Game-Theoretic Approach to Self-Stabilizing Minimal Independent Dominating Sets

Li-Hsing Yen and Guang-Hong Sun

Department of Computer Science, National Chiao Tung University, Taiwan
lhyen@nctu.edu.tw, martian206@gmail.com

Abstract. An independent dominating set (IDS) is a set of vertices in a graph that ensures both independence and domination. The former property asserts that no vertices in the set are adjacent to each other while the latter demands that every vertex not in the set is adjacent to at least one vertex in the IDS. We extended two prior game designs, one for independent set and the other for dominating set, to three IDS game designs where players independently determine whether they should be in or out of the set based on their own interests. Regardless of the game play sequence, the result is a minimal IDS (i.e., no proper subset of the result is also an IDS). We turned the designs into three self-stabilizing distributed algorithms that always end up with an IDS regardless of the initial configurations. Simulation results show that all the game designs produce relatively small IDSs with reasonable convergence rate in representative network topologies.

Keywords: Independent dominating set · Self-stabilization · Distributed algorithm · Game theory.

1 Introduction

Given an undirected graph $G = (V, E)$, where V is the vertex set and E is the edge set, $S \subseteq V$ is an *independent set* if no vertices in S are adjacent to one another. Vertex set $S \subseteq V$ is a *dominating set* if every vertex in $V \setminus S$ is adjacent to at least one vertex in S. Vertex set $S \subseteq V$ is an *independent dominating set* (IDS) if S is both independent and dominating. Fig. 1 shows an example of IDS.

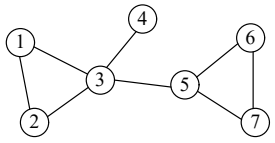


Fig. 1. Both vertex sets $\{3, 6\}$ and $\{3, 7\}$ are IDSs. Vertex set $\{3, 5\}$ is not an IDS because it is not independent.

A dominating set (independent or not) is said to be *minimal* if it contains no proper subset that is also a dominating set. A dominating set (independent or not) is said to be *minimum* if it is of the minimum cardinality. Finding a minimum dominating set is NP-hard [6] for which many heuristics and approximations have been proposed. In this paper, we focus on minimal (rather than minimum) IDS identification in a distributed system. A distributed system consisting of multiple processes interconnected by communication links can be modeled as a connected undirected graph where vertices represent processes and edges represent communication links between processes. A dominating set in a distributed system may represent a collection of servers that provide some type of service or resource to adjacent non-server processes. An example is a set of cluster heads in a wireless sensor network that collect sensory data from nearby non-head sensor nodes and transmit the data to a remote base station. Here cluster heads should not hear each other's signal (i.e., independent to each other) to avoid potential transmission collisions and bandwidth contentions.

In this paper, we are interested in the design of self-stabilizing distributed algorithms for IDS identification. After a transient fault, a system is still alive but could be in an arbitrary, possibly illegitimate state. Starting from an arbitrary state, a *self-stabilizing* distributed algorithm takes actions step by step bringing the system back to legitimate states in finite time [3]. Self-stabilization provides a paradigm for designing distributed algorithm that autonomously resolves transient faults without external intervention. There have been some self-stabilizing distributed algorithms that identify minimal dominating sets [10, 17, 12, 15, 7] and maximal independent sets [14, 11, 8, 15]. However, to the best knowledge of the authors, there has been no self-stabilizing algorithms for IDS in the literature.

We have designed self-stabilizing distributed algorithms under the framework of game theory [18, 19]. Game theory helps us derive stability, correctness, and efficiency properties of the corresponding design. We follow the same thread to derive the solutions, which include three games and associated self-distributed algorithms. One of the games is a modification on the maximal independent set (MIS) game we proposed in [19] and the others are extensions to our previously-proposed minimal dominating set (MDS) game [18].

