A Self-Stabilizing Algorithm for Finding a Minimal Positive Influence Dominating Set in Social Networks

Guangyuan Wang[1] **Hua Wang**[1] **Xiaohui Tao**[1] **Ji Zhang**[1]

[1] Department of Maths and Computing
University of Southern Queensland, QLD, Australia 4350,
Email: guangyuan.wang@usq.edu.au

Abstract

Online social network has developed significantly in recent years. Most of current research has utilized the property of online social network to spread information and ideas. Motivated by applications in social networks (such as alcohol intervention strategies), a variation of the dominating set called a positive influence dominating set (PIDS) has been studied in the literature. However, the existing work all focused on greedy algorithms for the PIDS problem with different approximation ratios, which are limited to find approximate solutions to PIDS in large networks. In order to select a minimal PIDS (MPIDS) in large social networks, we first present a self-stabilizing algorithm for the MPIDS problem in this paper, which can find a MPIDS in an arbitrary network graph without any isolated node. It is assumed that the nodes in the proposed algorithm have globally unique identifiers, and the algorithm works 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.

Keywords: Self-stabilizing algorithm, Minimal positive influence dominating set, Graph algorithm

1 Introduction

1.1 Minimal Positive Influence Dominating Set Problem

Recently, social networks have received dramatic interest in research and development, partly due to more and more social networks are built online and the fast development of Web 2.0 applications, such as the e-learning research (Wang et al. 2009, Sun & Wang 2011) and privacy protection (Wang et al. 2005, Sun et al 2011, Sun et al. 2012). On the other hand, many different kinds of social networks in our lives such as friendship networks, telephone call networks, and academia co-authorship networks can be modeled by graph structures using vertices and edges. Some classical graph problems such as domination problems in social networks have various new applications.

The concept of the positive influence dominating set (PIDS) was introduced in 2009 by Wang et al. (2009). PIDS can deal with some social problems, such as drinking, smoking and drug 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 (Jaccard et al. 2005, Larimer & Cronce 2007, Standridge et al. 2004). For example, within the context of the drinking problem, a person named Mark never drinks and has positive impact on his direct friends (Barry, etc.), but Mark might turn into a binge drinker and have negative impact on his neighbors if Barry and other friends are binge drinkers and vice versa. So, a person can be an abstainer or a binge drinker. In order to truly alleviate the main source of the drinking problem, intervention programs are important tools to help combat some of the social problems through disseminated education and therapy via mail, Internet, or face-to-face interviews. Ideally, we want to educate all binge drinkers, since this will reduce the possibility of converted binge drinker being influenced by his binge drinking friends who are not chosen in the intervention program. On the other hand, due to the budget limitations, the lower total cost of the education and therapy program, the better. So, it is impossible to include all the binge drinkers in the intervention program. Therefore, it becomes an important research problem as to how to choose a subset of individuals to be part of the program so that the effect of the intervention program can spread through the whole group under consideration.

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) (Wang et al. 2009, 2011) 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 $\lceil \frac{d(i)}{2} \rceil$ neighbors) 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 neighbors in D, secondly every node in D also has at least half of its neighbors in D. A PIDS D is *minimal* (MPIDS) if no proper subset of D is a PIDS.

For the drinking example remarked earlier, a minimal positive influence dominating set (MPIDS) is a plausible solution since MPIDS can guarantee that by selecting MPIDS nodes to participate in the intervention program, each individual in the social network has more positive neighbors than negative ones to ensure that the intervention can result in a globally positive impact on the entire social network.

In the domination problems, finding a PIDS of minimum size is APX-hard (Wang et al. 2011). APXhardness of PIDS means that if $NP \neq P$, then PIDS has no PTAS (polynomial-time approximation scheme). Some greedy approximation algorithms have been proposed (Wang et al. 2009, 2011), which are all limited to find approximate solutions to positive influence dominating sets (PIDS) 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 program while satisfying positively dominating the whole group. Moreover, it might offer considerable benefit to both the economy and society. In this paper, we present a self-stabilizing algorithm for finding a minimal PIDS (MPIDS) in general networks.

