Selfish Self-Stabilizing Approach to Maximal Independent Sets

Li-Hsing Yen and Jean-Yao Huang
Dept. of Computer Science & Information Engineering
National University of Kaohsiung, Kaohsiung, Taiwan 811, R.O.C.

***Abstract*—Given an undirected graph $G = (V, E)$, $S \subseteq V$ is an independent set if no two nodes in S are adjacent to each other. S is a maximal independent set (MIS) if no proper superset of S is also an independent set. We model the problem of finding an MIS in a distributed system as a noncooperative graphical game and propose an algorithmic mechanism design for the problem. We show Nash equilibrium, correctness, and Pareto optimality of the design and then turn the design into a self-stabilizing algorithm running under a synchronous daemon. The convergence property and time complexity of the algorithm is shown. Simulation results indicate that the proposed protocol performs better than previous work in terms of MIS size under various representative types of network topologies.**

***Keywords*-self-stabilization; independent set; distributed algorithms; game theory**

I. Introduction

We consider a distributed system consisting of several processes. Processes communicate by sharing variables. Each variable is owned by and exclusively updated by one process. Other processes can only read the value of this variable. A snapshot capturing all variable's values comprises a system state. A system state may be legitimate or illegitimate with respect to a predicate that specifies the system's goal. When a system is in a legitimate state, an occurrence of a fault may bring the system into an illegitimate state. A self-stabilizing distributed algorithm tolerates transient faults (faults that do not persist) by always reaching legitimate system states in a finite time regardless of initial system configuration [1].

Game theory provides a mathematical framework for the study of competitions and potential cooperations among participating agents. This study views a self-stabilizing system as a noncooperative game. Processes that have their own behavior are analogous to autonomous agents or players in a game. Processes update their variables as agents change their strategies. A quiescent state in self-stabilizing systems, in which no process will further modify its variables, naturally corresponds to a Nash equilibrium in noncooperative games where no agent has the incentive to change her/his own strategy. However, agents in a game are rational yet selfish. They typically seek their own interests rather than a common good. In contrast, processes in a self-stabilizing system have no local, private objectives. They are coded to achieve a system goal. It is our motivation to achieve a system goal by designing private objectives of individual processes. In other words, we were seeking utility functions for autonomous players whose rational yet selfish interactions with one another dynamically lead the game to a stable state where the system goal is satisfied. This holds regardless of initial game state because system state is arbitrary after a fault.

Game theory has been applied to solve certain instances of combinatorial optimization problems. However, little work has been done toward the application of game theory to the design of self-stabilizing distributed algorithms. Cohen et al. [2] first introduced the notion of selfish stabilization, where processes may seek their private goals while cooperating with one another in achieving a common goal. The private goals of different processes may be conflicting. Under this assumption, the authors devised a stabilizing distributed algorithm that forms a spanning tree. If there are two types of processes that have conflicting private goals, the authors proved that the algorithm is weakly stabilizing [3]. Weak stabilization means that starting from an arbitrary state, there exists *at least one* sequence of state transitions that leads the system to a stable state. In contrast, self-stabilizations typically demands that *any* sequence of state transitions leads the system into a stable state starting from whatever initial state. Gouda [3] shows that weak stabilization of a system implies stabilization of the same system under some reasonable conditions.

Gouda and Acharya [4] assumed that gain functions (i.e., utility functions) of processes are defined only at quiescent states. The action systems of all processes are assumed independent of the gain functions and designed to lead the system into a quiescent state. After entering a quiescent state, processes may cause perturbations as a result of attempting to increase their individual gains. The authors are concerned with whether such behaviors can bring the system into illegitimate states.

Both studies mentioned above argued that individual goals may be an obstacle to system goal. In contrast, Yen and Chen [5] proposed an algorithmic mechanism design that achieves an intended system goal through private goals of processes. The proposed game-theoretic design guarantees the formation of a minimal multi-dominating set [6] starting from an arbitrary initial state. The authors converted the game design into a distributed algorithm which achieves weak stabilization.

This paper considers designing a self-stabilizing algorithm to find a *maximal independent set* (MIS) among all processes in a distributed system. In this problem, a binary relation

on the set of all processes is abstracted by an undirected graph, where nodes represent processes and edges represent the existence of the relation between two relevant processes[1]. In 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. S is an MIS if no proper superset of S is also an independent set.