We conducted simulations to evaluate the proposed approach. Various types of network topologies were considered, including Erdös-Rényi (ER) [5], Watts-Strogatz (WS) [16], and Barabási-Albert (BA) [1] graphs. Simulation results indicate that the approach based on MIS game outperforms the other twos in the size of dominating sets and also the time to stabilization.

The rest of this paper is organized as follows. Section 2 presents background knowledge and related work. The proposed game-theoretic approaches to the minimal IDS are presented in Section 3. Section 4 compares the three proposed approaches in terms of the cardinality of IDS and the time to stabilization. Section 5 concludes this paper.

2 Background and Related Work

To clearly define self-stabilization for a system, we need a predicate over all states of the system [13] that differentiates correct or *legitimate* states from incorrect or *illegitimate* states. Concerning our problem, a system state is legitimate if all processes that claim themselves as dominators indeed form a minimal IDS. A distributed algorithm is self-stabilizing with respect to the predicate if, starting from *any* system state, *any* sequence of state transitions leads to a legitimate state and all states following it (if any) are also legitimate. The proposed approach corresponds to *silent* stabilizing [4], meaning that all legitimate states are quiescent states for which no further state transition is possible. Some approaches do not achieve self-stabilization but weak stabilization, which implies the existence of *at least one* sequence of state transitions that leads the system to a legitimate state. Weak stabilization implies stabilization under some reasonable conditions [9].

We assume an asynchronous system, where processes perform computations at arbitrary relative speeds and interleave one another in an arbitrary order. We also assume a *central daemon*, which is a conceptual scheduler that allows only one process to execute at a time. Note this scheduler is not needed at run time. We need this daemon only for analysis and verification purpose.

All the games considered here are dynamic non-cooperative games, where players make decisions by turns (asynchronously) only for their own good. This abstracts the asynchronous nature of process's execution and communication speeds. However, the game model assumes that a player knows the current decisions of all other's when making a decision. This assumption demands some state or communication synchronization facility among processes. Most self-stabilizing approaches assume *shared-variable* inter-process communication model, where a process can update its own variables but not other's. A process can only read variables of its own and all its neighbor's. Therefore, if a process need access some process's variable value to make its decision and that process is not its neighbor, it is possible that the value is not the latest. This limitation may cause problem, as we shall explain later.

Designing self-stabilizing approaches in the framework of game theory is still a new concept with a very few studies in the literature. In [2], selfish processes seeking their own payoffs will form a spanning tree. This approach is weakly stabilizing. We proposed a game-theoretical approach to minimal multi-dominating set in [18], which is a generalization of dominating set by allowing each vertex to demand a specific number of adjacent dominators. Though the game design itself guarantees stability (Nash equilibrium), the corresponding distributed algorithms are weakly stabilizing. The gap comes from the memory access constraint imposed by the shared-variable inter-process communication model.

We also proposed another game-theoretical approach to *maximal* weighted independent sets [19]. An independent set is maximal if there is no proper superset of it that is also an independent set. A weighted independent set aims to maximize the total vertex weight in an independent set. The game designs guar-

antee stability and are Pareto optimal. The distributed algorithms transformed from the game designs are self-stabilizing.

Theoretically speaking, there are two possible approaches to identifying a minimal IDS in a graph. One is to find a minimal dominating set that is also independent. However, no self-stabilizing dominating set approach ever considers the constraint of independence. The other is to identify a maximal independent set because maximal independent sets are also minimal dominating sets. However, existing approaches to maximal independent sets normally seek large cardinalities. In contrast, when finding a minimal IDS, we usually want to minimize the cardinality of the IDS as much as we can. Existing approaches to maximal independent set thus do not perform well.