1.2 Dijkstra's Central Daemon Model

Self-stabilization is an optimistic fault tolerance approach for distributed systems. It was introduced by Dijkstra (Dijkstra 1974, Dijkstra & van Gasteren 1986). According to his work, a self-stabilizing system is guaranteed to reach a correct state, in a finite time, regardless of its initial state (Dolev 2000). 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 behavior 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,

(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.

Recently, some self-stabilizing algorithms for dominating sets, independent sets, colorings, and matchings in graphs have been developed (Hedetniemi et al. 2003, Goddard et al. 2008, Hedetniemi et al. 2003, Manne et al. 2007). In a self-stabilizing algorithm, each node maintains its local variables, and can make decisions based on the knowledge of its neighbors' states. A node changes its local state by making a *move* (a change of local state). The algorithm is a set of rules of the form "if $P(i)$ then M", where $P(i)$ is a predicate and M is a move. A node i becomes *privileged* if $P(i)$ is true. When a node becomes privileged, it may execute the corresponding move. A *central daemon* selects, among all privileged nodes, the next node to move. If two or more nodes are privileged, one cannot predict which node will move next. In this paper, We assume the model in which no two nodes move simultaneously. Another popular scheduler is the *distributed daemon*, which selects a subset of the system processes to execute an atomic step at the same time. 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. Multiple protocols exist (Nesterenko & Arora 1999, Beauquier et al. 2002, Goddard et al. 2003) that provide such a scheduler.

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. As remarked earlier, throughout this paper, 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 neighborhood*, denotes the set of nodes to which i is adjacent. Every node $j \in N(i)$ is called a neighbor of node i. We denote by $d(i)$ the number of neighbors of node i, or its degree ($d(i) = |N(i)|$). Throughout this paper we assume G is connected and $n > 1$.

1.3 Main Results and the Organization of the Rest of the Paper

In this paper, we are interested in a minimal positive influence dominating set (MPIDS) raised from some social problems. We believe the following to be our contributions in this paper.

1. We present a self-stabilizing algorithm for finding a minimal positive influence dominating set (MPIDS) in an arbitrary connected network graph under a central daemon.
2. 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.

To the best of our knowledge, this is the first work using a self-stabilizing algorithm to find a MPIDS. The rest of this paper is organized as follows. Section 2 presents related work. Section 3 presents a self-stabilization algorithm and an illustration for finding a MPIDS. Section 4 analyzes the complexity of the algorithm. Section 5 discusses the algorithm comparison. Section 6 concludes the paper and discusses the future work.

2 Related Work

Graph algorithms have natural applications in networks and distributed systems, since a distributed system can be modeled with an undirected graph. 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 publication of Dijkstra's pioneering paper, some self-stabilizing algorithms for graph problems have been proposed in the literature, such as the self-stabilizing algorithms for dominating sets (Gairing & Johnson 2003), independent sets and matchings in graphs (Goddard et al. 2008, Hedetniemi et al. 2003). Due to the NP-complete of domination problems (Garey et al. 1979), researchers have developed some self-stabilizing algorithms for finding minimal dominating sets.

Since any maximal independent set in a graph is a minimal dominating set in that graph, a self-stabilizing algorithm for the maximal independent set problem can be viewed as a self-stabilizing algorithm for the minimal dominating set problem. Hedetniemi et al. (2003) presented two *uniform algorithms* (a distributed algorithm is said to be uniform if all of the individual processes run the same code) for the dominating set (DS) and the minimal dominating set

(MDS) problems. The algorithms all work for any connected graph and assume a central daemon. Goddard et al. (2008) proposed another uniform algorithm for finding a minimal dominating set (MDS) in an arbitrary graph under a distributed daemon.

On the other hand, some self-stabilized algorithms have been proposed in the multiple domination case. In a graph $G(V, E)$, a set of nodes $D \subseteq V$ is called a k-dominating set if each node not in D is adjacent to at least k nodes in D. When the positive integer $k = 1$, it is a question of single domination.