Shukla et al. [7] proposed the first self-stabilizing algorithm to find an MIS, followed by others [8]–[11]. Refer to [12] for a recent literature survey on self-stabilizing algorithms for MIS. All the existing algorithms are ad hoc designed. To the best knowledge of the authors, the proposed approach is the first game-theoretic protocol design for MIS formation. Game theory helps rigorous proof of desired properties. For example, we can prove that the proposed game-theoretic approach always enters a Nash equilibrium regardless of initial configurations. This property corresponds to the convergence property of self-stabilizing algorithms [1]. We show that the proposed design indeed identifies an MIS in any Nash equilibrium. We also show that any Nash equilibrium in the proposed game model is also Pareto optimal.

We convert the game design into a distributed algorithm expressed in guarded commands [13]. We show the convergence property and time complexity of the algorithm running under a synchronous daemon. A salient feature of our work that sets us apart from previous game-theoretic designs [2], [5] is that the proposed algorithm is self-stabilizing rather than merely weak-stabilizing. We conducted extended simulations to study the average-case performance of the proposed work. The results with a variety of network topologies show that, compared with existing self-stabilizing MIS algorithms, the proposed work provides a significant improvement regarding the size of independent set.

The remainder of this paper is organized as follows: Background information and related work are presented in Section II. In the following section, we present a game model for the MIS problem, prove its properties, and convert it to self-stabilizing algorithms. Section IV studies the performance of the proposed approach through simulations. The simulation results are compared with those of existing solutions. Section V concludes this paper.

II. Background and Related Work

An independent set is a maximum independent set if it has the largest size among all. While finding a maximum independent set for a given graph is known to be NP-complete [14], finding an MIS is not. Conventional approach to the MIS problem is to sequentially or recursively [15] examine each node to determine if it should be in the MIS.

A distributed algorithm is self-stabilizing if it meets *convergence* and closure conditions [1]. The former requires that starting from arbitrary state (possibly illegitimate), the algorithm eventually reaches a legitimate state. The latter property states that any state following a legitimate state is also legitimate. In our problem, a state is legitimate if all nodes that decide to be in the set in the state indeed constitute an MIS.

Self-stabilizing algorithms typically assume that processes communicate with one another via shared variables. In this model, each process owns local variables to which it can exclusively write values. Local variables can be read and can only be read by the owner and the owner's neighboring processes. A distributed algorithm is anonymous if it does not assume the availability of unique process identifiers. The program executed by an individual process is often expressed in the form of guarded commands [13]. Each command consists of a guard (Boolean expression) and an action (assignment statements) separated by '$\rightarrow$'. A command is *enabled* if its guard is true. Only at that time can the corresponding action be executed. The whole system enters a quiescent state (called a fixed point in [4]) if no commands are enabled. The level of execution concurrency among processes is specified by three possible execution models, namely, central, synchronous, and distributed daemons. The central daemon allows only one process to execute at a time, while all processes execute in a lock-step manner in a synchronous daemon. These two daemons are subsumed by the distributed daemon, which allows a subset of all processes to execute in parallel.

Shukla et al. [7] proposed an anonymous self-stabilizing algorithm that finds an MIS under a central daemon. This algorithm consists of two simple rules. First, a node joins the MIS if none of its neighbors is in the MIS. Second, a node leaves the MIS if one of its neighbors is also in the MIS. The same algorithm was also proposed by other researchers [16].

These two rules are general, but the proposed algorithm was designed to run under a central daemon. It does not work under a distributed or synchronous daemon. More specifically, two neighboring nodes could enter the MIS by the first rule in the same round and then leave the MIS by the second rule in the next round. Such scenario causes instability. Ikeda et al. [8] took the first rule but used node identifiers to break node symmetry in the second rule. In their algorithm, a node should leave the MIS only if any of its neighbors that has a smaller identifier is also in the MIS.

Turau [9] proposed a self-stabilizing algorithm that has a lower time complexity than the one proposed by Ikeda et al. This algorithm is also not anonymous and runs under a distributed daemon. Beside identifier, each node maintains a variable that indicates the current status of the node. While values IN and OUT indicate the node is and is not in the MIS, respectively, value WAIT indicates that the node wants to but not yet join the MIS under construction.