3 IDS Games

This section presents three game designs and associated self-stabilizing algorithms as distributed approaches to the IDS problem. The game-theoretic approaches here are extensions to our previous work in [18, 19]. We assume a distributed system consisting of n processes (vertices) interconnected by communication links. Each process is a player in the IDS game. Let $P = \{p_1, p_2, \ldots, p_n\}$ be the *player set* that represents all processes in the system. For each $p_i \in P$, p_i's *strategy set* S_i is the collection of all p_i's feasible decisions. Each player p_i has a strategy variable $c_i \in S_i = \{0, 1\}$, which indicates whether p_i is a member of IDS. The *strategy space* of the game $\Sigma = S_1 \times S_2 \times \cdots \times S_n$ is the Cartesian product of all strategy sets. A *strategy profile* $C = (c_1, c_2, \ldots, c_n) \in \Sigma$ is a tuple of n strategies, where $c_i \in S_i$. For a specific p_i, we may express C as $C = (c_i, c_{-i})$, where $c_{-i} = (c_1, c_2, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n)$ denotes the set of all player's strategies excluding c_i. p_i's payoff with respect to a strategy profile C is given by p_i's utility function $u_i(C)$. We define *IDS game* as $\Gamma = [P; \{S_i\}_{i=1}^n; \{u_i(\cdot)\}_{i=1}^n]$. The objective of each player $p_i \in P$ in Γ is $\max_{c_i \in S_i} u_i(c_i, c_{-i})$.

We do not presume any action sequence among players to reflect the nondeterministic nature of process's executions in an asynchronous distributed system. Due to numerous potential state transition sequences, players do not perform backward induction but are rather myopic in the sense that a player chooses a *best response* that maximizes its utility with respect to the current strategy profile. Formally, the best response for player p_i is a function

$$r_i(c_{-i}) = \{c_i \in S_i | \forall c_i' \in S_i : u_i(c_i, c_{-i}) \geq u_i(c_i', c_{-i})\}. \tag{1}$$

Players can change their strategies whenever their current strategies are not their best responses. It is theoretically possible that such strategy changes never stop and the game does not end up with a stable solution. A strategy profile is said to be a *Nash equilibrium* if every player's current strategy is already his best response. A Nash equilibrium corresponds to silent stabilization of a self-stabilizing distributed algorithm. Our goal is to define $u_i(\cdot)$ for every $p_i \in P$ such that Γ always ends up with a Nash equilibrium regardless of game play sequence, which renders a self-stabilizing distributed approach to minimal IDS.

In all the game designs, N_i is the set of players adjacent to p_i. We assume no isolated vertices so $|N_i| \geq 1$ for all p_i.

3.1 MIS-based IDS Game

The first game modifies the MIS game proposed in [19]. We define the utility of p_i associated with a strategy profile $C = (c_i, c_{-i})$ as

$$u_i(C) = c_i \left(1 - \alpha \sum_{p_j \in L_i} c_j\right), \tag{2}$$

where α is constant greater than 1 and $L_i = \{p_j | p_j \in N_i, \deg(p_j) \geq \deg(p_i)\}$. The game here differs from the prior MIS game in that this game aims to identify an MIS with *small* cardinality (rather than large cardinality as the prior MIS game does.) This can been seen from that fact that any neighboring vertex of p_i in L_i, which has equal or higher node degree than p_i, can prevent p_i from declaring itself as a member of the IDS. The stability of the game can be proved in a way analogous to that for the asymmetric weighted MIS game [19]. In fact, the MIS-based IDS game is a variant of the asymmetric weighted MIS game.

The best response of p_i is $c_i = 0$ if $c_j = 1$ for any $p_j \in L_i$. It would rather choose $c_i = 1$ if $c_j = 0$ for all $p_j \in L_i$. By the guidelines proposed in [19], the game design can be transformed into the following guarded commands:

R1 $c_i \neq 0 \wedge \exists p_j \in L_i, c_j = 1 \rightarrow$
$c_i := 0$
R2 $c_i \neq 1 \wedge \nexists p_j \in L_i, c_j = 1 \rightarrow$
$c_i := 1$

A guarded command is a condition (i.e., a Boolean expression) followed by a statement. The condition and statement are separated by $\rightarrow$. The statement of a command can be executed only when the preceding condition is evaluated true.