Kamei and Kakugawa (2003) presented two self-stabilizing algorithms for the minimal k-dominating set (MKDS) problem in a tree. The first uniform algorithm works under a central daemon and the second algorithm works under a distributed daemon. Huang et al. (2007, 2008) presented two self-stabilizing algorithms to find a minimal 2-dominating set (M2DS) under a distributed daemon (Huang et al. 2007) and a central daemon (Huang et al. 2008) respectively in an arbitrary graph.

However, there is no algorithm for the MKDS problem in arbitrary graphs that works under a distributed daemon. The proposed algorithms for the minimal k-dominating set (MKDS) either work for trees (Kamei & Kakugawa 2004) or find a minimal 2-dominating set (Huang et al. 2007, 2008). For a more detailed discussion of self-stabilizing algorithms for dominating sets, see a survey (Guellati & Kheddouci 2010).

Positive influence dominating set (PIDS) which has application in social networks can be considered as a special case of multiple domination, introduced by Wang et al. (2009). Wang et al. (2009) proposed a greedy approximation PIDS selection algorithm and analyzed its effect on a real online social network data set through simulations. Wang et al. (2011) also proved the PIDS problem is APX-hard and developed another greedy approximation algorithm and analyzed its approximation ratio.

All of these proposed algorithms can only find approximate solutions to PIDS in large social networks. On the other hand, none of these algorithms for the PIDS problem are self-stabilizing. In order to obtain a minimal PIDS (MPIDS), in this paper, we first propose a new self-stabilization algorithm for computing a MPIDS in a general network and analyze the time complexity of the proposed algorithm.

3 Self-Stabilizing Positive Influence Domination Algorithm

3.1 Formal Definition of the Problem

Let $G = (V, E)$ be a simple connected undirected graph. Assume now that for each node $i \in V$, the set $N(i)$ represents its *open neighborhood*, denotes the set of nodes to which i is adjacent. $d(i)$ represents the number of neighbors 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 $h(i) \leq N(i)$, 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 seek a MPIDS $D \subseteq V$ in which, for all $i \in V$,

$$|N(i) \cap D| \geq h(i) \tag{1}$$

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 (1) is precisely uniformly one.

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 $i \rightarrow j$. 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 at least $h(i) - k$ members in $N(i) - D$.

Definition 1 *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 2 *A set of pointers $Q(i)$ is designed as follows:*

$$Q(i) = \begin{cases} (D \cap N(i)) \cup M_i & \text{if } |N(i) \cap D| = k \leq h(i) \\ \emptyset & \text{if } |N(i) \cap D| > h(i). \end{cases} \tag{2}$$

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

Definition 3 *The boolean condition $y(i)$ is defined to be true if and only if a neighbor of i points to it.*

3.2 Proposed Algorithm

The Algorithm 1 consists of one rule (R1) is shown below. In Algorithm 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 neighbor of i points to it. $P(i)$ is a set of pointers. $Q(i)$ is counted by equation (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 1 is shown in Fig. 1.

```
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 $x(i) = y(i) \wedge P(i) = Q(i)$
```

Algorithm 1: Finding Minimal Positive Influence Dominating Set

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, a minimal positive influence dominating set (MPIDS) $D = \{i \in V | x(i) = true\}$ can be identified.

Lemma 1 *If Algorithm 1 stabilizes then $D = \{i \in V | x(i) = true\}$ is a minimal positive influence dominating set (MPIDS) satisfying inequality (1).*

Proof: Suppose that D satisfies inequality (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 $i's$ neighbor 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 (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.

3.3 An Illustration

The example in Fig. 1 illustrates the execution of Algorithm 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 1 to make a move or reset the value of $P(i)$.

In the first subgraph of Fig. 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 definition of Algorithm 1, after a serials of moves, the system reaches a legitimate state. As the last subgraph of Fig. 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).

4 The Stabilization Time of Algorithm 1

In this section, we show the convergence of Algorithm 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 4 *A move is an in-move if it causes $x(i)$ to become true, thereby causing a node i to enter D.*

Lemma 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 neighbor 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 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 smaller than 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 show 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 1 *Algorithm 1 always stabilizes, and finds a minimal positive influence dominating set (MPIDS).*

Proof: In light of Lemma 1 we see that if Algorithm 1 is stabilizing it always finds a minimal positive influence dominating set (MPIDS). We need only prove stabilization. It suffices to show that every node makes only a finite number of in-moves. By Lemma 2, node n, which has 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 neighbors. when node n is not making an in-move, using Lemma 2 again, node $n - 1$ makes at most a finite number of moves. It is easy to show this argument can be repeated, showing that each node can make only finitely many in-moves 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 1.

