithm allows a node to join the set D if some neighbour is pointing to it, and to leave the set D otherwise. So after stabilization, the set D will become an MWDS. The time complexity of our algorithm in any arbitrary graphs is $O(n^2)$ steps. To the best of our knowledge, this is the first work using a self-stabilizing algorithm to find an MWDS, which can find an exact solution to the WDS problem (minimal).

Table 5.1: Self-stabilizing algorithms for the dominating set

Reference	**Output**	**Topology**	**Self-stabilizing**	**Daemon**	**Complexity**
Hedetniemi et al. [54]-1	DS	Arbitrary	Yes	Central	$O(n)$ steps
Hedetniemi et al. [54]-2	MDS	Arbitrary	Yes	Central	$O(n^2)$ steps
Turau [59]	MDS	Arbitrary	Yes	Distributed	$O(n)$ steps
Zou et al. [16]	WMDS	Arbitrary	No (Greedy)		$(4+\epsilon)$-approxim
Algorithm 5.1	MWDS	Arbitrary	Yes	Central	$O(n^2)$ steps

The algorithms we compared in this section are summarized in Table 5.1. As we can see, the basic ideas of the first three algorithms are self-stabilizing; and all of them deal with the dominating set with uniform weight cases. Zou et al.'s work is a greedy approximation algorithm to deal with the MWDS case. Our algorithm is the first work using a self-stabilizing approach to discuss the weighted domination problem.

5.8 Summary

In this chapter, we have proposed a self-stabilizing algorithm to find an MWDS which arises from some cost issues in networks. The algorithm can be used for an arbitrary connected graph. Previously known algorithms are self-stabilizing algorithms considering uniform weight cases or greedy algorithms. We have also shown the stabilization time of the algorithm with $O(n^2)$ steps under a central daemon. We briefly discuss how the ideas can be further generalized.

We can extend these ideas even further to other graph dominations such as weak, strong and optional domination [100, 101]. It can also be altered to allow a node to have weights in a range $\{-b'(i), ..., b(i)\}$ and so handle minus domination. Another interesting direction for further research is to develop self-stabilizing algorithms for weighted domination problems that operate under a distributed or synchronous daemon.

Chapter 6

A New Algorithm for the Positive Influence Dominating Set

A positive influence dominating set (PIDS) in a graph is a set of nodes such that each node in the graph has at least half of its neighbours in the set. In the past, a few algorithms for the PIDS problem have been studied in the literature. However, all the existing work focused on greedy algorithms for the PIDS problem with different approximation ratios, which are limited to finding approximate solutions to a PIDS in large social networks. In order to select a minimal PIDS (MPIDS) in large social networks, we first present a self-stabilizing algorithm for the MPIDS problem. It can work for any general graphs under a central daemon. We further prove that the worst case convergence time of the algorithm from any arbitrary initial state is $O(n^2)$ steps where n is the number of nodes in the network.

The information in this chapter is based on one published paper [73].

6.1 Introduction

6.1.1 Positive Influence Dominating Set

Recently, social networks for example, friendship networks, telephone call networks, and academia co-authorship networkshave received dramatic interest in research and development, partly due to more and more social networks being built online and the fast development of Web 2.0 applications, such as Facebook [102–104], Twitter [32–34], e-learning research [78, 105] and privacy protection [106–108]. However, social networks have also introduced to the research community many new challenges. Graph theory has been considered a working solution to tackle these challenges. Some classical graph problems such as domination problems in social networks have various new applications.

In a social network which consists of individuals with a certain type of social problem, people can have both positive and negative impact on each other, and a person can take and switch among different roles since they are affected by their peers [82–84]. For example, for the drinking problem, a person named

Mark never drinks and has positive impact on his direct friends (Barry, etc.), but Mark might turn into an alcoholic and has negative impact on his neighbours if his friends Barry et al. are all alcoholics and vice versa. So, a person can be an abstainer or an alcoholic. In order to truly alleviate the main source of the drinking problem, intervention programmes are important tools to help alleviate some of the social problems through disseminated education and therapy via mail, Internet, or face-to-face interviews. Ideally, we want to educate all alcoholics, since this will reduce the possibility of converted alcoholics being influenced by their alcoholic friends who are not selected in the intervention programme. On the other hand, due to budget limitations, the lower the total cost of the education and therapy programme, the better. So, it is too expensive to include all the alcoholics in the intervention programme. Therefore, it becomes an important research problem as to how to choose a subset of individuals to be part of the programme so that the effect of the intervention programme can spread through the whole group under consideration.

The concept of the positive influence dominating set (PIDS) was introduced in 2009 by Wang et al. [27,28]. The PIDS can deal with some social problems, such as drinking, smoking and drug use related issues. Formally, a social network can be represented as a graph $G = (V, E)$, where $i \in V$ represents a person (node) in the social network and edge $e_{ij} \in E$ represents a relationship between persons i and j. Recall that $D \subseteq V$ is a *positive influence dominating set* (PIDS) [27,28] such that any node i in V is dominated by at least $\lceil \frac{d(i)}{2} \rceil$ nodes (that is, i has at least half of its neighbours) in D where $d(i)$ is the degree of node i. Note that there are two requirements for PIDS: firstly, every node not in D has at least half of its neighbours in D, secondly every node in D also has at least half of its neighbours in D. A PIDS D is *minimal* (MPIDS) if no proper subset of D is a PIDS.

For the drinking example remarked on earlier, an MPIDS is a plausible solution since the MPIDS can guarantee that by selecting MPIDS nodes to participate in the intervention programme, each individual in the social network has more positive neighbours than negative ones to ensure that the intervention can result in a globally positive impact on the entire social network.

Wang et al. proposed a greedy approximation PIDS selection algorithm and analyzed its effect on a real online social network data set and through simulations [27]. Their simulation results reveal that approximately 60% of the whole group under consideration needs to be selected into the PIDS to achieve the goal that every individual in the community has more positive neighbours than negative neighbours. It also reveals that by strategically selecting 26% more people into the PIDS to participate in the intervention programme, the average positive degree increases by approximately 3.3 times. It means that by moderately increasing the participation related cost, the probability of positive influencing the whole community through the intervention programme is significantly higher.

Wang et al. also proved the PIDS problem is APX-hard (APX-hardness of the PIDS problem means that if $NP \neq P$, then PIDS has no polynomial-time approximation scheme) and developed another greedy approximation algorithm with an approximation ratio of $H(\delta)$ where H is the harmonic function and δ is the maximum vertex degree of the graph representing a social network [28].

6.2 Self-Stabilization

Self-stabilization is a paradigm for distributed systems that allows the system to achieve a desired global state, even in the presence of faults [56, 57, 89]. A self-stabilizing programme consists of a collection of rules of the form if the condition then changes state. A node is called *privileged* if the predicate of one of the rules is true. It *moves* by changing state. A fundamental idea of self-stabilizing algorithms is that no matter what global state the system finds itself in, after a finite amount of time the system will reach a correct and desired global state. Although the concept of self-stabilization was introduced in 1974 by Dijkstra [56], serious work on self-stabilizing algorithms did not start until the late 1980's.

An algorithm is *self-stabilizing* if

(i) for any initial illegitimate state it reaches a legitimate state after a finite number of node moves (convergence), and

(ii) for any legitimate state and for any move allowed by that state, the next state is a legitimate state (closure).

The convergence property ensures that, starting from any incorrect state, the distributed system reaches a correct state. The closure property ensures that, after convergence, the system remains in the set of correct states.

We assume that each transition from one configuration to another is driven by a *scheduler*, also called a *daemon*. A *central daemon* selects, among all privileged nodes, the next node to move. We assume a serial model in which no two nodes move simultaneously. If two or more nodes are privileged, one cannot predict which node will move next.

An *unfair* daemon can choose any node among those privileged nodes without any consideration of fairness. The consequence of using an unfair daemon is that a privileged node may never be chosen as long as there is at least one other privileged node at each step.

Under this computational model, a great number of papers regarding self-stabilizing algorithms have been published for graph problems. Such as, self-stabilizing algorithms for dominating sets, independent sets, colorings, and matchings in graphs have been developed [54, 61, 64, 88].

6.3 Motivations and Contributions

The PIDS which has application in social networks can be considered as a special case of multiple domination, introduced by Wang et al. [27, 28]. Wang et al. proposed a greedy approximation PIDS selection algorithm and analyzed its effect on a real online social network data set and through simulations [27]. Wang et al. also proved the PIDS problem is APX-hard and developed another greedy approximation algorithm and analyzed its approximation ratio [28]. APX-hardness of the PIDS problem means that if $NP \neq P$, then the PIDS problem has no polynomial-time approximation scheme.

In the domination problems, finding a PIDS of minimum size is APX-hard [28]. Some greedy approximation algorithms have been proposed [27,28], which are all limited to finding approximate solutions to PIDSs in large social networks. If we can obtain a smaller size solution to the PIDS, it might save the total cost of an intervention and education programme while positively dominating the whole group. Moreover, it might offer considerable benefits to both the economy and society.

On the other hand, none of these algorithms for the PIDS problem are self-stabilizing. In order to obtain a minimal PIDS (MPIDS), we first propose a new self-stabilization algorithm for computing an MPIDS in a general network and analyze the time complexity of the proposed algorithm. To the best of our knowledge, this is the first work using a self-stabilizing algorithm to find an MPIDS. The following are our contributions.

1. A self-stabilizing algorithm for finding a minimal positive influence dominating set (MPIDS) in an arbitrary connected network graph under a central daemon is presented.
2. The correctness of the proposed algorithm is verified. The worst case convergence time of the algorithm, which is $O(n^2)$ steps from any arbitrary initial state where n is the number of nodes in the network, is proved.

6.4 Positive Influence Dominating Set

Let $G = (V, E)$ be a simple connected undirected graph with nodes set V and edges set E. Assume now that for each node $i \in V$, the set $N(i)$ represents its *open neighbourhood*, denotes the set of nodes to which i is adjacent. $d(i)$ represents the number of neighbours of node i, or its degree ($d(i) = |N(i)|$). Our algorithm requires that every node has a unique ID. Sometimes i interchangeably denotes a node or the node's ID. Assume there is a total ordering on the IDs. Assume further that each node has a target integer $h(i) \leq d(i)$, where $h(i) = \lceil \frac{d(i)}{2} \rceil$ indicates that any node $i \in V$ is dominated by at least $\lceil \frac{d(i)}{2} \rceil$ nodes in $N(i)$. Given these assumptions we define a minimal positive influence dominating set $D \subseteq V$ as follows:

Definition 6.1. *A subset $D \subseteq V$ is said to be a **positive influence dominating set (PIDS)** if for every node $i \in V$,*

$$|N(i) \cap D| \geq h(i). \tag{6.1}$$

*A PIDS D is a **minimal** PIDS (MPIDS) if any proper subset of D is not a PIDS in G.*

Note that in the case of total domination (a set $D \subseteq V$ is said to be total dominating if every $i \in V$ is adjacent to at least a member of D), the $h(i)$ in Inequality (6.1) is a precisely uniform one.

The concept of an MPIDS problem gives rise to the following problem:

Minimal Positive Influence Dominating Set

INSTANCE: A connected simple undirected graph $G = (V, E)$ with nodes set V and edges set E.

QUESTION: How to find an MPIDS $D \subseteq V$ such that the D is a PIDS of the graph G and minimal.

6.5 Positive Dominating Set Algorithm

In this section, we present our self-stabilizing algorithm. In a graph without isolated nodes, a PIDS must exist.

In our algorithm, assume each node i has two variables: a set of pointers $P(i)$ and a Boolean flag $x(i)$. If $P(i) = \{j\}$, then we say that i points to j, written $i \to j$. We allow $P(i)$ to contain i and its cardinality is bounded by $h(i)$. Each node also has a Boolean flag $x(i)$. At any given time, we will denote with D the current set of nodes i with $x(i) = true$.

At a given time, assume $|N(i) \cap D| = k \leq h(i)$. Then since $h(i) \leq |N(i)|$, there are at least $h(i) - k$ members in $N(i) - D$.

Definition 6.2. *Let M_i denote the unique set of those $h(i)-k$ nodes in $N(i)-D$ having the smallest IDs.*

Note this set depends on $N(i)$ and D.