Table 1 shows a possible game play sequence of the MIS-based IDS game for the network topology shown in Fig. 1. Observe that the game ends up with an IDS $\{3, 7\}$, at which time no player has the incentive to further change his strategy.

3.2 Symmetric MDS-based IDS Game

The second game is a modification of the MDS game proposed in [18]. Formally, given $C = (c_1, c_2, \ldots, c_n)$, define

$$v_i(C) = \sum_{p_j \in M_i} c_j \tag{3}$$

Table 1. A possible game play sequence of the MIS-based IDS game

Step	c_1	c_2	c_3	c_4	c_5	c_6	c_7
0	1	1	0	0	0	1	0
1	1	1	0	0	$\underline{1}$	1	0
2	1	1	$\underline{1}$	0	1	1	0
3	1	$\underline{0}$	1	0	1	1	0
4	1	0	1	0	1	$\underline{0}$	0
5	1	0	1	0	$\underline{0}$	0	0
6	1	0	1	0	0	0	$\underline{1}$
7	$\underline{0}$	0	1	0	0	0	1

for each player p_i, where $M_i = N_i \cup \{p_i\}$. Also define $g_i(C)$ as

$$g_i(C) = \begin{cases} \alpha & \text{if } v_i(C) = 1 \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where $\alpha > 0$ is a constant. Define

$$w_i(C) = \sum_{p_j \in N_i} c_i c_j \gamma, \tag{5}$$

where $\gamma > n\alpha$ is a constant. The utility function of p_i is defined as

$$u_i(C) = \begin{cases} \left(\sum_{p_j \in M_i} g_j(C)\right) - \beta - w_i(C) & \text{if } c_i = 1 \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

where β is another constant such that $0 < \beta < \alpha$.

When two neighboring players are both in the set, both players will have negative utilities because $\gamma > n\alpha$. Any of them can get a positive gain by leaving the set. However, none of them has the priority over the other to stay in the set. Which one will stay is pure stochastic. This is why this game is referred to as symmetric game. Later we will present an asymmetric design.

The stability of the symmetric MDS-based IDS game can be proved by showing that this game is an exact potential game, i.e., there exists an exact potential function $\pi(C)$ such that $\pi(c_i^*, c_{-i}) - \pi(c_i, c_{-i}) = u_i(c_i^*, c_{-i}) - u_i(c_i, c_{-i})$ whenever an player p_i changes its strategy from c_i to c_i^*.

Theorem 1. *The symmetric MDS-based IDS game is an exact potential game.*

Proof. The function

$$\pi(C) = \left(\sum_{j=1}^{n} \sum_{k=0}^{v_j(C)} h_j(k) \right) - \eta(C), \tag{7}$$

where

$$h_i(k) = \begin{cases} \alpha & \text{if } k = 1 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

and

$$\eta(C) = \beta \sum_{j=1}^{n} c_j + \frac{1}{2} \sum_{j=1}^{n} w_j(C), \tag{9}$$

is an exact function. If $(c_i, c_i^*) = (0, 1)$,

$$\begin{aligned} \pi(C^*) - \pi(C) &= \sum_{p_j \in M_i} g_j(C^*) - \beta \\ &= u_i(C^*) - u_i(C). \end{aligned} \tag{10}$$

If $(c_i, c_i^*) = (1, 0)$,

$$\begin{aligned} \pi(C^*) - \pi(C) &= -\sum_{p_j \in M_i} g_j(C) + \beta + |\Omega_i|\gamma \\ &= u_i(C^*) - u_i(C). \end{aligned} \tag{11}$$

□

In an exact potential game, Nash equilibrium always exists and can be found by player's individual movements. In fact, exact potential games enjoy the *finite improvement property*, which means every game play sequence is finite.