Theorem 2 *Algorithm 1 produces a minimal positive influence dominating set (MPIDS) and stabilizes in $O(n^2)$ steps.*

Proof: From Lemma 1 and Theorem 1, we see that Algorithm 1 produces a MPIDS. We need only prove Algorithm 1 stabilizes in $O(n^2)$ steps. By Lemma 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 $\lceil \frac{\Delta}{2} \rceil n$ changes of $P(i)$-value on all nodes in all the 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 1 is $O(n^2)$ steps.

5 Algorithm Comparison

In this section we respectively present and discuss the existing self-stabilizing algorithms and greedy algorithms for dominating sets. We also compare our algorithm with them.

Hedetniemi et al. (2003) presented two uniform algorithms for the dominating set (DS) and the minimal dominating set (MDS) problems. The algorithms work for any connected graph. 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 dominating set (DS) problem stabilizes in linear time ($O(n)$ steps) under a central daemon. The second algorithm calculates a MDS. The main idea of this algorithm is that it allows a node to join the set S, if it has no neighbor in S. On the other hand, a node that is already a member of S, and has a neighbor that is also a member of S, will leave the set if all its neighbors are not pointing to it. Thus, after stabilization the set S will be a MDS. The algorithm for the minimal dominating set (MDS) problem stabilizes in $O(n^2)$ steps under a central daemon.

Recently, Goddard et al. (2008) proposed another uniform self-stabilizing algorithm for finding a minimal dominating set (MDS) in an arbitrary graph under a distributed daemon. 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,

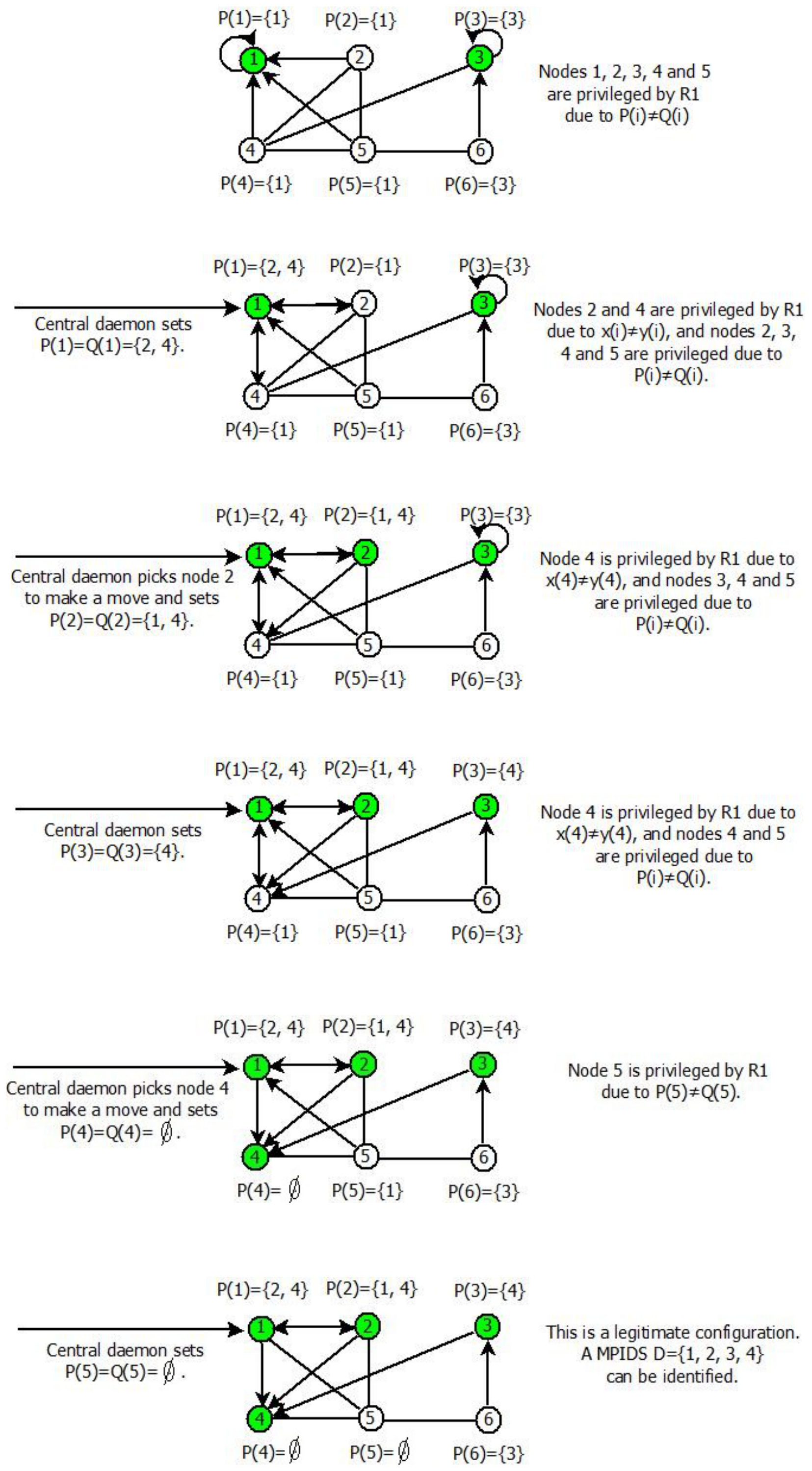
P(1)={1}
P(2)={1}
P(3)={3}
P(4)={1}
P(5)={1}
P(6)={3}
Nodes 1, 2, 3, 4 and 5 are privileged by R1 due to P(i)≠Q(i)
Central daemon sets P(1)=Q(1)={2, 4}.
P(1)={2, 4}
P(2)={1}
P(3)={3}
P(4)={1}
P(5)={1}
P(6)={3}
Nodes 2 and 4 are privileged by R1 due to x(i)≠y(i), and nodes 2, 3, 4 and 5 are privileged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