Definition 6.3. *A set of pointers $Q(i)$ is designed as follows:*

$$Q(i) = \begin{cases} (D \cap N(i)) \cup M_i & \text{if } |N(i) \cap D| = k \leq h(i) \\ \emptyset & \text{if } |N(i) \cap D| > h(i). \end{cases} \tag{6.2}$$

Note that the value $Q(i)$ can be computed by i (i.e., it uses only local information).

Definition 6.4. *The Boolean condition $y(i)$ is defined to be true if and only if a neighbour of i points to it.*

6.5.1 Proposed Algorithm

Algorithm 6.1: Finding a Minimal Positive Influence Dominating Set

```
Input: A graph $G = (V, E)$, $\forall i \in V$, a Boolean
flag $x(i)$ and a set of pointers $P(i)$
 Output: $D = \{i \in V | x(i) = true\}$
 R1: if  $x(i) \neq y(i) \vee P(i) \neq Q(i)$
      then  $x(i) = y(i) \wedge P(i) = Q(i)$
```

The Algorithm 6.1, consisting of one rule (R1), is shown above. In Algorithm 6.1 each node i has a Boolean variable $x(i)$ indicating membership in the set D that we are trying to construct. The value $x(i) = true$ indicates that $i \in D$, while the value $x(i) = false$ indicates that $i \notin D$. The Boolean condition $y(i)$ is defined to be true if and only if a neighbour of i points to it. $P(i)$ is a set of pointers. $Q(i)$ is counted by Equation (6.2). Thus, a node i is *privileged* if $x(i) \neq y(i)$ or $P(i) \neq Q(i)$. If R1 executes, then it sets $x(i) = y(i)$ and $P(i) = Q(i)$. An example to illustrate the execution of Algorithm 6.1 is shown in Fig. 6.1.

It is obvious that the system is in a *legitimate configuration* if and only if no node in the system is privileged. The following Lemma clarifies that in any legitimate configuration, an MPIDS $D = \{i \in V | x(i) = true\}$ can be identified.

Lemma 6.1. *If Algorithm 6.1 stabilizes then* $D = \{i \in V | x(i) = true\}$ *is an MPIDS satisfying Inequality (6.1).*

Proof: Suppose that D satisfies Inequality (6.1). By contradiction suppose that for some i, $|N(i) \cap D| < h(i)$. Then $M_i \neq \emptyset$, and since the system is stable there is a node j of $i's$ neighbour such that $j \in Q(i) = P(i)$ and $j \notin D$, then $y(j)$ is true but $x(j)$ is false, a contradiction, so D is a PIDS. We now claim D is minimal as well. Suppose there exists a subset $D' \subset D$ is a PIDS. For some node $j \in D - D'$, there is a node $i \in N(j)$ that points to it. That means $P(i) = \{..., j\} \neq \emptyset$. Since no node in the system is privileged, we have $P(i) = Q(i)$, and we must have $|N(i) \cap D| = h(i)$ according to the Equation (6.2). Thus, the removal of j from D will leave $|N(i) \cap D| < h(i)$ and D is not a PIDS. So the PIDS D is minimal. ■

6.5.2 An Illustration

Example 6.1. *The example in Fig. 6.1 illustrates the execution of Algorithm 6.1. Note that in each configuration, the shaded nodes represent nodes in D (x-value is true). The privileged node selected by the central daemon according to Algorithm 6.1 to make a move or reset the value of* $P(i)$.

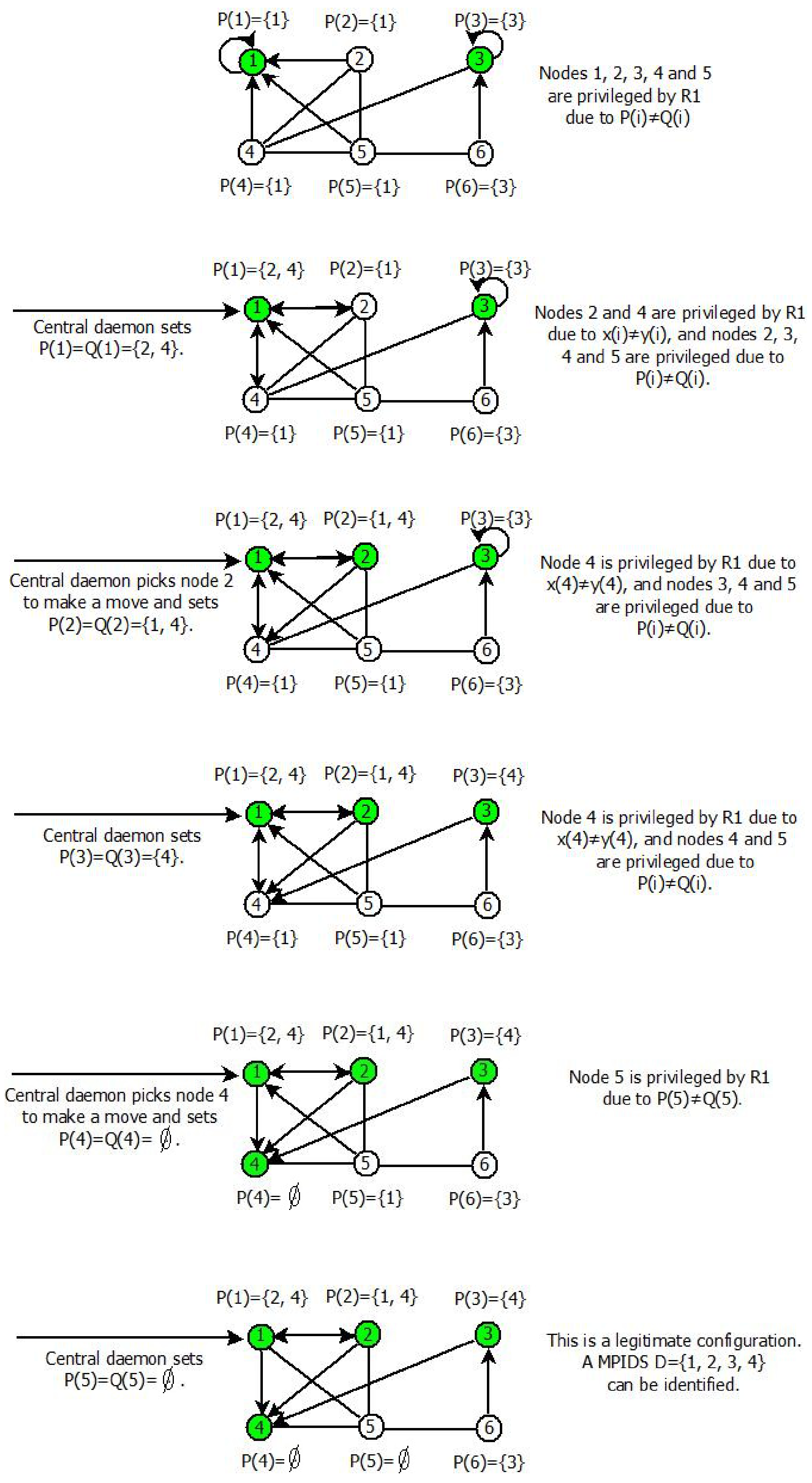
P(1)={1}
P(2)={1}
P(3)={3}
P(4)={1}
P(5)={1}
P(6)={3}
Nodes 1, 2, 3, 4 and 5 are privileged by R1 due to P(i)≠Q(i)
P(1)={2, 4}
P(2)={1}
P(3)={3}
Central daemon sets P(1)=Q(1)={2, 4}.
Nodes 2 and 4 are privileged by R1 due to x(i)≠y(i), and nodes 2, 3, 4 and 5 are privileged due to P(i)≠Q(i).
P(4)={1}
P(5)={1}
P(6)={3}
P(1)={2, 4}
P(2)={1, 4}
P(3)={3}
Central daemon picks node 2 to make a move and sets P(2)=Q(2)={1, 4}.
Node 4 is privileged by R1 due to x(4)≠y(4), and nodes 3, 4 and 5 are privileged due to P(i)≠Q(i).
P(4)={1}
P(5)={1}
P(6)={3}
P(1)={2, 4}
P(2)={1, 4}
P(3)={4}
Central daemon sets P(3)=Q(3)={4}.
Node 4 is privileged by R1 due to x(4)≠y(4), and nodes 4 and 5 are privileged due to P(i)≠Q(i).
P(4)={1}
P(5)={1}
P(6)={3}
P(1)={2, 4}
P(2)={1, 4}
P(3)={4}
Central daemon picks node 4 to make a move and sets P(4)=Q(4)= ∅.
Node 5 is privileged by R1 due to P(5)≠Q(5).
P(4)= ∅
P(5)={1}
P(6)={3}
P(1)={2, 4}
P(2)={1, 4}
P(3)={4}
Central daemon sets P(5)=Q(5)= ∅.
This is a legitimate configuration. A MPIDS D={1, 2, 3, 4} can be identified.
P(4)=∅
P(5)=∅
P(6)={3}

Figure 6.1: An example to illustrate the execution of Algorithm 6.1 .

In the first subgraph of Fig. 6.1, we set $x(1) = x(3) = true$, other nodes' x-values are false, that means $D = \{1,\ 3\}$. $P(1) = P(2) = P(4) = P(5) = \{1\}$, $P(3) = P(6) = \{3\}$, just as the arrows point in the first subgraph. According to the rule (R1) of Algorithm 6.1 , after a series of moves, the system reaches a legitimate state. As the last subgraph of Fig. 6.1 shows, which is a legitimate configuration, we can see a minimal positive influence dominating set $D = \{1,\ 2,\ 3,\ 4\}$ can be identified (the shaded nodes).

6.5.3 The Stabilization Time of Algorithm 6.1

In this section, we prove the convergence of Algorithm 6.1 . We say that node i *invites* node j (with $j = i$ allowed) if at some time $|N(i) \cap D| < h(i)$, $j \in M_i$, j executes a move. For a node to join D, it must be pointed to from an initial state or be invited.

Definition 6.5. *A move is an in-move if it causes $x(i)$ to become true, thereby causing a node i to enter D.*

Lemma 6.2. *Let i be a node and suppose that between two moves t and t', there is no in-move by any node $k > i$. Then during this move interval node i can make at most two in-moves.*

Proof: If i is never invited during this interval, then once i leaves D, it cannot re-join. The first in-move made by i may have been because a neighbour node happened to initially point to i. The second in-move made by i must be by invitation. So suppose i is invited by node j, allowing i to make an in-move. Once i enters D it must remain there if j continues pointing to it. And this is ensured, provided $|N(j) \cap D| \leq h(j)$ throughout. Suppose during the move interval from t to t', $|N(j) \cap D| = k$. Nodes having IDs larger than that of i do not move during this period, but the smaller nodes can. during the move interval from t to t', i is among the $h(j) - k$ smallest nodes in $N(j) - D$. Even if all nodes with smaller IDs than that of i were to enter D, we would still have $|N(j) \cap D| \leq h(j)$. It follows that j will remain pointing to i throughout, and i will remain in D. Hence, $x(i)$ can make at most two in-moves during this move interval. ■

We now prove our algorithm stabilizes. Observe that if D remains the same, then every node can execute at most once (to correct its pointer). So it suffices to show that D changes at most a finite number of times.

Theorem 6.1. *Algorithm 6.1 always stabilizes, and finds an MPIDS.*

Proof: In light of Lemma 6.1 we see that if Algorithm 6.1 is stabilizing it always finds an MPIDS. We need only prove stabilization. It suffices to show that every node makes only a finite number of in-moves. By Lemma 6.2, node n, which has the largest ID, makes at most two in-moves. During each of the move intervals from t to t', the pointer set $P(i)$ makes a finite number of moves since it only use the information of its neighbours. When node n is not making an in-move, using Lemma 6.2 again, node $n-1$ makes at most a finite number of moves and resets. It is easy to show this argument can be repeated, showing that each node can make only a finite number of in-moves and resets during the intervals in which larger nodes are inactive. ■

We provide a correctness proof and a computation of the worst case stabilization time for Algorithm 6.1.