Goddard et al. [10] proposed a non-anonymous self-

[1]We thus use *nodes* and *processes* interchangeably in this paper

stabilizing algorithm that identifies an MIS under a synchronous daemon. Each node in this algorithm executes the following two rules. First, a node joins the MIS if it is not currently in the MIS and there is no neighboring node in the MIS that has a larger identifier than it. Second, a node leaves the MIS if there is any neighboring node in the MIS that has a larger identifier than it.

Unlike all approaches mentioned above, Shi et al. [11] proposed a self-stabilizing algorithm that identifies an 1-MIS under a central daemon. An 1-MIS is an MIS with an extra property that any removal of a node from the MIS does not allow an addition of more nodes into the MIS. This algorithm works for tree-structured topology and is anonymous.

Table I summarizes all the related work.

III. The Proposed Approach

A. The MIS Game

A noncooperative game Γ can be defined as $\Gamma = [\{p_i\}_{i=1}^n; \{S_i\}_{i=1}^n; \{u_i\}_{i=1}^n]$, where $P = \{p_i\}_{i=1}^n$ is the set of players (agents), S_i is player p_i's *strategy set* indicating p_i's available choices, and u_i is p_i's utility function. In our problem, P is the set of all processes in a distributed system. We define $S_i = \{\text{IN}, \text{OUT}\}$ indicating whether player p_i chooses to be in the MIS or not.

Our problem considers an undirected graph $G = (V, E)$, where $V = P$ and $E \subseteq P \times P$. The pair (G, Γ) formulates a graphical game [17]. We refer to this game as the *MIS game*. We assume that the MIS game is a dynamic game, where no two or more players act simultaneously. Sec. III-B will relax this assumption and discuss how to deal with simultaneous actions when the game design is turned into a self-stabilizing algorithm running under a synchronous or distributed daemon.

Let $N_i = \{p_j | (p_i, p_j) \in E\}$ be the set of p_i's neighboring nodes in G. For each player p_i, let $M_i = N_i \cap \{p_j | \deg(p_j) \leq \deg(p_i)\}$. A *strategy profile* is an n-tuple $C = (c_1, c_2, \ldots, c_n)$, where $c_i \in S_i$ represents player p_i's strategy. We sometimes express C as (c_i, c_{-i}), where c_{-i} indicates a tuple of all player's strategies other than p_i's. Given a strategy profile $C = (c_i, c_{-i})$, we define the utility of p_i associated with C as

$$u_i(C) = \begin{cases} 0 & \text{if } c_i = \text{OUT} \\ -\alpha & \text{if } c_i = \text{IN} \wedge \exists p_j \in M_i : c_j = \text{IN} \\ \alpha & \text{otherwise,} \end{cases} \quad (1)$$

where $\alpha > 0$ is a constant. The initial (tentative) strategy of each player is arbitrary, resembling an arbitrary system state after a fault. Players can change their strategies if necessary, following the so-called *best-response rule*. This rule states that player p_i selects strategy c_i* only if

$$c_i* = \arg\max_{c_i \in S_i} u_i(c_i, c_{-i}). \quad (2)$$

The sequence of players changing their strategies is arbitrary. When choosing strategies, players are aware of other player's choices.

Consider two neighboring nodes p_i and p_j that are initially OUT and compete in joining the MIS. Either one can be IN earlier than the other because the sequence of strategy changes is non-deterministic. This action does not prevent the other from also joining the MIS later, as long as doing so is the other's best response. As a reaction, the one that acts first may change its strategy to OUT to avoid a negative utility. It is our major concern whether such a chain reaction leads to instability of the MIS game, i.e., a situation where some players repeatedly changing their strategies.

The stability of a game is characterized by the notion of Nash equilibrium. When a game enters a Nash equilibrium, no player can further increase its utility by unilaterally deviating from its current strategy.

Definition 1 (Nash equilibrium): Given a game $\Gamma = [P; \{S_i\}_{i=1}^n; \{u_i\}_{i=1}^n]$, a strategy profile $C^* = (c_1^*, c_2^*, \ldots, c_n^*)$ is a Nash equilibrium if $\forall i \in \{1, 2, \ldots, n\} : \forall c_i \in S_i :: u_i(c_i^*, C_{-i}^*) \geq u_i(c_i, C_{-i}^*)$.

To ease our discussion, we decompose the stability of a game into a function of the stabilities of individual players.

Definition 2: During a game play, a player is *stable* if it changes its strategy a finite number of times.