Table 2. A possible game play sequence of the Symmetric MDS-based IDS game

Step	c_1	c_2	c_3	c_4	c_5	c_6	c_7
0	1	1	0	0	0	1	0
1	1	1	0	$\underline{1}$	0	1	0
2	$\underline{0}$	1	0	1	0	1	0

Table 2 shows a possible game play sequence of the symmetric MDS-based IDS game running on the network topology shown in Fig. 1. Here no node has a priority other any others. Therefore, p_3 cannot become an MIDS member when some of its neighbors (i.e. p_4) is already in the set.

There are extra considerations when transforming the game design into guarded commands. When making decisions, p_i has to know the value of $v_j(C)$ for all $p_j \in M_i$. This involves the access of c_k for $p_k \in M_j$. However, with the shared-variable inter-process communication model, p_i cannot read the value of c_k for all $p_k \in M_j \setminus M_i$. To overcome this limitation, each process p_i maintains an auxiliary variable d_i to denote the current domination status of p_i. More explicitly,

$$d_i = \begin{cases} \text{UNDER if } |\{p_j | p_j \in M_i, c_j = 1\}| < 1, \\ \text{EQUAL if } |\{p_j | p_j \in M_i, c_j = 1\}| = 1, \\ \text{OVER} \quad \text{if } |\{p_j | p_j \in M_i, c_j = 1\}| > 1. \end{cases} \tag{12}$$

However, when $p_j \in M_i$ changes c_j, p_j cannot update d_i as a constraint imposed by the shared-variable inter-process communication model. It is p_i itself that should update d_i. The result is a distributed algorithm consisting of six rules (R1 - R6) in each process p_i as shown below.

R1 $|\{p_j|p_j \in M_i, c_j = 1\}| < 1 \wedge d_i \neq \text{UNDER}$
$\rightarrow d_i := \text{UNDER}$
R2 $|\{p_j|p_j \in M_i, c_j = 1\}| = 1 \wedge d_i \neq \text{EQUAL}$
$\rightarrow d_i := \text{EQUAL}$
R3 $|\{p_j|p_j \in M_i, c_j = 1\}| > 1 \wedge d_i \neq \text{OVER}$
$\rightarrow d_i := \text{OVER}$
R4 $\exists p_j \in N_i, c_j = 1 \wedge c_i \neq 0$
$\rightarrow c_i := 0$
R5 $\nexists p_j \in N_i, c_j = 1 \wedge \exists p_j \in M_i, d_j = \text{UNDER} \wedge c_i \neq 1$
$\rightarrow c_i := 1$
R6 $\nexists p_j \in N_i, c_j = 1 \ \wedge \ \forall p_j \in M_i, d_j = \text{OVER} \wedge c_i \neq 0$
$\rightarrow c_i := 0$

The first three commands are for the maintenance of d_i. The last three commands implement player's best response. It is possible that the conditions of two commands are both true at the same time. In that case, we assume that any one of them and only one of them can be executed. As a consequence, the value of c_i and d_i may be inconsistent at some instant. An example is $c_i = 1$ but $d_i = \text{UNDER}$. Note that the problem remains even if p_i updates d_i whenever it updates c_i because the value of d_i also depends on c_j of another neighbor $p_j \in N_i$. Though the game design itself ensures stability, this memory access constraint makes the game implementation weakly-stabilizing [18].

3.3 Asymmetric MDS-based IDS Game

The symmetric MDS-based IDS game does not give priority to any vertex to stay in the set when one of two neighboring vertices should leave the set to preserve independence. However, giving priority to the vertex with higher degree, which is also used in the design of the MIS-based IDS game, may hopefully yield fewer set members. For this reason, we modified the symmetric MDS-based IDS game to incorporate the degree-based priority scheme. The result is referred to as asymmetric MDS-based IDS game.