Figure 1: An Example to illustrate the execution of Algorithm 1.

Table 1: Algorithms for dominating set

Reference	Output	Required topology	Self-stabilizing	Daemon	Complexity
Hedetniemi et al. (2003)-1	DS	Arbitrary	Yes	Central	$O(n)$ steps
Hedetniemi et al. (2003)-2	MDS	Arbitrary	Yes	Central	$O(n^2)$ steps
Goddard et al. (2008)	MDS	Arbitrary	Yes	Distributed	$O(n)$ steps
Kamei & Kakugawa (2003)-1	MKDS	Tree	Yes	Central	$O(n^2)$ steps
Kamei & Kakugawa (2003)-2	MKDS	Tree	Yes	Distributed	$O(n^2)$ steps
Huang et al. (2007)	M2DS	Arbitrary	Yes	Distributed	
Huang et al. (2008)	M2DS	Arbitrary	Yes	Central	$O(n)$ steps
Wang et al. (2009)	PIDS	Arbitrary	Greedy		
Wang et al. (2011)	PIDS	Arbitrary	Greedy		$H(\delta)$ AR
This paper	MPIDS	Arbitrary	Yes	Central	$O(n^2)$ steps

and an integer to count a node's neighbors that are members of the MDS. The algorithm allows an undominated node that has smaller identifier than any undominated neighbor 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 neighbors. 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 (2003) presented two uniform algorithms for the minimal k-dominating set (MKDS) problem in a tree. The first algorithm allows a node to join the set under construction S if it has fewer than k neighbors in S, and to leave the set S if it has more than k neighbors in S. The first algorithm works for a central daemon. Based on this idea, in the second algorithm, a node having more than k neighbors in the set under construction S will first make a request to leave S, and then leaves the set S only if its identifier is the smallest among all the neighbors requesting to leave S. So, after stabilization the set S will become a minimal k-dominating set (MKDS). The second algorithm works under a distribute daemon. The time complexity of the two algorithms are both $O(n^2)$ steps.