Because only one player is allowed to change her/his strategy at one time, the best-response rule implies the following two properties of the MIS game.

Property 1: If p_i sets c_i to IN at some time t other than the initial time, then either $M_i = \emptyset$ or $\forall p_j \in M_i : c_j = \text{OUT}$ at t.

Property 2: If p_i sets c_i to OUT at some time t other than the initial time, then $\exists p_j \in M_i : c_j = \text{IN}$ at t.

These two properties are needed in the proof of Lemma 1.

Lemma 1: Define $D_k = \{p_i | \deg(p_i) = k\}$ and $U_k = \{p_i | \deg(p_i) \leq k\}$. In any game play, if $D_{k+1} \neq \emptyset$ and either $U_k = \emptyset$ or all players in U_k are stable, where $k \geq 0$, then all players in D_{k+1} are stable as well.

Proof: By way of contradiction, assume that either $U_k = \emptyset$ or all players in U_k are stable but some player $p_i \in D_{k+1}$ is not, i.e., p_i constantly changes its strategy. Note that $M_i \neq \emptyset$ since otherwise p_i can be stable by setting $c_i = \text{IN}$. Consider any time point t after which no player in U_k changes its strategy (t is simply set to 0 if $U_k = \emptyset$). The condition that p_i constantly changes its strategy implies that p_i changes c_i from IN to OUT for infinitely many times. Suppose that p_i sets c_i to IN at some time $t_1 > t$ and changes c_i back to OUT at some later time $t_2 > t_1$. By Property 1, the former action implies that $\forall p_j \in M_i : c_j = \text{OUT}$ at t_1. By Property 2, the latter action implies that $\exists p_j \in M_i : c_j = \text{IN}$ at t_2. It turns out that there must exist some $p_j \in M_i$ that changes c_j from OUT to IN at some time $t_3 \in (t_1, t_2)$. Refer to Fig. 1 for an illustration. It is impossible that $p_j \in D_{k+1}$, because in

Table I
EXISTING SELF-STABILIZING MIS ALGORITHMS

Work	Objective	Control daemon	Topology	Anonymous?
Shukla et al. [7]	MIS	Centralized	General	Yes
Ikeda et al. [8]	MIS	Distributed	General	No
Turau [9]	MIS	Distributed	General	No
Goddard et al. [10]	MIS	Synchronous	General	No
Shi et al. [11]	1-MIS	Centralized	Tree	Yes
This work	MIS	Synchronous	General	No

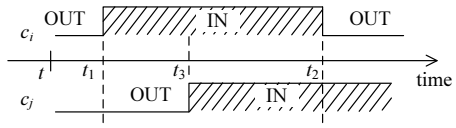

Figure 1. Interaction between neighboring players p_i and p_j

that case changing c_j from OUT to IN at t_3 only causes a negative utility gain for p_j (as $c_i =$ IN at t_3). Therefore, p_j must be in $M_i \setminus D_{k+1} = N_i \cap U_k$, which contradicts with the assumption that either $U_k = \emptyset$ or no player in U_k changes its strategy after t. ■

Lemma 1 indicates that, as long as all players with lower node degrees are stable, race conditions between any two neighboring players that are of the same degree do not cause instability. When both are eligible to join the MIS set, the one that acts first does not revise her decision, preventing the other from also joining the set. Because no two players can act at the same time, this arbitration rule guarantees stability.

If all players are stable, then eventually no player further changes its strategy. This corresponds to a Nash equilibrium. We therefore have the following result.

Theorem 1: The MIS game (G, Γ) eventually reaches a Nash equilibrium regardless of initial game configurations.

Proof: We prove the theorem by showing that all players are stable in any game play. Let c be the minimal degree in G. If $c = 0$, any node in U_0 is isolated and can be stable by choosing the dominating strategy IN. If $c > 0$, then $U_{c-1} = \emptyset$. So all players in D_c and thus U_c are stable by Lemma 1. For any positive integer $k > c$, assume that all players in U_{k-1} are stable. Lemma 1 shows that all players in D_k and thus $U_k = U_{k-1} \cup D_k$ are stable as well. By induction on k, we can show that all players are stable. ■

Theorem 1 proves the stability of the MIS game. The following theorem shows that when the MIS game ends up entering a Nash equilibrium, the set $S = \{p_i | c_i =$ IN$\}$ is indeed an independent set.