The only difference with the symmetric MDS-based game is the definition of $w_i(\cdot)$ function. When determining whether p_i should leave the set, p_i only cares those that has a degree higher than p_i. Therefore,

$$w_i(C) = \sum_{p_j \in L_i} c_i c_j \gamma. \tag{13}$$

Table 3 shows a possible game play sequence of the asymmetric MDS-based IDS game with the network topology shown in Fig. 1. Here high-degree vertices like p_5 and p_3 can join the set even if some of their neighbors is already in the set.

Concerning the corresponding distributed algorithm, R1 to R3 remain unchanged. R4 to R6 are changed to

Table 3. A possible game play sequence of the asymmetric MDS-based IDS game

Step	c_1	c_2	c_3	c_4	c_5	c_6	c_7
0	1	1	0	0	0	1	0
1	1	1	0	0	$\underline{1}$	1	0
2	1	1	$\underline{1}$	0	1	1	0
3	$\underline{0}$	1	1	0	1	1	0
4	0	1	1	0	1	$\underline{0}$	0
5	0	1	1	0	$\underline{0}$	0	0
6	0	1	1	0	0	$\underline{1}$	0
7	0	$\underline{0}$	1	0	0	1	0

R4 $\exists p_j \in L_i, c_j = 1 \wedge c_i \neq 0$
$\rightarrow c_i := 0$
R5 $\nexists p_j \in L_i, c_j = 1 \wedge \exists p_j \in M_i, d_j = \text{UNDER} \wedge c_i \neq 1$
$\rightarrow c_i := 1$
R6 $\nexists p_j \in L_i, c_j = 1 \wedge \forall p_j \in M_i, d_j = \text{OVER} \wedge c_i \neq 0$
$\rightarrow c_i := 0$

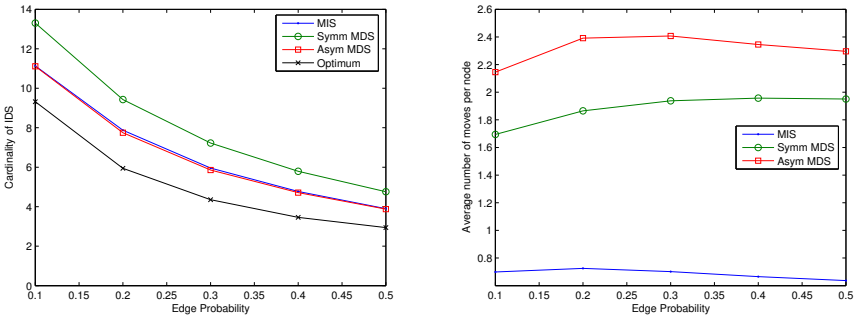
To see the stability of the asymmetric MDS-based IDS game, observe that the degree-based priority together with some tie-breaking information such as node identifier is transitive and asymmetric. Therefore, it is impossible to find any game play sequence that contains cycles. Since the strategy space Σ is finite, the acyclic property ensures stability.

4 Simulation Results

We measured the quality of IDS by its size and the time efficiency of self-stabilizing algorithms by the number of moves they take to reach stability. We conducted simulations to study the performance of all the proposed game-theoretic approaches. The quality results are compared with the optimum that was found by brute force. We considered representative network topologies generated by three models in the simulations: ER, WS, and BA. We fixed the number of vertices to be 30. Since self-stabilizing algorithms are supposed to start from arbitrary states, we assigned randomly generated values to variables (c_i's were randomly assigned either 0 or 1 and d_i's were either UNDER, EQUAL, or OVER with equal probability). Each average was obtained over 1000 runs.

The ER model [5] generates random graphs. In a random graph, whether an edge exists between a pair of vertices is an independent event with probability p_e. We varied the value of p_e from 0.1 to 0.5. The result is shown in Fig. 2. As p_e increases, the expected number of edge also increases and the graph become denser. As a result, smaller IDS can be found with larger p_e value.