Huang et al. (2007) presented a self-stabilizing algorithm to find a minimal 2-dominating set (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 neighbors 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 neighbors are either in the set under construction or dominated by more than two nodes.

Huang et al. (2008) presented another self-stabilizing algorithm for finding a minimal 2-dominating set (M2DS) in an arbitrary graph. The algorithm allows a node to join the set under construction S if it has fewer than 2 neighbors in S, and to leave the set S if it has more than 2 neighbors in S. The algorithm works under a central daemon, with liner time complexity.

Wang et al. (2009) introduced the notion of 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 chosen into the PIDS to achieve the goal that every individual in the community has more positive neighbors than negative neighbors. They also presented another greedy approximation algorithm and gave theoretical analysis about its approximation ratio (AR) in 2011 (Wang et al. 2011). 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.

In order to select a small size of positive influence dominating set (PIDS) in large social networks, we present a uniform algorithm for finding a minimal PIDS 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 neighbors having the smallest identifiers if i has less than $h(i)$ neighbors in the set under construction D. On the other hand, if a node i has more than $h(i)$ neighbors in the set D then $P(i)$ will point to emptyset; otherwise $P(i)$ will point to its unique neighbors that are members of the set D. The algorithm allows a node to join the set D if some neighbor is pointing to it, and to leave the set D otherwise. So after stabilization, the set D will become a 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 a MPIDS, which can find an exact solution for PIDS (minimal).

The algorithms we compared in this section are summarized in Table 1. As we can see, the basic ideas of the first six algorithms are self-stabilizing, and the algorithms for the PIDS problem are greedy. Our algorithm is the first work using a self-stabilizing approach to discuss the MPIDS problem.

6 Conclusions and Future work

In this paper, we have proposed a self-stabilizing distributed algorithm to find a minimal positive influence dominating set (MPIDS) which arises from some social problems in social networks; Algorithm 1 can be used for an arbitrary connected graph. All previously known algorithms are approximate greedy algorithms. We have also shown the stabilization time of Algorithm 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. For weighted domination, each node i has an allowable range of values $\{(0, 1, ..., b(i)\}$ (in the previous section $b(i)$ was uniformly 1) and is assigned a weight $w(i)$. Each node has a target $t(i)$ for the sum $\sum_{j\in N(i)} w(j)$. We want a minimal assignment of values that satisfy the constraints. A primitive way to achieve this is to provide each node with $b(i)$ flags each with separate ID. It is more efficient though to provide each node with a counter $X(i)$ limited to the range and an array of weights $P(i)$ that counts how many times the node points to each neighbor.

We can also extend these ideas even further to other graph dominations such as weak, strong and optional domination (Haynes et al. 1998). It can also be altered to allow a node to have weights in a range $\{-b'(i), ..., b(i)\}$ and so handle minus domination.

References

Beauquier, J., Datta, A. K., Gradinariu, M. & Magniette, F. (2002), 'Self-Stabilizing Local Mutual Exclusion and Daemon Refinement', *Chicago Journal of Theoretical Computer Science* **2002**(1).

Dijkstra, E. W. (1974), 'Self-stabilizing Systems in Spite of Distributed Control', *Commun. ACM* **17**(11), 643–644.

Dijkstra, E. W. & van Gasteren A. J. M. (1986), 'A Simple Fixpoint Argument Without the Restriction to Continuity', *Acta Inf.* **23**(1), 1–7.

Dolev, S. (2000), *Self-Stabilization*, MIT Press.

Gairing, M., Hedetniemi, S. T., Kristiansen, P. & McRae, A. A. (2003), Self-Stabilizing Algorithms for {k}-Domination, *in* 'Self-Stabilizing Systems', pp. 49–60.

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

Goddard, W., Hedetniemi, S. T., Jacobs, D. P. & Srimani, P. K. (2003), A Self-Stabilizing Distributed Algorithm for Minimal Total Domination in an Arbitrary System Grap, *in* 'IPDPS', pp. 240.

Goddard, W., Hedetniemi, S. T., Jacobs, D. P. Srimani, P. K. & Xu, Z. (2008), 'Self-Stabilizing Graph Protocols', *Parallel Processing Letters* **18**(1), 189–199.