Theorem 6.2. *Algorithm 6.1 produces an MPIDS and stabilizes in $O(n^2)$ steps.*

Proof: From Lemma 6.1 and Theorem 6.1, we see that Algorithm 6.1 produces an MPIDS. We need only prove Algorithm 6.1 stabilizes in $O(n^2)$ steps. By Lemma 6.2, each node will change its x-value at most twice. Therefore, there can be at most $2n$ changes of x-value on all nodes during the whole time. If there is no change in x-value of any node in a time-step, then the time-step involves only changes in $P(i)$-values. The change in a $P(i)$-value is determined only by $Q(i)$-values. Since we are working with the central daemon, there cannot be two consecutive time-steps without a change in x-value or $P(i)$-value. Therefore, there can be at most $\lceil\frac{\Delta}{2}\rceil n$ changes of $P(i)$-value on all nodes during the whole time (where Δ is the maximum degree of G). So, the upper bound of the execution time is $(\lceil\frac{\Delta}{2}\rceil + 2)n$ time-steps. Considering the graph G is a simple undirected graph, therefore, the stabilization time of Algorithm 6.1 is $O(n^2)$ steps. ■

6.6 Related Work and Comparison

In this section we present and discuss the existing self-stabilizing algorithms and greedy algorithms for the dominating set respectively. We also discuss the main idea of Algorithm 6.1.

Hedetniemi et al. presented two self-stabilizing algorithms for the dominating set (DS) and the minimal dominating set (MDS) problems. The algorithms work for any connected graph. The algorithm for the DS problem stabilizes in linear time ($O(n)$ steps) under a central daemon. The second algorithm calculates an MDS and stabilizes in $O(n^2)$ steps under a central daemon.

Goddard et al. [58] gave a self-stabilizing algorithm working on the minimal total dominating set (MTDS) problem. The algorithms works under a central daemon for any general graphs. Goddard et al. [88] proposed another uniform self-stabilizing algorithm for finding an MDS in an arbitrary graph under a distributed daemon. The algorithm stabilizes in $O(n)$ steps.

On the other hand, some self-stabilizing algorithms have been proposed in the k-domination case. Kamei and Kakugawa [26] presented two uniform algorithms for the minimal k-dominating set (MKDS) problem in a tree under a central daemon. The second algorithm works under a distributed daemon. The time complexity of the two algorithms are both $O(n^2)$ steps.

Huang et al. presented two self-stabilizing algorithms to find a minimal 2-dominating set (M2DS) in an arbitrary graph [55, 68]. The first algorithm works under a distributed daemon [68] and the second algorithm works under a central daemon with linear time complexity [55].

We presented two self-stabilizing algorithm for finding an MKDS [71, 72] in any general graphs. The first algorithm works for the MKDS problem in general graphs under a central daemon [71] and the second one works for the MKDS problem under a distributed daemon [72]. We also proposed a self-stabilizing algorithm for finding a minimal weighted dominating set (MWDS) in any general graphs [75]. The algorithm works under a central daemon in general graphs.

Wang et al. introduced the notion of the positive influence dominating set (PIDS) and proposed a greedy approximation PIDS selection algorithm in 2009. They revealed that approximately 60% of the whole group under consideration needs to be selected into the PIDS to achieve the goal that every individual in the community has more positive neighbours than negative neighbours.

Wang et al. also presented another greedy approximation algorithm and gave theoretical analysis about its approximation ratio (AR) in 2011 [28]. The authors proved that PIDS is APX-hard and proposed a greedy PIDS selection algorithm with an approximation ratio of $H(\delta)$ where H is the harmonic function and δ is the maximum vertex degree of the graph representing a social network.

Table 6.1: Algorithms for the dominating set

Reference	**Output**	**Topology**	**Self-stabilizing**	**Daemon**	**Complexity**
Hedetniemi et al. [54]-1	DS	Arbitrary	Yes	Central	$O(n)$ steps
Hedetniemi et al. [54]-2	MDS	Arbitrary	Yes	Central	$O(n^2)$ steps
Goddard et al. [58]	MTDS	Arbitrary	Yes	Central	
Goddard et al. [88]	MDS	Arbitrary	Yes	Distributed	$O(n)$ steps
Kamei & Kakugawa [26]-1	MKDS	Tree	Yes	Central	$O(n^2)$ steps
Kamei & Kakugawa [26]-2	MKDS	Tree	Yes	Distributed	$O(n^2)$ steps
Huang et al. [68]	M2DS	Arbitrary	Yes	Distributed	
Huang et al. [55]	M2DS	Arbitrary	Yes	Central	$O(n)$ steps
Wang et al. [71]	MKDS	Arbitrary	Yes	Central	$O(n^2)$ steps
Wang et al. [72]	MKDS	Arbitrary	Yes	Distributed	$O(n^2)$ steps
Wang et al. [75]	MWDS	Arbitrary	Yes	Central	$O(n^2)$ steps
Wang et al. [27]	PIDS	Arbitrary	Greedy		
Wang et al. [28]	PIDS	Arbitrary	Greedy		$H(\delta)$ AR
Algorithm 6.1	MPIDS	Arbitrary	Yes	Central	$O(n^2)$ steps

In order to select a small size of PIDS in large social networks, we present a uniform algorithm for finding an MPIDS that works in arbitrary graphs. We assume globally unique identifiers for the nodes and a central daemon. The algorithm uses a mechanism of pointers to show that a node i will point to its neighbours having the smallest identifiers if i has less than $h(i)$ neighbours in the set D under construction. On the other hand, if a node i has more than $h(i)$ neighbours in the set D then $P(i)$ will point to an empty set; otherwise $P(i)$ will point to its unique neighbours that are members of the set D. The algorithm allows a node to join the set D if some neighbour is pointing to it, and to leave the set D otherwise. So after stabilization, the set D will become an MPIDS. The time complexity of our algorithm in any arbitrary graphs is $O(n^2)$ steps. To the best of our knowledge, this is the first work using a self-stabilizing algorithm to find an MPIDS, which can find an exact solution to a PIDS (minimal).

The algorithms we compared in this section are summarized in Table 6.1. As we can see, the basic ideas of the first eleven algorithms are self-stabilizing, and the algorithms for the PIDS problem from Wang et al. are greedy [27, 28] . Our Algorithm 6.1 is the first work using a self-stabilizing approach to discuss the MPIDS problem.

6.7 Summary

In this chapter, we have proposed a self-stabilizing algorithm to find an MPIDS which arises from some social problems in social networks; Algorithm 6.1 can be used for an arbitrary connected graph. All previously known algorithms are approximate greedy algorithms for the PIDS problem. We have also shown the stabilization time of Algorithm 6.1 with $O(n^2)$ steps under a central daemon. We briefly discuss how the ideas can be further generalized.

One may obtain self-stabilizing algorithms for other domination problems. We can also extend these ideas even further to other graph dominations such as weak, strong and optional domination [100]. It can also be altered to allow a node to have weights in a range $\{-b'(i), ..., b(i)\}$ and so handle minus domination. Another interesting direction for further research is to develop self-stabilizing algorithms for weighted domination problems that operate under a distributed or synchronous daemon.

Chapter 7

Positive Influence Dominating Set Games

Motivated by applications in social networks, a new type of dominating set called positive influence dominating set (PIDS) has been studied in the literature. The PIDS can deal with some social problems such as drinking, smoking and drug use by education/intervention programme. From the cost point of view, we have the interest to investigate how to allocate the total cost of educating/intervening persons among an intervention program. In this chapter, we consider cooperative cost games arising from the PIDS problem which can deal with social problems. We introduce two games, the rigid PIDS game and relaxed PIDS game and focus on the cores of both games. First, a relationship between the cores of both games is obtained. Next, we prove that the core of the relaxed PIDS game is non-empty if and only if there is no integrality gap for the relaxation linear programming of the PIDS problem on the graph.

The information in this chapter is based on one paper [74].

7.1 Introduction

7.1.1 Cooperative Game

Recent years have seen an increased interest in computational complexity aspects of solution concepts in Cooperative Game Theory [109–111]. Game Theory, which arises from Combinatorial Optimization problems, is divided into two branches, called the non-cooperative and cooperative branches. In this chapter, we are going to look at the cooperative branch.

It is well known that mathematical modelling of various real-world decision making situations gives rise to Combinatorial Optimization problems. For situations where more than one decision maker is involved, classical Combinatorial Optimization Theory does not suffice, and it is here that Cooperative Game Theory can make an important contribution. If a group of decision makers decide to undertake a project together in order to increase the total revenue or decrease the total costs, they face two problems. The first one is how to execute the project in an optimal way. The second one is how to allocate the revenue or cost among the participants. Cooperative Game Theory can deal with the second problem. The solution concepts from Cooperative Game Theory can be applied to arrive at allocation methods [112].

The main fundamental question in Cooperative Game Theory is the question of how to distribute the total revenue or cost. Different requirements for fairness and rationality lead to different distributions of revenue or cost which are generally referred to solution concepts of cooperative games. Among many of these solution concepts, the core has attracted much attention from researchers [111, 113–116]. We will discuss the concept of the cooperative game and core as follows.

In Game Theory, a cooperative game is a game where groups of players ("coalitions") may enforce cooperative behaviour, hence the game is a competition between coalitions of players, rather than between individual players. The main fundamental question in Cooperative Game Theory is the question how to allocate the total generated wealth or cost by the collective of all players in the player set N itself over the different players in the game. In other words, what binding contract between the players in N has to be written? Various criteria have been developed [117].

A cooperative game consists of two elements:

(1) a set of players, and

(2) a characteristic function specifying the value created by different subsets of the players in the game.

Formally, a cooperative game $\Gamma = (N, c)$ consists of a player set $N = \{1, 2, \dots, n\}$ and a characteristic function c. Let $N = \{1, 2, \dots n\}$ be the (finite) set of players, and let i, where i runs from 1 through n, index the different members of N. The characteristic function is a function, denoted $c : 2^N \to R$, where for each subset S of N (called a coalition), $c(S)$ represents the revenue or cost incurred by the coalition of players in S without participation of other players. In sum, a cooperative game is a pair (N, c), where N is a finite set and c is a function mapping subsets of N to numbers. We consider terms only for cooperative cost games, as a symmetric statement holds for cooperative revenue games.

The main fundamental question in Cooperative Cost Game Theory is the question of how to distribute the total cost $c(N)$ among the individual players in a 'fair' way. Different requirements for fairness and rationality lead to different distributions of cost which are generally referred to solution concepts of cooperative games. Among many of these solution concepts, the core has attracted much attention from researchers. A distribution vector $x = \{x_1, x_2, \dots, x_n\}$ is called an imputation of the game $\Gamma = (N, c)$ if $\sum_{i \in N} = c(N)$ and $\forall i \in N : x_i \leq c(i)$ (individual rationality). Recall the following definitions from the note "Cooperative Game Theory: Characteristic Functions, Allocations":

(i) an allocation is a collection $x = \{x_1, x_2, \dots, x_n\}$ of numbers;

(ii) an allocation $\{x_1, x_2, \dots, x_n\}$ is *individually rational* if $\sum_{i \in N} = c(N)$ and $\forall i \in N : x_i \leq c(i)$;

(iii) an allocation $\{x_1, x_2, \ldots, x_n\}$ is *efficient* if $\sum_{i \in N} = c(N)$.

The core of the game $\Gamma = (N, c)$ is defined as:

$Core(\Gamma) = \{x \in R^n : x(N) = c(N) \text{ and } x(S) \leq c(S),\ \forall S \subseteq N\}$,

where $x(S) = \sum_{i \in S} x_i$ for $S \subseteq N$.

The set of constraints imposed on $Core(\Gamma)$ is called subgroup rationality which ensures that no coalition has an incentive to split from the grand coalition N and does better on its own. If the core of a game is non-empty, then the game is called *balanced* [114, 115].

7.1.2 Positive Influence Dominating Set Problem

The concept of the positive influence dominating set (PIDS) was introduced by Wang et al. [27, 28]. The PIDS can deal with some social problems, such as drinking, smoking and drug use related issues. In a social network which consists of individuals with a certain type of social problem, people can have both positive and negative impact on each other, and a person can take and switch among different roles since they are affected by their peers [82–84].