Theorem 2: When the MIS game (G, Γ) ends up in a Nash equilibrium, the set $S = \{p_i | c_i =$ IN$\}$ is an independent set in G.

Proof: By way of contradiction, assume that $S = \{p_i | c_i =$ IN$\}$ at a Nash equilibrium is not an independent set in $G = (V, E)$. It is implied that there exists $(p_i, p_j) \in E$ such that $p_i \in S$ and $p_j \in S$. In that case, p_i or p_j (or both, if $\deg(p_i) = \deg(p_j)$) has a negative utility value $(-\alpha)$. Either player can have a utility gain if it changes its strategy to OUT, contradicting with the assumption that the game has reached a Nash equilibrium. So S must be an independent set. ■

We can prove further that the independent set at a Nash equilibrium is maximal by showing that the Nash equilibrium is Pareto optimal.

Definition 3 (Pareto optimal): A strategy profile $C = (c_1, c_2, \ldots, c_n)$ is Pareto optimal if and only if there exists no other strategy profile $C' = (c'_1, c'_2, \ldots, c'_n)$ such that $\forall i \in \{1, 2, \ldots, n\} : u_i(C') \geq u_i(C)$ and $\exists j \in \{1, 2, \ldots, n\} : u_j(C') > u_j(C)$.

To see why, first note that the utility of any player at a Nash equilibrium is either 0 or α. Because any player with utility value $-\alpha$ can raise its utility to 0 by switching its strategy from IN to OUT, no players can have utility value $-\alpha$ at a Nash equilibrium. Furthermore, all players having utility value α cannot further increase their utilities. Therefore, only players with utility value 0 (i.e., players choosing OUT) can possibly increase their utilities. If a Nash equilibrium is Pareto optimal, which implies that no players with utility value 0 can increase their utilities without decreasing the utility of any others, we cannot add any node to the independent set S without removing any others from S. Therefore, there exists no superset of S that is also an independent set. On the other hand, if S at a Nash equilibrium is an MIS, then adding any player $p_j \notin S$ into S does not increase p_j's utility because some neighbor of p_j must be in S. Since all players in S already have the maximal utility value, the Nash equilibrium is Pareto optimal.

Theorem 3: Every Nash equilibrium of the MIS game is Pareto optimal.

Proof: We prove the theorem by showing that no players with utility value 0 at a Nash equilibrium can increase their utilities without decreasing the utility of any others. By (1), any player p_i having utility value 0 at a Nash equilibrium implies that $c_i =$ OUT and $\exists p_j \in M_i : c_j =$ IN. Note that p_j's utility must be α. To increase p_i's utility, c_i must be changed to IN while c_j must be changed to OUT. This means that p_j's utility must be degraded. Therefore, it is impossible to increase p_i's utility without decreasing p_j's,

which means that every game result is Pareto optimal. ■

Although the result in any Nash equilibrium is an MIS, it is not necessarily a maximum independent set.

B. Self-Stabilizing MIS Algorithm

When converting the MIS game into a self-stabilizing algorithm, player's strategies in the game correspond to local states of processes. We assume that each process p_i uses a local variable c_i to keep p_i's current strategy. Let A be the converted algorithm. A consists of two guarded commands at each process p_i as shown below.

R1 $c_i \neq \text{OUT} \wedge \exists p_j \in M_i : c_j = \text{IN}$
$\rightarrow c_i := \text{OUT}$

R2 $c_i \neq \text{IN} \wedge (M_i = \emptyset \vee \forall p_j \in M_i : c_j = \text{OUT})$
$\rightarrow c_i := \text{IN}$

R1 and R2 correspond to Properties 2 and 1, respectively. Each process p_i needs the information of M_i, which includes the set of neighbors of p_i and the degree of each neighbor. We may assume that N_i is fixed and let each process p_i maintain another shared variable d_i that represents p_i's degree. Then we should have another guarded command that corrects the value of d_i after a transient fault. To ease our discussion, we simply assume that the graph information (i.e., N_i and d_i) is preserved when transient failures occur.

The proposed MIS game is a dynamic game, which disallows simultaneous decision makings. This property ensures the correctness of A running under a central daemon. However, it requires special treatments to turn a dynamic game into a self-stabilizing algorithm that runs under a synchronous or distributed daemon.