N. Guellati, N. & Kheddouci, H. (2010), 'A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs', *Parallel Distrib. Comput.* **70**(4), 406–415.

Haynes, T. W., Hedetniemi, S. T. & Slater, P. J. (1998), *Domination in Graphs: Advanced Topics*, Marcel Dekker.

Haynes, T. W., Hedetniemi, S. T. & Slater, P. J. (1998), *Fundamentals of Domination in Graphs*, Marcel Dekker.

Hedetniemi, S. M., Hedetniemi, S. T., Jacobs, D. P. & Srimani, P. K. (2003), 'Self-stabilizing algorithms for minimal dominating sets and maximal independent sets', *Computer Mathematics and Applications* **46**(5-6), 805–811.

Hedetniemi, S. T., Jacobs, D. P. & Srimani, P. K. (2003), 'Linear time self-stabilizing colorings', *Information Processing Letters* **87**(5), 251–255.

Huang, T. C., Chen, C. Y. & Wang, C. P. (2008), 'A Linear-Time Self-Stabilizing Algorithm for the Minimal 2-Dominating Set Problem in General Networks', *Inf. Sci. Eng.* **24**(1), 175–187.

Huang, T. C., Lin, J. C., Chen, C. Y. & Wang, C. P. (2007), 'A self-stabilizing algorithm for finding a minimal 2-dominating set assuming the distributed demon model', *Computers and Mathematics with Applications* **54**(3), 350–356.

Jaccard, J., Blanton, H. & Dodge, T. (2005), 'Peer influences on risk behavior: Analysis of the effects of a close friend', *Developmental Psychology* **41**(1), 135–147.

Kamei, S. & Kakugawa, H. (2003), A self-stabilizing algorithm for the distributed minimal k-redundant dominating set problem in tree network, *in* 'PDCAT' pp. 720-724.

Kamei, S. & Kakugawa, H. (2004), 'A Self-Stabilizing Distributed Algorithm for the Steiner Tree Problem', *IEICE Transactions* **87-D**(2), 299–307.

Larimer, M. E., & Cronce, J. M. (2007), 'Identification, prevention, and treatment revisited: Individual-focused college drinking prevention', *Addictive Behaviors* **32**, 2439–2468.

Manne, F., Mjelde, M., Pilard, L. & Tixeuil, S. (2007), 'A New Self-Stabilizing Maximal Matching Algorithm', *in* 'SIROCCO' pp. 96–108.

Nesterenko, M. & Arora, A. (1999), 'Stabilization-Preserving Atomicity Refinement', *in* 'DISC' pp. 254–268.

Standridge, J. B., Zylstra, R. G. & Adams, S. M. (2004), 'Alcohol consumption: An overview of benefits and risks', *Southern Medical Journal* **97**(7), 664–672.

Sun L. & Wang H. (2011), 'Access control and authorization for protecting disseminative information in E-learning workflow', *Concurrency and Computation: Practice and Experience* **23**(16), 2034-2042.

Sun, X., Wang H., Li J. & Zhang Y. (2012), 'Satisfying Privacy Requirements Before Data Anonymization', *Comput. J.* **55**(4), 422-437.

Sun, X., Wang H., Li J. & Pei J. (2011), 'Publishing anonymous survey rating data', *Data Min. Knowl. Discov.* **23**(3), 379-406.

Wang, F., Camacho, E. & Xu, K. (2009), 'Positive Influence Dominating Set in Online Social Networks', *in* 'COCOA' pp. 313–321.

Wang, F., Du, H., Camacho, E., Xu, K., Lee, W., Shi, Y. & Shan, S. (2011), 'On positive influence dominating sets in social networks', *Theor. Comput. Sci.* **412**(3), 265–269.

Wang H., Cao J. & Zhang Y. (2005), 'A flexible payment scheme and its role based access control', *IEEE Transactions on Knowledge and Data Engineering* **17**(3), 425–436

Wang H., Zhang Y., Cao J. (2009), 'Effective collaboration with information sharing in virtual universities', *IEEE Transactions on Knowledge and Data Engineering*, **21**(6), 840-853.