Formally, a social network can be represented as a graph $G = (V, E)$, where $i \in V$ represents a person (node) in the social network and edge $(i, j) \in E$ represents a relationship between persons i and j. Recall that $D \subseteq V$ is a *positive influence dominating set* (PIDS) [27, 28] if any node i in V is dominated by at least $\lceil \frac{d(i)}{2} \rceil$ positive nodes (that is, i has at least half of positive neighbours) in D where $d(i)$ is the degree of node i. Note that there are two requirements for a PIDS: firstly, every node not in D has at least half of its neighbours in D, secondly every node in D also has at least half of its neighbours in D. The PIDS problem is to find a so-called minimum PIDS of G. The number of a minimum PIDS is called the PIDS number.

In the domination problems, finding a PIDS of minimum size is APX-hard [28]. APX-hardness of the PIDS problem means that if $NP \neq P$, then the PIDS problems has no PTAS (polynomial-time approximation scheme). Some greedy approximation algorithms have been proposed [27, 28], which can find an approximate solution to a PIDS in a large social network.

7.2 Related Work

The combinatorial optimization techniques have been often utilized in much cooperative games. Especially, integer linear programming and its duality theory have proven itself a very powerful tool in the study of cores. Shapley and Shubik [113] formulated the assignment game as a two-sided market, and showed that the core is exactly the set of optimal solutions of a dual linear programming of the assignment game problem. This approach is further exploited in the study of linear production games [118, 119], packing and covering games [120]. Velzen [121] introduced three kinds of cooperative games that arise from the weighted minimum dominating set problem on a graph. It was shown that the core of each game is non-empty if and only if the corresponding linear programming relaxation of the weighted minimum dominating set problem has an integer optimal solution, and in this case, an element in the core can be found in polynomial time.

Recently, Fang et al. proposed three domination games: dominating set games [122], total dominating set games [123] and integer domination games [124]. Fang et al. established a new kind of 0-1 program formulation to model the domination problem on graphs, and gave a strong connection between LP relaxation of this 0-1 program and the cost allocation problem concerning the core of a dominating set game [122]. Fang et al. studied the balancedness (non-empty of the core) of the total dominating set games by making use of the technique of linear programming and its duality. The authors proved that in general the problems of testing the balancedness and testing the membership of the core are all NP-hard for both total dominating set games [123]. Fang et. al also discussed the integer domination games [124]. The authors introduced two k-domination games and focused on their cores. The authors gave characterizations of the cores and the relationship between two k-domination games by making use of the technique of linear programming and its duality.

Due to the new technology of web 2.0, social networks are a rapidly growing research area for information system scholars. Among much exploiting research, the relationships and influences among individuals in social networks might offer considerable benefits to both the economy and society. Domingos and Richardson [125] were the first ones to study the propagation of influence and the problem of identification of the most influential users in networks. Kempe et al. [53, 126] formulated the influence maximization problem as an optimization problem. Leskovec et al. [127] studied the influence propagation in a different perspective in which they aimed to find a set of nodes in networks to detect the spread of virus as soon as possible. Consequently, finding a proper subset of most influential individual is formulated into a domination problem in which an individual in the network becomes “influenced”. For example, Eubank et al. [36] proposed a greedy approximation algorithm and proved that the algorithm gives a $1 + O(1)$ approximation with a small constant in O(1) to the dominating set problem in a power-law graph.

Zhu et al. [128] studied a new type of dominating set (DS) which satisfies the property that for every node not in the DS has at least half of its neighbours which are in the dominating set. They presented results regarding the complexity and approximation in general graphs. Wang et al. [27] introduced a variation of a DS, called positive influence dominating set (PIDS), which originated from the context of influence propagation in social networks. Recently, Wang et al. [28] also proved that finding a PIDS of minimum size is APX-hard and proposed a greedy algorithm with an approximation ratio of $H(\delta)$ where H is the harmonic function and δ is the maximum vertex degree of the graph representing a social network. Dinh et al. [129] provided tight hardness results and approximation algorithms for many existing domination problems, especially the PIDS problem and its variations.

In this chapter, we consider the PIDS problem from the education/intervention cost point of view. We introduce two PIDS game models and prove the balancedness of the relaxed PIDS game using linear programming and its duality theory.

7.3 Motivations and Contributions

In order to deal with some social problems, such as drinking, smoking and drug use related issues, Wang et al. introduced the positive influence dominating set (PIDS) [27,28]. An illustration of the PIDS problem is the following example. For the drinking example remarked on earlier, a PIDS can guarantee that by selecting PIDS nodes to participate in the intervention programme, each individual in the social network has more positive neighbours than negative ones to ensure that the intervention can result in a globally positive impact on the entire social network.

There must be cost fees in an education/intervention programme in order to ensure that the intervention can result in a globally positive impact on the entire social network. Therefore, a natural question arising from the above example is how to allocate the total cost of educating or intervening the persons among the education/intervention programme. This is a cost allocation problem. In this paper, we use Cooperative Game Theory to study this problem and give a ”fair” distribution of the total cost. We first introduce two closely related cooperative cost games to model the cost allocation problem. To the best of our knowledge, this is the first work using Cooperative Game Theory to consider the cost of the education/intervention programme. The contributions are as follows:

1. Two new game models, the rigid PIDS game and relaxed PIDS game are presented.

2. A relationship between the cores of both games is obtained. The core of the relaxed PIDS game is non-empty if and only if there is no integrality gap for the relaxation linear programming of the PIDS problem on the graph.

7.4 Definition of Positive Influence Dominating Set Games

In this section, we introduce two cooperative cost games that are modeled on the cost allocation problem arising from the PIDS problems on social network graphs. We begin with some concepts and notions in Cooperative Game Theory.

7.4.1 Cooperative Game

A cooperative game (in characteristic function form) $\Gamma = (V, c)$ consists of a player set $V = \{1, 2, \ldots, n\}$ and a characteristic function $c : 2^V \to R$ with $c(\emptyset) = 0$. For each coalition $S \subseteq V$, $c(S)$ represents the revenue or cost achieved by the players in S together. The main issue is how to fairly distribute the total revenue or cost $c(V)$ among all the players. We define terms only for cost games, as a symmetric statement holds for revenue games.

A vector $x = \{x_1, x_2, \ldots, x_n\}$ is called an *imputation* if and only if $\sum_{i \in V} x_i = c(V)$ and $\forall i \in V : x_i \leq c(i)$. $x(S)$ is defined to be $\sum_{i \in S} x_i$ for each $S \subset V$. Now, the core of a game $\Gamma = (N, c)$ is defined as follows:

Definition 7.1. *An* imputation $x = \{x_1, x_2, \ldots, x_n\}$ *is said to lie in the core of the game* $\Gamma = (N, c)$ *if it is efficient and is such that:*

$Core(\Gamma) = \{x \in R^n : x(N) = c(N)$ *and* $x(S) \leq c(S),\ \forall S \subseteq V\}$,

where $x(S) = \sum_{i \in S} x_i$ for $S \subseteq V$.

The set of constraints imposed on $Core(\Gamma)$, which is called group rationality, ensures that no coalition would have an incentive to split from the grand coalition V , and do better on its own.

The study of the core is closely associated with another important concept, the balanced set. With techniques essentially the same as linear programming duality, Bondareva [114] and Shapley [115] proved that a game has non-empty core *if and only if it is balanced.*

Definition 7.2. *A game* $\Gamma = (V, c)$ *is called a* ***monotonic*** *game if it satisfies* $c(S) \leq c(T)$ *for every* $S \subseteq T \subseteq V$.

Proposition 7.1. *Given a balanced monotonic game* $\Gamma = (V, c)$ *and* $x \in Core(\Gamma)$, *it holds that* $x_i = c(V) - \sum_{j \in V \setminus i} x_j \geq c(V) - c(V \setminus i) \geq 0$ *for every* $i \in V$.

That is, each core element of a monotonic balanced game is non-negative.

7.5 Positive Influence Dominating Set Games

In this section, we present two PIDS cooperative cost games. Let $G = (V, E)$ be a connected undirected graph with vertex set V and edge set E. Two distinct vertices $u, v \in V$ are called *adjacent* if edge $(u, v) \in E$. For any non-empty set $V' \in V$, the induced subgraph by V', denoted by $G[V']$, is a subgraph of G whose vertex set is V' and whose edge set is the set of edges having both endpoints in V'. The *open neighbourhood* $N(v)$ of vertex $v \in V$ consists of the vertices adjacent to v, i.e., $N(v) = \{u \in V : (u, v) \in E\}$. $d(i)$ represents the number of neighbours of node i, or its degree ($d(i) = |N(i)|$). For any subset $S \subseteq V$, we define the *open neighbouring set* of S to be the union of the open neighbourhoods of all vertices in S, denoted by $N(S) = \bigcup_{v \in S} N(v)$.

A *positive influence dominating set* of graph G is a set of vertices $D \subseteq V$ such that $|N(v) \cap D| \geq \lceil \frac{d(v)}{2} \rceil$ for each $v \in V$. That is, every vertex in V is adjacent to at least half of its neighbours in D. The PIDS problem is to find a so-called minimum PIDS of G, which minimizes the total number of its vertices. The number of a minimum PIDS is called the *PIDS number*, denoted by $\gamma_p(G)$. Throughout the paper, we assume that graph G has no isolated vertex to ensure the existence of a PIDS of G.

Definition 7.3. *Let $G = (V, E)$ be a connected simple undirected graph. A* ***rigid PIDS game*** $\Gamma = (V, c)$ *corresponding to G is defined as:*

1) The player set is $V = \{1, 2, \ldots, n\}$;
2) For each coalition $S \subset V$,

$$c(S) = \begin{cases} (\gamma_p(G[S]) & \textit{if } G[S] \textit{ has a PIDS} \\ +\infty & \textit{otherwise.} \end{cases} \tag{7.1}$$

In the rigid PIDS game, each coalition can not place rescue facilities in vertices not belonging to itself. We define another related PIDS game, the relaxed PIDS game, by dropping the requirement that coalitions are only allowed to use vertices corresponding to members of the coalition. Formally, the *relaxed PIDS game* $\tilde{\Gamma} = (V, \tilde{c})$ corresponding to G is defined as:

Definition 7.4. *Let $G = (V, E)$ be a connected simple undirected graph. A* ***relaxed PIDS game*** $\Gamma = (V, c)$ *corresponding to G is defined as:*

1) The player set is $V = \{1, 2, \ldots, n\}$;
2) For each coalition $S \subset V$,

$\tilde{c}(S) = min\{\gamma_p(G[T]) : T \supseteq S \text{ and } G[T] \text{ has a } PIDS\}$.

Since coalitions have more choice of placing the facilities in the relaxed PIDS game than in the rigid PIDS game, for all $S \subset V$, it holds that $c(S) \geq \tilde{c}(S)$. For the grand coalition V, $c(V) = \tilde{c}(V) = \gamma_p(G)$. It follows that

$$Core(\tilde{\Gamma}) \subseteq Core(\Gamma). \tag{7.2}$$

Moreover, we show that $Core(\tilde{\Gamma})$ coincides with the nonnegative part of $Core(\Gamma)$.

Theorem 7.1. *Let $\Gamma = (V, c)$ and $\tilde{\Gamma} = (V, \tilde{c})$ be the rigid and relaxed PIDS games corresponding to graph $G = (V, E)$, respectively. Then we have*

$$Core(\tilde{\Gamma}) = Core(\Gamma) \bigcap R^n_+.$$

Proof: It is easy to see that the relaxed PIDS game $\tilde{\Gamma} = (V, \tilde{c})$ is monotonic, i.e., $\tilde{c}(S) \leq \tilde{c}(T)$ for every $S \subseteq T$. For each $x \in Core(\tilde{\Gamma})$, we have that $x_i = \tilde{c}(V) - \sum_{j \in V \backslash i} x_j \geq \tilde{c}(V) - \tilde{c}(V \backslash i) \geq 0$ for every $i \in V$. Also followed from expression (7.2), it holds that $Core(\tilde{\Gamma}) \subseteq Core(\Gamma) \bigcap R^n_+$.