The execution time of a distributed algorithm under a synchronous or distributed daemon can be divided into a series of *rounds*. In each round, one or more processes act in parallel. The action of each process in a round depends on the status of its neighboring processes in the previous round.

In case of A, allowing two neighboring processes that are of the same degree to act simultaneously may cause ping-pong effects and thus instability. To illustrate, let p_i and p_j be two neighboring processes such that $\deg(p_i) = \deg(p_j)$. Suppose that all rounds are sequentially numbered as $T_1, T_2, \ldots$. If p_i and p_j both have R2 enabled in round T_k, they will join the set in round T_{k+1}. Then, because $p_i \in M_j$ and $p_j \in M_i$, p_i and p_j will have R1 enabled and consequently leave the set in round T_{k+2}. See Fig. 2. This possibility can lead to a strategy change cycle that lasts forever.

As a remedy, we use process identifiers to break the symmetry that occurs to neighboring processes of the same degree. Suppose that every node has a unique identifier. We define *precedence relation* between nodes as follows.

Definition 4: The precedence relation ($\prec$). Let *id*(u) be the identifier of node u. For any two nodes p_i and p_j, $p_i \prec p_j$ if and only if (1) $\deg(p_i) < \deg(p_j)$ or (2) $\deg(p_i) = \deg(p_j)$ and *id*$(p_i) <$ *id*(p_j).

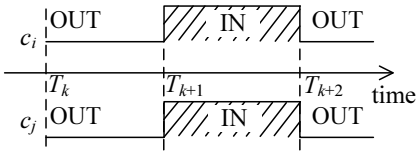

Figure 2. Simultaneous moves of two neighboring players p_i and p_j

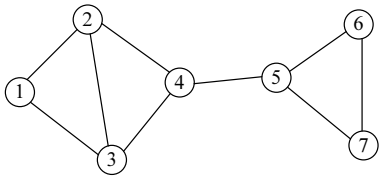

Figure 3. A sample topology

Note that $\prec$ is irreflexive, antisymmetric and transitive. Define $\mathcal{M}_i$ as $\mathcal{M}_i = N_i \cap \{p_j | p_j \prec p_i\}$. The modified algorithm A' consists of two guarded commands as shown below.

R1 $c_i \neq \text{OUT} \wedge \exists p_j \in \mathcal{M}_i : c_j = \text{IN}$
$\rightarrow c_i := \text{OUT}$

R2 $c_i \neq \text{IN} \wedge (\mathcal{M}_i = \emptyset \vee \forall p_j \in \mathcal{M}_i : c_j = \text{OUT})$
$\rightarrow c_i := \text{IN}$

For the topology shown in Fig. 3, Table II shows a possible state transition sequence of A'. Here *id*$(p_i) = i$ for all i, $1 \leq i \leq 7$.

C. Stability and Time Complexity Analyses

We shall now show self-stability and time complexity of A'. We first define stabilities of processes under a synchronous or distributed daemon.

Definition 5: When running a distributed self-stabilizing algorithm under a synchronous or distributed daemon, process p_i is *stable* in round T_k if none of p_i's guarded commands are enabled after T_k. We say that p_i *becomes* stable in round T_k if it is stable in T_k but not in round T_{k-1}.

The stability of nodes relies on precedence relation between nodes. A *precedence graph* is a directed graph $G_p(V, L)$ constructed from a given undirected graph $G(V, E)$ such that G and G_p share the same vertex set V

Table II
A POSSIBLE STATE TRANSITION SEQUENCE

Round	c_1	c_2	c_3	c_4	c_5	c_6	c_7
0	OUT	IN	IN	OUT	IN	OUT	OUT
1	IN	IN	OUT	OUT	IN	IN	IN
2	IN	OUT	OUT	OUT	OUT	IN	OUT
3	IN	OUT	OUT	IN	OUT	IN	OUT

and for every $(p_i, p_j) \in E$, $(p_i, p_j) \in L$ iff $p_i \prec p_j$. Because $\prec$ is irreflexive and transitive, G_p must not contain directed cycles. That is, G_p is an acyclic orientation of G.

Theorem 4: Let P be a finite set of nodes. Let $R_i \subseteq P$ be the set of nodes that become stable in the i-th round when running Algorithm A' under a synchronous daemon. Suppose that no nodes are stable initially. Then, the following two condition hold.