The comparisons among the three proposed approaches reveal that the MIS-based IDS game (denoted by 'MIS') and the asymmetric MDS-based IDS game (denoted by 'Asym MDS') performed nearly the same. The symmetric MDS-based IDS game (denoted by 'Symm MDS') was inferior to the others. This is

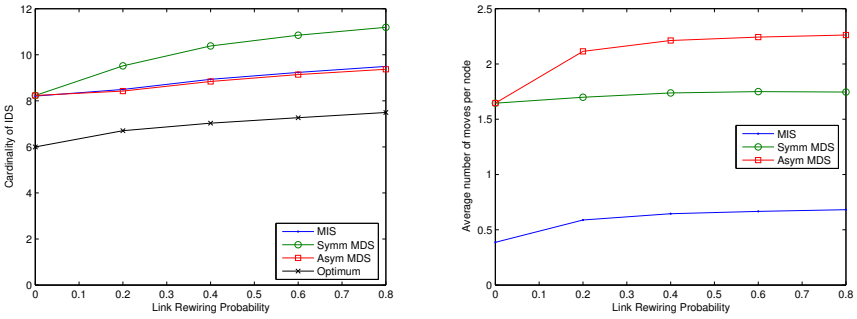


(a) Average size of IDS (b) Average number of moves per process

Fig. 2. Performance in ER graphs (30 vertices)

justifiable because Symm MDS does not favor vertices with higher node degree. In terms of convergence time, MIS performs the best, followed by Symm MDS and then Asym MDS. This is because the latter two need extra moves to update d_i's.

The WS model [16] builds a small-world network by first creating a regular network, where each vertex has exactly n_k links connecting to its n_k nearest neighbors. The regular network is then converted to a small-world network by rewiring each link with a probability p_r to a randomly chosen vertex. We varied the value of p_r from 0 to 0.8. The result is shown in Fig. 3.

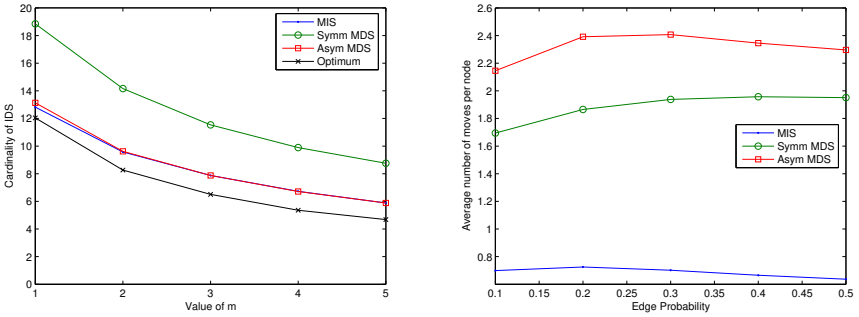


(a) Average size of IDS (b) Average number of moves per process

Fig. 3. Performance in WS graphs (30 vertices)

When p_r increases, the variance of node degree increases. As a result, the size of IDS also increases. We found that Asym MDS performed slightly better than MIS while Symm MDS still performed the worst. In terms of convergence time, MIS performed the best, followed by Symm MDS and then Asym MDS.

The BA model [1] generates a scale-free network that exhibits a power-law distribution of node degrees. It starts with a small number (m_0) of connected vertices. At every round, a new vertex x with m ($m \leq m_0$) incident edges are added to the network. The probability of a new edge connecting x and an existing vertex y is proportional to the node degree of y.



(a) Average size of IDS (b) Average number of moves per process

Fig. 4. Performance in BA graphs (30 vertices)

Figure 4 shows the performance of the proposed approaches with respect to m. As m increases, the average node degree also increases. Consequently, smaller IDS can be identified. In general, the result here is similar to that in Fig. 3. MIS performed nearly the same as Asym MDS in terms of IDS cardinality, but outperformed the counterparts in terms of the time to convergence.