On the other hand, we prove that $Core(\Gamma) \bigcap R^n_+ \subseteq core(\tilde{\Gamma})$. Let $x \in Core(\Gamma) \bigcap R^n_+$. Clearly, $x(V) = c(V) = \tilde{c}(V)$. Let $T \subset V$ be an arbitrary subset such that $c(T) \geq \tilde{c}(T)$, and let $T_0 \subseteq V$ be a subset such that $T \subseteq N(T_0)$ and T_0 be the minimum PIDS of T. So we have $\tilde{c}(T) = min\{\gamma_p(G[T])\} = |T_0|$. It follows that $c(N(T_0)) = |T_0| = \tilde{c}(T)$. Hence we have

$$x(T) \leq x(N(T_0)) \leq c(N(T_0)) = \tilde{c}(T),$$

where the first inequality holds because $x \geq 0$ and the second inequality holds because $x \in Core(\Gamma)$. Therefore, $x \in Core(\tilde{\Gamma})$. ■

7.6 The Balancedness of the PIDS Game

In this section, we first provide descriptions of the cores for the rigid and relaxed PIDS games. Based on these descriptions, we prove a common necessary and sufficient condition for the balancedness of the relaxed PIDS games.

7.6.1 Balancedness of the relaxed PIDS game

For a graph $G = (V, E)$, the adjacent matrix of G, denoted by $A(G) = [a_{ij}]$, is a $|V| \times |V|$-matrix with rows and columns indexed by the vertices in V respectively, where $a_{ij} = 1$ if vertex i and j are adjacent, and $a_{ij} = 0$ otherwise. Then the PIDS problem can be formulated as the following $0-1$ integer linear programming (IP):

$$\gamma_t(G) = min \sum_{j=1}^{n} x_j \tag{7.3}$$

$$s.t. \begin{cases} A(G)x \geq b \\ x = (x_1, x_2, \ldots, x_n)^T \in \{0,1\}^n, \end{cases}$$

where $b = \{\lceil \frac{d(1)}{2} \rceil, \lceil \frac{d(2)}{2} \rceil, \ldots, \lceil \frac{d(n)}{2} \rceil\}^T$.

For this integer programming, we have the linear programming relaxation (LP) as follows:

$$\gamma_t(G) = min \sum_{j=1}^{n} x_j \tag{7.4}$$

$$s.t. \begin{cases} A(G)x \geq b \\ x = (x_1, x_2, \ldots, x_n)^T \geq 0, \end{cases}$$

and its dual (DLP):

$$max \sum_{i=1}^{n} yb \tag{7.5}$$

$$s.t. \begin{cases} yA(G) \leq 1 \\ y = (y_1, y_2, \ldots, y_n) \geq 0, \end{cases}$$

where $b = \{\lceil \frac{d(1)}{2} \rceil, \lceil \frac{d(2)}{2} \rceil, \ldots, \lceil \frac{d(n)}{2} \rceil\}^T$.

In the following, we show that the relaxed game $\tilde{\Gamma} = (V, \tilde{c})$ is balanced if and only if the above linear programming relaxation LP (7.4) has an integral optimal solution, i.e., the optimal value of LP (7.4) equals the minimum PIDS of the graph G. The following theorem provides a characterization of the core and a necessary and sufficient condition for the balancedness of the relaxed PIDS game $\tilde{\Gamma}$.

Theorem 7.2. *Let $\tilde{\Gamma} = (V, \tilde{c})$ be a relaxed PIDS game corresponding to a graph $G = (V, E)$. Then $Core(\tilde{\Gamma}) \neq \emptyset$ if and only if there is no integrality gap for the relaxation LP (7.4) of the PIDS problem on the graph G. In such case, a vector $z = \{z_1, z_2, \ldots, z_n\}$ is in the core if and only if it is an optimal solution to the dual program DLP (7.5).*

Proof: Considering the IP (7.3) relaxation LP (7.4) and its dual DLP (7.5) of LP (7.4), the relaxed PIDS game $\tilde{\Gamma} = (V, \tilde{c})$ belongs to the class of covering games introduced in Deng et al. [120]. For the proof of this theorem we refer to the Theorem 1 in the paper of Deng et al. [120].

7.6.2 Computational Complexity on Cores

The computational complexity as a rational measure for game theoretical concepts has attracted more and more attention recently. Deng and Papadimitriou [111] found a game for which the core is non-empty if and only if a certain imputation (Shapley value in this case) is in the core. Deng et al. [120] discussed the complexity concerning the core for a class of combinatorial optimization games. Goemans and Skutella [116] showed that, for a facility location game, if the core is non-empty, a core element can be found in polynomial time, and the membership testing problem can also be solved in polynomial time. However, it is NP-complete to decide whether the core is non-empty. Fang et. al [123] recently proved that it is NP-complete to decide whether the core of a total domination game is non-empty.

The computational complexity issues concerning the cores of the PIDS games will be the focus of this subsection. From Theorem 7.2, the relaxed PIDS game is balanced when the linear programming (LP (7.4)) has an integer optimal solution, and we can obtain a core element for the relaxed PIDS games from an optimal solution to its dual programming given in DLP (7.5). That is, a core element can be found in polynomial time when the core is non-empty. However, the problem of testing whether the linear programming (LP (7.4)) has an integer optimal solution is difficult in general. For the dominating set problem, it was shown to be NP-hard to determine the minimum number of the dominating set (see [37]). We conjecture that for the PIDS problem, the problem of testing whether the linear programming (LP) given in (7.4) has an integer optimal solution or not is NP-hard. That is, testing the balancedness for both PIDS games is NP-hard solution.

7.7 Summary

In this chapter, we are interested in cooperative cost games arising from the PIDS problem on social network graphs. We have presented two new cooperative cost game models, the rigid PIDS game and relaxed PIDS game. We discuss their cores and a relationship between the cores of two PIDS games is obtained. We also prove that the core of the relaxed PIDS game is non-empty if and only if there is no integrality gap for its relaxation linear programming.

We briefly discuss how the ideas can be further studied. Since we have the relationship of the two PIDS games cores from Theorem 7.1, we can further discuss the balancedness of the rigid PIDS game. We conjecture that in general the problems of testing the balancedness and testing the membership of the core are all NP-hard for both PIDS games.

Chapter 8

Conclusions

This final chapter presents the conclusions and future research arising from this work. The overall aim of this research was to develop new models and algorithms for the dominating set problems arising from online social networks. An overview of the previous chapters is initially provided to recap the significant trends in this PhD research before highlighting the conclusions about the research problems addressed. The key outcomes of the research and final opportunities for further research then presented according to the research problems identified in Chapter 1.

8.1 Overview of Previous Chapters

Chapter 1

A broad overview and background of the research theme and the domination problems in general is provided in this chapter. Especially, due to the rapid development of online social networks, a new type of dominating set called positive influence dominating set (PIDS) has been studied in the literature to deal with some social problems. It is shown that the PIDS can be applied only in undirected social networks with uniform weight. We have the motivations to address theses shortcomings through designing new models and algorithms. The research problems are outlined in this chapter. The potential contributions and significance of this research are also identified.

Chapter 2

The major objective of this chapter is to extend the PIDS model which was introduced by Wang et al. [27,28] since it has some drawbacks in real-life applications. We have designed a novel dominating set called weighted positive influence dominating set (WPIDS) and two WPIDS selection algorithms for the WPIDS problem in the e-learning environment, considering some key factors such as the attributes, directions and degrees of personal influence in the environment. Experimental results demonstrate that our WPIDS model and algorithms are more reasonable and effective than those of the PIDS which can deal with social problems only for undirected social networks with uniform weight [27].

Chapter 3

For the minimal k-dominating set (MKDS), the proposed self-stabilizing algorithms for the MKDS either work for trees (Kamei and Kakugawa [26]) or find a minimal 2-dominating set (Huang et al. [55, 68]). There is no self-stabilizing algorithm for the MKDS works for any general network graph. Hence, we have proposed a self-stabilizing algorithms for the MKDS when operating in any general graph in this chapter. The presented algorithm works under a central daemon model. We have also proved that the worst case convergence time of the algorithm from any arbitrary initial state is $O(n^2)$ steps where n is the number of nodes in the network.

Chapter 4

In this chapter, we continue discussing the MKDS problem we have studied in Chapter 3. We have proposed a self-stabilizing algorithm for the MKDS under a distributed daemon model when operating in any general network. We have further proved that the worst case convergence time of the algorithm from any arbitrary initial state is $O(n^2)$ steps where n is the number of nodes in the network.

Chapter 5

For the minimal weighted dominating set (MWDS), the proposed self-stabilizing algorithms for the MWDS all work for uniform weight cases. We have developed a self-stabilizing algorithm for finding an MWDS under a central daemon when it operates in a general graph and proved that the worst case convergence time of the algorithm from any arbitrary initial state is $O(n^2)$ steps where n is the number of nodes in the network.

Chapter 6

For the PIDS, the existing work all focused on greedy algorithms for the PIDS problem with different approximation ratios. In this chapter, we investigate minimal solution for the PIDS instead of greedy algorithms for approximation solutions. We have first presented a self-stabilizing algorithm for the minimal PIDS problem. It can work for any general graphs under a central daemon. We have also proved that the worst case convergence time of the algorithm from any arbitrary initial state is $O(n^2)$ steps where n is the number of nodes in the network.

Chapter 7

In Chapter 7, we investigate the PIDS problem from the education/intervention cost point of view. That is, how to allocate the total cost of educating or intervening persons among an education/intervention programme. We have introduced two cooperative cost games, the rigid PIDS game and relaxed PIDS game and focused on their cores (a “fair” distribution of the total cost among all the players). A relationship between the cores of both games is obtained. We have proved that the core of the relaxed PIDS game is non-empty if and only if there is no integrality gap for the relaxation linear programming of the PIDS problem on the graph.

8.2 Key Outcomes

This thesis focuses on domination problems arising from social networks. The outcomes of our study are shown below:

- Propose a new dominating set model and two WPIDS selection algorithms to evaluate the effect of educating a subset of the entire target group susceptible to a social problem. The simulation experimental results have revealed that the WPIDS model and selection algorithm are more effective than those of PIDS [27, 28]. The main reason is that we consider the tutors who play important roles in the e-learning community. So the size of WPIDS is smaller than that of PIDS in an online social network [27, 28].
- Propose two self-stabilizing algorithms for the MKDS in general graphs and discuss its computational complexity respectively.
- Propose a self-stabilizing algorithm for the MWDS in general graphs and discuss its computational complexity. .
- Propose a self-stabilizing algorithm for the MPIDS in general graphs and discuss its computational complexity. .
- Propose two cooperative cost game models, the rigid PIDS game and relaxed PIDS game. Discuss their cores and obtain a relationship between the cores of two PIDS games.

8.3 Future Research

This thesis is mainly focusing on algorithms of the domination problems on social networks. The proposed research aims to address certain novel dominations such as weighted positive influence domination, weighted domination, k domination and so on. Based on the research work in this thesis, we propose the following future research directions and issues:

- We have proposed a new dominating set model and two WPIDS selection algorithms to evaluate the effect of educating a subset of the entire target group susceptible to a social problem. To deeply understand the effect of WPIDS, comparing the WPIDS selection algorithm with the PIDS algorithm in some real-life e-learning communities is interesting in future work. Since it is very important to specify the reasonable arc weight values of the e-learning users' influence, the proper selection of parameter θ, and the cost of each tutor, designing the WPIDS models under these factors are challenged.
- In this thesis, we have successfully solved the self-stabilizing algorithms for the MKDS, the MWDS and the MPIDS. To discuss the much faster stabilization time of these proposed algorithms is interesting and challenging. An immediate extension of the work for the MKDS (MWDS or MPIDS) is to find if it is possible to enhance the stabilization time to $O(nlgn)$ steps.

Another future research topic for the MKDS (MWDS or MPIDS) is to attempt to find a proper upper bound size of an MKDS (MWDS or MPIDS) in an arbitrary network. If someone can develop some self-stabilizing algorithms for the MKDS (MWDS or MPIDS) under a synchronous daemon in an arbitrary network, it is an interesting work.

- For the cost of the PIDS problem, we have presented two new cooperative cost game models, the rigid PIDS game and the relaxed PIDS game. Since we have the relationship between the cores of two PIDS games from Theorem 7.1, discussing the balancedness of the rigid PIDS game is a future work. We conjecture that in general the problems of testing the balancedness and testing the membership of the core are all NP-hard for both PIDS games.
- For the WPIDS problem, if considering the cost issue in an intervention programme by Cooperative Game Theory, some good outcomes or solutions will be achieved.