1) $R_1 \neq \emptyset$.
2) If $R_i \neq \emptyset$ and $\cup_{1 \leq j \leq i} R_j \neq P$, then $R_{i+1} \neq \emptyset$.

Proof: Let D_i^+ denote the set of all nodes having indegree 0 in G_p. Because G_p is acyclic, $D_0^+ \neq \emptyset$. In the first round of A', all nodes in D_0^+ will choose IN by R2 and become stable. Thus $R_1 \neq \emptyset$. Suppose that $R_i \neq \emptyset$ and $\cup_{1 \leq j \leq i} R_j \neq P$ for some i. Assume that $R_{i+1} = \emptyset$, which means that no node in $P \setminus \{R_1 \cup R_2 \cup \ldots \cup R_i\}$ becomes stable in the $(i+1)$-th round. Consider any node $p_j \in P \setminus \{R_1 \cup R_2 \cup \ldots \cup R_i\}$. If $\mathcal{M}_j \subseteq \{R_1 \cup R_2 \cup \ldots \cup R_i\}$, either R1 or R2 of p_j is enabled and p_j can become stable in the $(i+1)$-th round. Therefore, there must be some $p_k \in \mathcal{M}_j$ such that $p_k \in P \setminus \{R_1 \cup R_2 \cup \ldots \cup R_i\}$. By the same argument, we can show that there must be some node $p_l \in \mathcal{M}_k$ such that $p_l \in P \setminus \{R_1 \cup R_2 \cup \ldots \cup R_i\}$, and so forth. It turns out that every node in $P \setminus \{R_1 \cup R_2 \cup \ldots \cup R_i\}$ must have a predecessor in G_p that is also in $P \setminus \{R_1 \cup R_2 \cup \ldots \cup R_i\}$. Because $P \setminus \cup_{1 \leq j \leq i} R_j$ is finite, there must be at least one cycle in G_p, which contradicts the property that G_p is acyclic. We thus have the proof. ■

Theorem 4 indicates that n nodes will become stable within n rounds when running A' under a synchronous daemon, so the time complexity of A' is $O(n)$. Consider running A' on a complete graph consisting of n nodes. The node that has the smallest identifier will join the MIS in the first round. All other nodes will be out of the set in the second round. This condition is valid regardless of n, which represents a best-case execution time.[2]

As an example of the worst-case execution, consider running A' on a ring graph. Suppose that all n nodes are consecutively assigned identifier $0, 1, \ldots, n-1$ along the ring. The corresponding digraph will be like that shown in Fig. 4. Suppose that all nodes are OUT initially. In the first round, all nodes will become IN. Only node 0 will become stable in this round. In the second round, nodes $1, 2, \ldots, n-1$ will change to OUT, and nodes 1 and $n-1$ will become stable. In the third round, nodes $2, 3, \ldots, n-2$ will change back to IN, and only node 2 will become stable. In general, node k, $0 \leq k < n-1$ will become stable in Round $k+1$. The whole execution takes $n-1$ rounds.

[2]A similar scenario also occurs to star topology, where all leaf nodes will join the set in the first round and the hub node will leave the set in the second round.

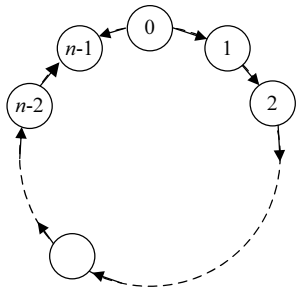

Figure 4. The precedence graph corresponding to a ring with monotonically increasing identifiers

IV. Simulation Results

We conducted simulations for performance comparisons between the proposed algorithm and several existing approaches. The approaches under consideration include those proposed by Ikeda et al. [8], Goddard et al. [10] and Turau [9]. These approaches all use node identifiers and run under a synchronous or distributed daemon. Our major concern is the size of MIS. A larger MIS indicates a better result.

Self-stabilizing algorithms ought to tolerate transient faults. After an arbitrary transient fault, the system state becomes unpredictable. To simulate this scenario, all process variables were assigned randomly-determined values initially. Process identifiers were randomly yet uniquely set. Each result was averaged over 1000 trials.

The size of MIS depends on network topology. For a fair comparison, we tested four representative types of network topologies. These types are Unit Disk Graph (UDG) [18], ER model [19], WS model [20], and BA model [21].