5 Conclusions

We have proposed three self-stabilizing distributed algorithms for minimal IDS in the framework of game theory. The first one based on MIS game is simple and guarantees self-stabilization. The second and the third ones based on MDS game are more complicated and can only guarantee weak stabilization. Simulation results show that the MIS-based approach is not inferior to any others in terms of the cardinality of IDS and outperforms all others in terms of the time to stabilization.

Possible extensions to the proposed approaches include 1) independent multi-dominating sets, where different nodes may demand different degrees of domination, and 2) weighted IDS, where nodes have weights and we want to minimize total weight of the IDS. Anotehr way is to extend the self-stabilizing algorithms to run under distributed and synchronous daemons.

References

1. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science **286**, 509–512 (Oct 1999)
2. Cohen, J., Dasgupta, A., Ghosh, S., Tixeuil, S.: An exercise in selfish stabilization. ACM Trans. on Autonomous and Adaptive Systems **3**(4) (Nov 2008)
3. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Comm. ACM **17**(11), 643–644 (Nov 1974)
4. Dolev, S., Gouda, M.G., Schneider, M.: Memory requirements for silent stabilization. Acta Informatica **36**, 447–462 (1999)
5. Erdös, P., Rényi, A.: On random graphs I. Publications Mathematicae, Debrecen **6**, 290–297 (1959)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
7. Goddard, W., Hededtniemi, S.T., Jacobs, D.P., Srimani, P.K., Xu, Z.: Self-stabilizing graph protocols. Parallel Process. Lett. **18**(1), 189–199 (2008)
8. Goddard, W., Hedetniemi, S.T., Jacobs, D.P., Srimani, P.K.: A self-stabilizing distributed algorithm for minimal total domination in an arbitrary system graph. In: Proc. 17th Int'l Parallel and Distributed Processing Symp. (Apr 2003)
9. Gouda, M.G.: The theory of weak stabilization. In: Datta, A., Herman, T. (eds.) Lecture Notes in Computer Science 2194, pp. 114–123. Springer-Verlag (2001)
10. Hedetniemi, S.M., Hedetniemi, S., Jacobs, D.P., Srimani, P.K.: Self-stabilizing algorithms for minimal dominating sets and maximal independent sets. Computers & Mathematics with Applications **46**(5-6), 805–811 (Sep 2003)
11. Ikeda, M., Kamei, S., Kakugawa, H.: A space-optimal self-stabilizing algorithm for the maximal independent set problem. In: Proc. 3rd Int'l Conf. on Parallel and Distributed Computing, Applications and Technologies (2002)
12. Kakugawa, H., Masuzawa, T.: A self-stabilizing minimal dominating set algorithm with safe convergence. In: Int'l Parallel and Distributed Processing Symposium (Apr 2006)
13. Kshemkalyani, A.D., Singhal, M.: Distributed Computing: Principles, Algorithms, and Systems, p. 634. Cambridge University Press, Cambridge, UK (2008)
14. Shukla, S.K., Rosenkrantz, D.J., Ravi, S.S.: Observations on self-stabilizing graph algorithms for anonymous networks. In: Proc. 2nd Workshop on Self-Stabilizing Systems (1995)
15. Turau, V.: Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler. Inform. Process. Lett. **103**(3), 88–93 (2007)
16. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature **393**, 440–442 (Jun 1998)
17. Xu, Z., Hedetniemi, S.T., Goddard, W., Srimani, P.K.: A synchronous self-stabilizing minimal domination protocol in an arbitrary network graph. In: Lecture Notes in Computer Science 2918, pp. 26–32. Springer-Verlag (2003)
18. Yen, L.H., Chen, Z.L.: Game-theoretic approach to self-stabilizing distributed formation of minimal multi-dominating sets. IEEE Trans. Parallel Distrib. Syst. **25**(12), 3201–3210 (Dec 2014)
19. Yen, L.H., Huang, J.Y., Turau, V.: Designing self-stabilizing systems using game theory. ACM Trans. on Autonomous and Adaptive Systems **11**(3) (Sep 2016)