Although several algorithms on dominating sets have been proposed, the work on domination is far from complete. We will have been planned experiment work for testing the performance of the proposed algorithm as an extension work in near future.

Data Collection

To perform necessary experiments in the proposed project, certain relational databases are necessary to conduct experiments. There are three ways to obtain experimental data. Firstly, we can use crawl softwares (Google search) to crawl needed data from famous websites, for instances, Facebook.com, Flickr.com, Myspace.com, YouTube.com, Yahoo.com, and so on. Secondly, masses of famous public datasets, such as LETOR, Libra, DBLB, etc, can be used by many researchers for application software development. Finally, simulated data for experiment is a common method as we have studied the WPIDS problem in one of my published papers [70].

Performance Analysis.

In general, one of the performance measures of domination problems is the time complexity. Although we have proved the computing complexity of the proposed algorithms, it is very important to consider the running time of the proposed algorithms, especially working on a large social network as the examiners' suggestions'. To analyze experiment results, techniques such as charts, tables, and figures can be better used so that the experiment results may be analyzed with better efficiency. By comparing with existing methods and algorithms the efficiency of our proposed methods and algorithms can be test. In summary, several ordinary performance metrics for evaluation purposes can be used,including the running time of these algorithms, the size of the dominating set, and the sensitivity of presented approaches.

References

[1] C. Berge, *Theory of Graphs and its Applications.* Methuen, London, 1962.

[2] O. Ore, *Theory of Graphs.* Amer. Math. Soc. Colloq. Publ., 38, 1962.

[3] E. J. Cockayne and S. T. Hedetniemi, "Towards a theory of domination in graphs," *Networks*, vol. 7, pp. 247–261, 1977.

[4] T. W. Haynes, S. T. Hedetniemi, and P. J. S. (Eds.), *Fundamentals of domination in graphs.* Marcel Dekker Inc., New York, 1998, vol. 208 of Monographs and Textbooks in Pure and Applied Mathematics.

[5] J. M. M. van Rooij, *Exact Exponential-Time Algorithms for Domination Problems in Graphs.* PhD thesis, Utrecht University, 2011.

[6] O. Favaron, M. Henning, C. Mynhart, and J. Puech, "Total domination in graphs with minimum degree three," *J. Graph Theory*, vol. 34, pp. 9–19, 2000.

[7] F. V. Fomin and D. M. Thilikos, "Dominating sets in planar graphs: branch-width and exponential speed-up," in *SODA*, 2003, pp. 168–177.

[8] J. M. M. van Rooij and H. L. Bodlaender, "Exact algorithms for edge domination," in *IWPEC*, 2008, pp. 214–225.

[9] C. Lenzen, Y. A. Oswald, and R. Wattenhofer, "What can be approximated locally?: case study: dominating sets in planar graphs," in *SPAA*, 2008, pp. 46–54.

[10] J. Wu and H. Li, "On calculating connected dominating set for efficient routing in ad hoc wireless networks," in *In: Proceedings of the 3rd ACM InternationalWorkshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 1999, pp. 7–14.

[11] E. J. Cockayne, R. M. Dawes, and S. T. Hedetniemi, "Total domination in graphs," *Networks*, vol. 10, pp. 211–219, 1980.

[12] M. A. Henning, "A survey of selected recent results on total domination in graphs," *Discrete Mathematics*, vol. 309, no. 1, pp. 32–63, 2009.

[13] T. W. Haynes, M. A. Henning, and L. C. van der Merwe, "Total domination supercritical graphs with respect to relative complements," *Discrete Mathematics*, vol. 258, no. 1-3, pp. 361–371, 2002.

[14] M. A. Henning and H. C. Swart, “Bounds on a generalized total domination parameter,” *J. Combin. Math. Combin.Comput*, vol. 6, pp. 143–153, 1989.

[15] J. A. Torkestani and M. R. Meybodi, “Finding minimum weight connected dominating set in stochastic graph based on learning automata,” *Information Sciences*, vol. 200, pp. 57–77, 2012.

[16] F. Zou, Y. Wang, X. Xu, X. Li, H. Du, P. Wan, and W. Wu, “New approximations for minimum-weighted dominating sets and minimum-weighted connected dominating sets on unit disk graphs,” *Theor. Comput. Sci.*, vol. 412, no. 3, pp. 198–208, 2011.

[17] X. Zhu, W. Wang, S. Shan, Z. Wang, and W. Wu, “A ptas for the minimum weighted dominating set problem with smooth weights on unit disk graphs,” *J. Comb. Optim.*, vol. 23, no. 4, pp. 443–450, 2012.

[18] A. Potluri and A. Singh, “Hybrid metaheuristic algorithms for minimum weight dominating set,” *Appl. Soft Comput.*, vol. 13, no. 1, pp. 76–88, 2013.

[19] E. Sampathkumar and H. B. Walikar, “The connected domination number of a graph,” *Math. Phys. Sci*, vol. 13, pp. 607–613, 1979.

[20] G. S. Khuller, “Approximation algorithms for connected dominating sets,” *Algorithmica*, vol. 20, pp. 374–387, 1998.

[21] X. Cheng, F. Wang, and D. Du, “Connected dominating set,” in *Encyclopedia of Algorithms*, 2008.

[22] M. T. Thai, F. Wang, D. Liu, S. Zhu, and D. Du, “Connected dominating sets in wireless networks with different transmission ranges,” *IEEE Trans. Mob. Comput.*, vol. 6, no. 7, pp. 721–730, 2007.

[23] D. Kim, Z. Zhang, X. Li, W. Wang, W. Wu, and D. Du, “A better approximation algorithm for computing connected dominating sets in unit ball graphs,” *IEEE Trans. Mob. Comput.*, vol. 9, no. 8, pp. 1108–1118, 2010.

[24] L. Ding, X. Gao, W. Wu, W. Lee, X. Zhu, and D. Du, “An exact algorithm for minimum cds with shortest path constraint in wireless networks,” *Optimization Letters*, vol. 5, no. 2, pp. 297–306, 2011.

[25] M. Gairing, S. T. Hedetniemi, P. Kristiansen, and A. A. McRae, “Self-stabilizing algorithms for {k}-domination,” in *Self-Stabilizing Systems*, 2003, pp. 49–60.

[26] S. Kamei and H. Kakugawa, “A self-stabilizing algorithm for the distributed minimal k-redundant dominating set problem in tree network,” in *ICPDC*, 2003.

[27] F. Wang, E. Camacho, and K. Xu, “Positive influence dominating set in online social networks,” in *COCOA*, 2009, pp. 313–321.

[28] F. Wang, H. Du, E. Camacho, K. Xu, W. Lee, Y. Shi, and S. Shan, “On positive influence dominating sets in social networks,” *Theor. Comput. Sci.*, vol. 412, no. 3, 2011.

[29] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *The 7th ACM SIGCOMM Conference on Internet Measurement Conference(IMC)*, 2007, pp. 29–42.

[30] A. Anagnostopoulos, R. Kumar, and M. Mahdian, "Influence and correlation in social networks," in *the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, 2008, pp. 7–15.

[31] A. Nazir, S. Raza, and C. N. Chuah, "Unveiling facebook: a measurement study of social network based applications," in *The 8th ACM SIGCOMM Internet Measurement Conference (IMC)*, 2008, pp. 43–56.

[32] C. Yang, R. Harkreader, J. Zhang, S. Shin, and G. Gu, "Analyzing spammers' social networks for fun and profit: a case study of cyber criminal ecosystem on twitter," in *WWW*, 2012, pp. 71–80.

[33] M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi, "Measuring user influence in twitter: the million follower fallacy," in *In Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media*, 2010.

[34] H. Kwak, C. Lee, H. Park, and S. B. Moon, "What is twitter, a social network or a news media?" in *WWW*, 2012, pp. 591–600.

[35] D. Kempe, J. Keinberg, and É. Tardos, "Maximizing the spread of infuence through a social network," in *The 9th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD).*, 2003, pp. 137–146.

[36] S. Eubank, V. Kumar, M. V. Marathe, A. Srinivasan, and N. Wang, "Structural and algorithmic aspects of massive social networks," in *SODA*, 2004, pp. 718–727.

[37] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[38] H. L. Bodlaender and J. M. M. van Rooij, "Exact algorithms for intervalizing colored graphs," in *TAPAS*, 2011, pp. 45–56.

[39] F. Fomin, F. Grandoni, and D. Kratsch, "Some new techniques in design and analysis of exact (exponential) algorithms," *Bulletin of the EATCS*, vol. 87, pp. 47–77, 2005.

[40] F. V. Fomin, D. Kratsch, and G. J. Woeginger, "Exact (exponential) algorithms for the dominating set problem," in *WG*, 2004, pp. 245–256.

[41] J. M. M. van Rooij and H. L. Bodlaender, "Design by measure and conquer, a faster exact algorithm for dominating set," in *STACS*, 2008, pp. 657–668.

[42] F. V. Fomin, F. Grandoni, and D. Kratsch, "A measure & conquer approach for the analysis of exact algorithms," *J. ACM*, vol. 56, no. 5, 2009.

[43] L. Ding, X. Gao, W. Wu, W. J. Lee, X. Zhu, and D. Du, "An exact algorithm for minimum cds with shortest path constraint in wireless networks," *Optim Lett.*, vol. 5, pp. 297–306, 2011.

[44] J. M. M. van Rooij and H. L. Bodlaender, “Exact algorithms for dominating set,” *Discrete Applied Mathematics*, vol. 159, no. 17, pp. 2147–2164, 2011.

[45] L. B. N. Alon and A. Itai, “A fast and simple randomized parallel algorithm for the maximal independent set problem,” *J. Algorithms*, vol. 7, no. 4, pp. 567–583, 1986.

[46] N. Linial, “Locality in distributed graph algorithms,” *SIAM J. Comput.*, vol. 21, no. 1, pp. 193–201, 1992.

[47] M. Naor and L. J. Stockmeyer, “What can be computed locally?” *SIAM J. Comput.*, vol. 24, no. 6, pp. 1259–1277, 1995.

[48] D. Peleg, *Distributed computing: a locality-sensitive approach.* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

[49] M. Cardei, M. Cheng, X. Cheng, and D. Du, “Connected domination in multihop ad hoc wireless networks,” in *JCIS*, 2002, pp. 251–255.

[50] X. Cheng, X. Huang, D. Li, W. Wu, and D. Z. Du, “A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks,” *Networks*, vol. 42, no. 4, pp. 202–208, 2003.

[51] Y. Li, M. T. Thai, F. Wang, C. Yi, P. J. Wan, and D. Du, “On greedy construction of connected dominating sets in wireless networks,” *Wireless Communications and Mobile Computing*, vol. 5, pp. 927–932, 2005.

[52] A. Barabasi, *Emergence of scaling in complex networks*, in: s. bornholdt, h. schuster (eds.), handbook of graphs and networks ed. Wiley, 2003.

[53] D. Kempe, J. M. Kleinberg, and É. Tardos, “Influential nodes in a diffusion model for social networks,” in *ICALP*, 2005, pp. 1127–1138.

[54] S. M. Hedetniemi, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani, “Self-stabilizing algorithms for minimal dominating sets and maximal independent sets,” *Computer Mathematics and Applications*, vol. 46, no. 5-6, pp. 805–811, 2003.

[55] T. Huang, C. Chen, and C. Wang, “A linear-time self-stabilizing algorithm for the minimal 2-dominating set problem in general networks,” *J. Inf. Sci. Eng.*, vol. 24, no. 1, pp. 175–187, 2008.

[56] E. W. Dijkstra, “Self-stabilizing systems in spite of distributed control,” *Commun. ACM*, vol. 17, no. 11, pp. 643–644, 1974.

[57] E. W. Dijkstra and A. J. M. van Gasteren, “A simple fixpoint argument without the restriction to continuity,” *Acta Inf.*, vol. 23, no. 1, pp. 1–7, 1986.

[58] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani, “A self-stabilizing distributed algorithm for minimal total domination in an arbitrary system graph,” in *IPDPS*, 2003, p. 240.

[59] V. Turau, “Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler,” *Information Processing Letters*, vol. 103, no. 3, pp. 88–93, 2007.

[60] M. Ikeda, S. Kamei, and H. Kakugawa, “A space-optimal self-stabilizing algorithm for the maximal independent set problem,” in *Proc. 3rd International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2002, p. 0.

[61] S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani, “Linear time self-stabilizing colorings,” *Information Processing Letters*, vol. 87, no. 5, pp. 251–255, 2003.

[62] A. Kosowski and L. Kuszner, “Self-stabilizing algorithms for graph coloring with improved performance guarantees,” in *Proc. 8th International Conference on Artificial Intelligence and Soft Computing*, 2006, pp. 1150–1159.

[63] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani, “Fault tolerant algorithms for orderings and colorings,” in *Proc. 18th International Parallel and Distributed Processing Symposium*, 2004, pp. 174–182.

[64] F. Manne, M. Mjelde, L. Pilard, and S. Tixeuil, “A new self-stabilizing maximal matching algorithm,” *CoRR*, vol. abs/cs/0701189, 2007.

[65] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani, “Self-stabilizing distributed algorithm for strong matching in a system graph,” in *Proc. 10th International Conference on High Performance Computing*, 2003, pp. 66–73.

[66] F. Manne, M. Mjelde, L. Pilard, and S. Tixeuil, “A self-stabilizing $\frac{2}{3}$-approximation algorithm for the maximum matching problem,” in *Proc. 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 2008, pp. 94–108.

[67] S. Dolev, *Self-Stabilization.* MIT Press, 2000.

[68] T. Huang, J. Lin, C. Chen, and C. Wang, “A self-stabilizing algorithm for finding a minimal 2-dominating set assuming the distributed demon model,” *Computers and Mathematics with Applications*, vol. 54, no. 3, pp. 350–356, 2007.

[69] G. Wang, H. Wang, X. Tao, and J. Zhang, “Positive influence dominating set in e-learning social networks,” in *In proceedings of the 10th International Conference on Web-based Learning (ICWL 2011)*, 2011, pp. 82–91.

[70] G. Wang, H. Wang, X. Tao, J. Zhang, and G. Guo, “Finding a weighted positive influence dominating set in e-learning social networks,” *Internal Journal Computing & Technology*, vol. 10, no. 10, pp. 2136–2145, 2013.

[71] G. Wang, H. Wang, X. Tao, and J. Zhang, “A self-stabilizing algorithm for finding a minimal k-dominating set in general networks,” in *In proceedings of 2012 International Conference on Data and Knowledge Engineering (ICDKE 2012)*, 2012, pp. 74–85.

[72] G. Wang, H. Wang, X. Tao, J. Zhang, and J. Zhang, “Minimising k-dominating set in arbitrary network graphs,” in *In proceedings of the 9th International Conference on Advanced Data Mining and Applications (ADMA 2013)*, 2013.

[73] G. Wang, H. Wang, X. Tao, and J. Zhang, "A self-stabilizing algorithm for finding a minimal positive influence dominating set in social networks," in *In proceedings of 24th Australasian Database Conference (ADC 2013)*, 2013, pp. 93–99.

[74] G. Wang, H. Wang, X. Tao, J. Zhang, and X. Yi, "Positive influence dominating set games," in *To appear in the 18th IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD 2014)*, 2014.

[75] G. Wang, H. Wang, X. Tao, and J. Zhang, "A self-stabilizing protocol for minimal weighted dominating sets in arbitrary networks," in *In proceedings of the 17th IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD 2013)*, 2013, pp. 496–501.

[76] I. Giannoukos, I. Lykourentzou, G. Mpardis, V. Nikolopoulos, V. Loumos, and E. Kayafas, "Collaborative e-learning environments enhanced by wiki technologies," in *PETRA*, 2008, p. 59.

[77] I. H. Hsiao, J. Guerra, D. Parra, F. Bakalov, B. König-Ries, and P. Brusilovsky, "Comparative social visualization for personalized e-learning," in *AVI*, 2012, pp. 303–307.

[78] H. Wang, Y. Zhang, and J. Cao, "Effective collaboration with information sharing in virtual universities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 6, pp. 840–853, 2009.

[79] N. Ghasem-Aghaee, S. Fatahi, and T. I. Ören, "Agents with personality and emotional filters for an e-learning environment," in *SpringSim*, 2008, p. 5.

[80] H. Shi, S. Revithis, and S. Chen, "An agent enabling personalized learning in e-learning environments," in *AAMAS*, 2002, pp. 847–848.

[81] D. J. Crandall, D. Cosley, D. P. Huttenlocher, J. M. Kleinberg, and S. Suri, "Feedback effects between similarity and social influence in online communities," in *KDD*, 2008, pp. 160–168.

[82] J. Jaccard, H. Blanton, and T. Dodge, "Peer influences on risk behavior: Analysis of the effects of a close friend," *Developmental Psychology*, vol. 41, no. 1, pp. 135–147, 2005.

[83] J. B. Standridge, R. G. Zylstra, and S. M. Adams, "Alcohol consumption: An overview of benefits and risks," *Southern Medical Journal*, vol. 97, no. 7, pp. 664–672, 2004.

[84] M. E. Larimer and J. M. Cronce, "Identification, prevention, and treatment revisited: Individual-focused college drinking prevention," *Addictive Behaviors*, vol. 32, pp. 2439–2468, 2007.

[85] S. T. Walters and M. E. Bennett, "Addressing drinking among college students: a review of the empirical literature," *Alcoholism Treatment Quarterly*, vol. 18, no. 1, pp. 61–67, 2000.

[86] W. Wei, J. H.-M. Lee, and I. King, "Measuring credibility of users in an e-learning environment," in *WWW*, 2007, pp. 1279–1280.

[87] L. Lamport, "Solved problems, unsolved problems and non-problems in concurrency," in *PODC 1984 Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, 1984, pp. 1–11, invited address.

[88] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, P. K. Srimani, and Z. Xu, "Self-stabilizing graph protocols," *Parallel Processing Letters*, vol. 18, no. 1, pp. 189–199, 2008.

[89] E. W. Dijkstra, "Self-stabilization in spite of distributed control," in *Selected Writings on Computing: A Personal Perspective, Springer-Verlag*, 1982, pp. 41–46.

[90] ——, "A belated proof of self-stabilization," *Distrib. Comput.*, vol. 1, pp. 5–6, 1986.

[91] J. Beauquier, A. K. Datta, M. Gradinariu, and F. Magniette, "Self-stabilizing local mutual exclusion and daemon refinement," in *DISC*, 2000, pp. 223–237.

[92] M. Nesterenko and A. Arora, "Stabilization-preserving atomicity refinement," in *DISC*, 1999, pp. 254–268.

[93] T. Herman, "Probabilistic self-stabilization," *Information Processing Letters*, vol. 35, no. 2, pp. 63–67, 1990.

[94] S. Kamei and H. Kakugawa, "A self-stabilizing approximation algorithm for the distributed minimum k-domination," *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences E88-A*, vol. 5, pp. 1109–1116, 2005.

[95] Z. Xu, S. T. Hedetniemi, W. Goddard, and P. K. Srimani, "A synchronous selfstabilizing minimal domination protocol in an arbitrary network graph," in *Proc. 5th International Workshop on Distributed Computing*, 2003, pp. 26–32.

[96] S. K. Shukla, D. J. Rosenkrantz, and S. S. Ravi, "Developing self-stabilizing coloring algorithms via systematic randomization," in *Proc. 1st International Workshop on Parallel Processing*, 1994, pp. 668–673.

[97] J. F. Fink and M. S. Jacobson, *Graph theory with applications to algorithms and computer science.* Inc., New York, NY, USA, 1985, pp. 283–300.

[98] M. S. Jacobson and K. Peters, "Complexity questions for n-domination and related parameters," *, Congr. Numer of the 18th Manitoba Conference on Numerical Mathematics and Computing*, vol. 68, pp. 7–22, 1989.

[99] N. Guellati and H. Kheddouci, "A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs," *J. Parallel Distrib. Comput.*, vol. 70, no. 4, pp. 406–415, 2010.

[100] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater., *Domination in Graphs: Advanced Topics.* Marcel Dekker, 1998.

[101] ——, *Fundamentals of Domination in Graphs.* Marcel Dekker, 1998.

[102] P. A. Kirschner and A. C. Karpinski, "Facebook and academic performance," *Computers in Human Behavior*, vol. 26, pp. 1237–1245, 2010.

[103] C. Madge, J. Meek, J. Wellens, and T. Hooley, "Facebook, social integration and informal learning at university: it is more for socialising and talking to friends about work than for actually doing work," *Computers in Human Behavior*, vol. 34, no. 2, pp. 141–155, 2009.

[104] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in facebook," in *WOSN*, 2009, pp. 37–42.

[105] L. Sun and H. Wang, "Access control and authorization for protecting disseminative information in e-learning workflow," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 16, pp. 2034–2041, 2011.

[106] H. Wang, J. Cao, and Y. Zhang, "A flexible payment scheme and its role based access control," *Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 425–436, 2005.

[107] X. Sun, H. Wang, J. Li, and J. Pei, "Publishing anonymous survey rating data," *Data Min. Knowl. Discov.*, vol. 23, no. 3, pp. 379–406, 2011.

[108] X. Sun, H. Wang, J. Li, and Y. Zhang, "Satisfying privacy requirements before data anonymization," *Comput. J.*, vol. 55, no. 4, pp. 422–437, 2012.

[109] X. Deng, T. Ibaraki, and H. Nagamochi, "Combinatorial optimization games," in *Proc. 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1997, pp. 720–729.

[110] X. Deng, T. Ibaraki, H. Nagamochi, and W. Zhang, "Totally balanced combinatorial optimization games," *Mathematical Programming*, vol. 87, pp. 441–452, 2000.

[111] X. Deng and C. Papadimitriou, "On the complexity of cooperative game solution concepts," *Mathematics of Operations Research*, vol. 19, pp. 257–266, 1994.

[112] I. Curiel, *Cooperative game theory and applications.* Kluwer Academic Publishers, 1997.

[113] L. S. Shapley and M. Shubik, "The assignment game," *International Journal of Game Theory*, vol. 1, pp. 111–130, 1972.

[114] O. N. Bondareva, "Some applications of the methods of linear programming to the theory of cooperative games," *(Russian) Problemy Kibernet*, vol. 10, pp. 119–139, 1963.

[115] L. S. Shapley, "On balanced sets and cores," *Naval Res. Quart.*, vol. 14, pp. 453–460, 1967.

[116] M. X. Goemans and M. Skutella, "Cooperative facility location games," in *SODA*, 2000, pp. 76–85.

[117] R. P. Gilles, *The Cooperative Game Theory of Networks and Hierarchies.* Springer-Verlag Berlin Heidelberg, 2010.

[118] I. Curiel., *Cooperative game theory and applications - Cooperative games arising from combinatorial optimization problems.* Kluwer Academic, 1997.

[119] G. Owen, "On the core of linear production games," *Mathematical Programming*, vol. 9, no. 3, pp. 358–370, 1975.

[120] X. Deng, T. Ibaraki, and H. Nagamochi, "Algorithmic aspects of the core of combinatorial optimization games," *Mathematics of Operations Research*, vol. 24, pp. 751–766, 1999.

[121] B. V. Velzen, "Dominating set game," *Opertations Research Letters*, vol. 32, pp. 565–573, 2004.

[122] Q. Fang and H. K. Kim, "A note on balancedness of dominating set games," *J. Comb. Optim.*, vol. 10, no. 4, pp. 303–310, 2005.

[123] Q. Fang, H. K. Kim, and D. S. Lee, "Total dominating set games," in *WINE*, 2005, pp. 520–530.

[124] H. K. Kim and Q. Fang, "Balancedness of integer domination games," *Journal of the Korean Mathematical Society*, vol. 43, pp. 297–309, 2006.

[125] P. Domingos and M. Richardson, "Mining the network value of customers," in *KDD*, 2001, pp. 57–66.

[126] D. Kempe, J. M. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *KDD*, 2003, pp. 137–146.

[127] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. M. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *KDD*, 2007, pp. 420–429.

[128] X. Zhu, J. Yu, W. J. Lee, D. Kim, S. Shan, and D. Z. Du, "New dominating sets in social networks," *J. Global Optimization*, vol. 48, no. 4, pp. 633–642, 2010.

[129] T. N. Dinh, M. T. Thai, and D. T. Nguyen, *A Unified Approach for Domination Problems on Different Network Topologies*, (p. pardalos, d. du, and r. graham eds) ed. Springer Publisher.

Appendices

Appendix A

Procedure Code

The following is the procedure code in the paper [70].

```
mainclass.java
/** * * @author Guohun Zhu
*/
package Dominate;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
public class MainClass {
private final static int N=300;
static int totalPistive=10; // < N
/**
* @param args
*/
public static void main(String[ ] args) {
//TODO Auto-generated method stub
Graph graph = new Graph();
boolean computedByMinOutArcWSum = true;
for (totalPistive=10;totalPistive < 25;){
initgraph(graph,N,totalPistive);
System.out.printf("************************ Nodes: %d, \ t Teacher: %d *****************
\ n",N,totalPistive);
```

```
List<Node> P=new ArrayList<Node>();
List<Node> C=new ArrayList<Node>();
if(computedByMinOutArcWSum){
findDominatedSetByMinOutArcWSum(graph, P, C);
}
else {
findDominatedSetByMaxNegativeInArcWSum(graph, P, C);
}
int number=output(C, P);
double weight=GetAverageWeight(graph,P,C);
System.out.println("totoal node is: \ t \ t "+number);
System.out.printf("Average weight is:\ t %6.3f \ n",weight);
graph.clear();
totalPistive+=5;
}
}
private static int output(List<Node> c, List<Node> p) {
// TODO Auto-generated method stub
System.out.println("positive:");
for (Node node : c) {
System.out.print("\ t"+node);
}
System.out.println();
System.out.println("negative → positive:");
for (Node node : p) {
System.out.print("\ t"+node);
} System.out.println();
return p.size();
}
private static void findDominatedSetByMinOutArcWSum(Graph graph, List<Node>
P, List<Node> C) {
for (Node node : graph.getNodes()) {
if (node.positive==true) {
```

```
C.add(node);
}
}
boolean hasNegativeNode = true;
while (hasNegativeNode){
hasNegativeNode = false;
double min=0;
Node minNode=null;
for (Node node : graph.getNodes()) {
if (node.positive==false && node.computeTotalInArcW()< 0 &&
node.computeTotalOutArcW()< min) {
min=node.computeTotalOutArcW();
minNode=node;
}
}
if(minNode!=null){
hasNegativeNode = true;
minNode.positive=true;
P.add(minNode);
for( Arc arc: minNode.outArcList) {
arc.weight=Math.abs(arc.weight);
}
}
}
}
private static void findDominatedSetByMaxNegativeInArcWSum(Graph graph,
List<Node> P,
List<Node> C) {
for (Node node : graph.getNodes()) {
if (node.positive==true) {
C.add(node);
}
}
```

```
boolean hasNegativeNode = true;
while (hasNegativeNode){
hasNegativeNode = false;
double max=Double.NEGATIVE-INFINITY;
Node maxNode=null;
for (Node node : graph.getNodes()) {
if (node.positive==false && node.computeTotalInArcW()< 0 &&
node.computeTotalInArcW()> max) {
max=node.computeTotalInArcW();
maxNode=node;
}
}
if(maxNode!=null){
hasNegativeNode = true;
maxNode.positive=true;
P.add(maxNode);
for( Arc arc: maxNode.outArcList) {
arc.weight=Math.abs(arc.weight);
}
}
}
}
private static double GetAverageWeight(Graph graph,List<Node> c,
List<Node> p) {
double weight=0;
int number=0;
for (Node node : graph.getNodes()) {
weight+=node.computeTotalInArcW();
number++; } for (Node node :c) {
weight-=node.computeTotalInArcW(); number–; } for (Node node :p) { weight-
=node.computeTotalInArcW();
number–; } if (number<1)
return weight;
```

```
else
return weight /number; }
private static void initgraph(Graph graph,int nNodes,int totalPositive) {
int loc=0;
// initialize all nodes
Node node=null;
for (int i=1;i<=nNodes;i++){
node=new Node("v"+String.valueOf(i),false);
graph.add(node);
}
for (int i=0;i<totalPositive;i++){
Random tempR=new Random();
loc=tempR.nextInt(nNodes);
graph.setNodeAsTeacher(loc);
}
// initialize arc list
for (int i=1;i<=nNodes;i++){
Arc arc=new Arc();
arc.from=graph.getNode(i-1);
loc=i-1;
arc.to=null;
while(arc.to==null){
while(loc==i-1){
loc=(int)(Math.random()*nNodes);
}
arc.to=graph.getNode(loc);
}
arc.weight=(Math.random()-0.5)*2;
arc.from.outArcList.add(arc);
arc.to.inArcList.add(arc);
}
}
```

```
private static void Test1graph(Graph graph) {
int loc=0;
// initialize all nodes
String [ ]NodesName={"Bob","Ann","Don","Kris","Tom"};
Node node=null;
node=new Node(NodesName[0],true);
graph.add(node);
graph.setNodeAsTeacher(0);
for (int i=1;i<NodesName.length;i++){
node=new Node(NodesName[i],false);
graph.add(node);
}
// initialize arc list
Arc arc=new Arc();
arc.from=graph.getNodeByID("Bob");
arc.to=graph.getNodeByID("Don");
arc.weight=0.7;
arc.from.outArcList.add(arc);
arc.to.inArcList.add(arc);
arc=new Arc();
arc.from=graph.getNodeByID("Bob");
arc.to=graph.getNodeByID("Tom");
arc.weight=0.7;
arc.from.outArcList.add(arc);
arc.to.inArcList.add(arc);
arc=new Arc();
arc.from=graph.getNodeByID("Bob");
arc.to=graph.getNodeByID("Kris");
arc.weight=0.7;
arc.from.outArcList.add(arc);
arc.to.inArcList.add(arc);
arc=new Arc(); arc.from=graph.getNodeByID("Ann");
```

```
arc.to=graph.getNodeByID(”Don”);
arc.weight=-0.1;
arc.from.outArcList.add(arc);
arc.to.inArcList.add(arc);
arc=new Arc();
arc.from=graph.getNodeByID(”Ann”);
arc.to=graph.getNodeByID(”Kris”);
arc.weight=-0.2;
arc.from.outArcList.add(arc);
arc.to.inArcList.add(arc);
arc=new Arc();
arc.from=graph.getNodeByID(”Don”);
arc.to=graph.getNodeByID(”Kris”);
arc.weight=-0.3;
arc.from.outArcList.add(arc);
arc.to.inArcList.add(arc);
arc=new Arc();
arc.from=graph.getNodeByID(”Don”);
arc.to=graph.getNodeByID(”Tom”);
arc.weight=-0.3;
arc.from.outArcList.add(arc);
arc.to.inArcList.add(arc);
arc=new Arc();
arc.from=graph.getNodeByID(”Kris”);
arc.to=graph.getNodeByID(”Ann”);
arc.weight=-0.2;
arc.from.outArcList.add(arc);
arc.to.inArcList.add(arc);
arc=new Arc();
arc.from=graph.getNodeByID(”Kris”);
arc.to=graph.getNodeByID(”Don”);
arc.weight=-0.3;
arc.from.outArcList.add(arc);
arc.to.inArcList.add(arc);
arc=new Arc();
arc.from=graph.getNodeByID("Kris");
arc.to=graph.getNodeByID("Tom");
arc.weight=-0.3;
arc.from.outArcList.add(arc);
arc.to.inArcList.add(arc);
arc=new Arc();
arc.from=graph.getNodeByID("Tom");
arc.to=graph.getNodeByID("Don");
arc.weight=-0.3;
arc.from.outArcList.add(arc);
arc.to.inArcList.add(arc);
arc=new Arc();
arc.from=graph.getNodeByID("Tom");
arc.to=graph.getNodeByID("Kris");
arc.weight=-0.3;
arc.from.outArcList.add(arc);
arc.to.inArcList.add(arc);
}
}
```

arc.java

```
package Dominate;
public class Arc { Node from; Node to; double weight; public Arc(){}
void clear() { from=null; to=null; } }
```

Graph.java

```
package Dominate;
import java.util.ArrayList; import java.util.List;
public class Graph {
private List<Node> V;
```

```
public Graph(){
V=new ArrayList <Node>();
}
public void clear(){
for (Node node : V) {
node.clear();
}
V.clear();
}
public Node getNodeByID(String id) {
for (Node node : V) {
if (node.equalto(id))
return node;
}
return null;
}
public Node getNode(int index){
if (index<0 ) return null;
if (index<V.size())
return V.get(index);
else
return null;
}
void add(Node node) {
V.add(node);
}
List <Node> getNodes() {
return V;
}
public Node setNodeAsTeacher(int index) {
if (index<0 ) return null;
if (index<V.size()) {
```

```
V.get(index).positive=true;
return V.get(index);
}
else
return null;
}
}
```

Node.java

```
package Dominate;
import java.util.ArrayList;
import java.util.List;
public class Node {
private String ID;
boolean positive;
int x;
int y;
int[ ] ia3 =new int [ ]{1,2,3};
int delta_plus; /* edge starts from this node */
int delta_minus; /* edge terminates at this node */
int dist; /* distance from the start node */
int prev; /* previous node of the shortest path */
int p_edge;
int l;
int w;
int h;
char Color;
private char LeftRight; // 'L', 'R', 'M'
private int order; // visiting order;
private int Degree; // zero
private char Locate; // Frontal
// private String name;
```

```
List<Arc> inArcList;
List<Arc> outArcList;
public Node(){}
public Node(String id,boolean positive ){
ID=new String(id);
this.positive=positive;
inArcList=new ArrayList<Arc>();
outArcList=new ArrayList<Arc>();
} void clear(){
ID=null;
for (Arc x:inArcList) x.clear();
for (Arc x:outArcList) x.clear();
inArcList.clear();
outArcList.clear(); }
boolean equalto(String id){
boolean flag=false;
if (getID().compareTo(id)==0) return true;
return flag; }
public double computeTotalOutArcW(){
double sum=0;
for (Arc arc : outArcList) {
sum=sum+arc.weight;
}
return sum; }
@Override
public String toString(){
return getID();
}
public double computeTotalInArcW(){
double sum=0;
for (Arc arc : inArcList) {
sum=sum+arc.weight;
}
return sum;
}
/**
* @return the ID
*/
public String getID() {
return ID;
}
/**
* @param ID the ID to set
*/
public void setID(String ID) {
this.ID = ID;
}
}
```

Appendix B

Experimental Data

The following is the data used in paper [70].

code_java.data

************************ Nodes: 300, Teacher: 10 ************************

positive:

v43 v47 v66 v68 v76 v85 v154 v208 v252 v261

negative → positive:

v149 v279 v112 v295 v288 v91 v189 v108 v280 v95 v30 v120 v248 v105 v127 v160 v75 v253 v122 v23 v230 v294 v254 v225 v104 v265 v109 v193 v74 v82 v38 v267

total node is: 32

Average weight is: 0.207

************************ Nodes: 300, Teacher: 15 ************************

positive:

v5 v18 v27 v114 v135 v145 v166 v172 v179 v180 v230 v257 v275

negative → positive:

v32 v125 v40 v11 v163 v178 v200 v261 v29 v147 v266 v167 v2 v203 v79 v231 v165 v286 v111 v282 v24 v149 v104 v130 v191 v99 v267 v248

total node is: 28

Average weight is: 0.249

************************ Nodes: 300, Teacher: 20 ************************

positive:

v7 v9 v37 v59 v64 v84 v108 v162 v163 v172 v232 v239 v244 v257 v258 v278 v282 v294

negative → positive:

v122 v135 v202 v87 v47 v117 v199 v63 v19 v18 v218 v4 v75 v170 v156 v185 v14 v269 v179 v1 v68 v277 v66 v208 v234 v35 v23 v292 v77 v50 v82 v272 v154 v219 v79 v273 v271 v20 v231 v8 v21 v72 v25 v29 v42 v196 v220

total node is: 47

Average weight is: 0.243

-End of dissertation-