In a UDG, nodes are characterized by their locations and communication ranges. A link exists between two nodes if and only if these two nodes are within the communication rage of each other. We randomly deployed 50 to 100 nodes in a 1000×1000 m^2 region. Each node has a communication range of 200 m. Fig. 5 shows average MIS size with respect to the number of nodes. Clearly, the MIS size generally increases with increasing number of nodes. The proposed algorithm achieves the best result among all.

In ER model, whether an edge exists between any two nodes is determined by an edge probability p_e. The resulting topology is a random graph. Fig. 6 shows how average MIS size changes with p_e in random graphs with 100 nodes. As p_e changes from 0.1 to 0.5, the average MIS size decreases. The reason is that the average node degree increases with an increasing p_e. Here the proposed algorithm still performs the best.

In WS model, a regular graph is first formed, where each node has $2k$ edges connecting to its $2k$ nearest neighbors. Then, for each node, we rewire every edge of this node to a randomly selected node with probability p_r. We assumed a graph of 100 nodes and $k = 2$. Fig. 7 shows the result of

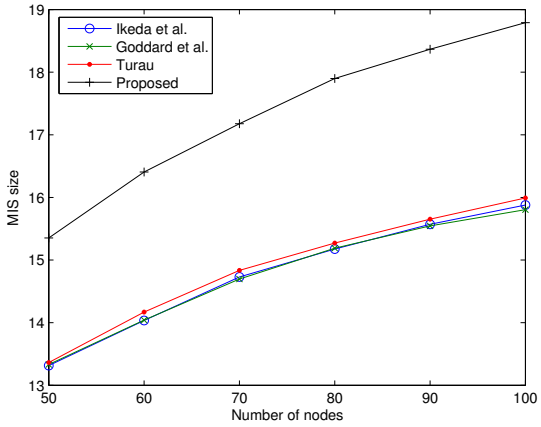

Figure 5. Average MIS size in UDG

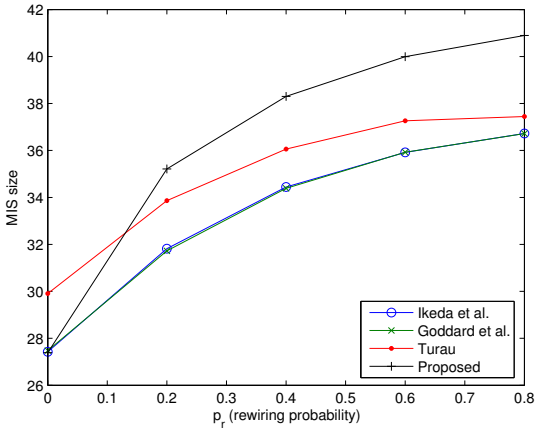

Figure 7. Average MIS size in WS model

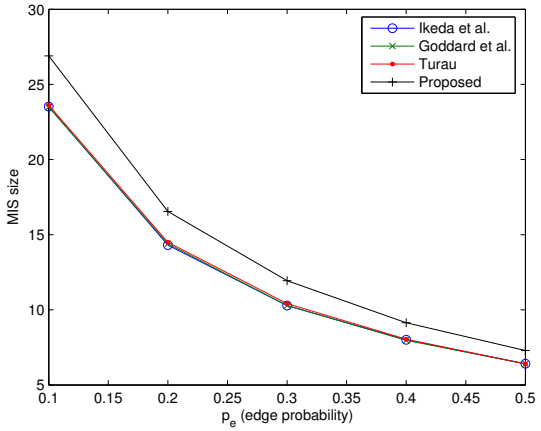

Figure 6. Average MIS size in ER model

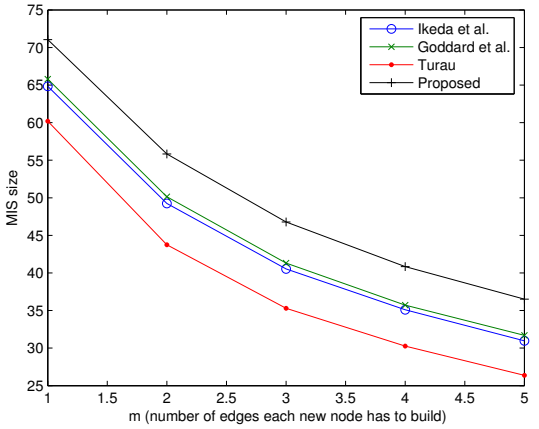

Figure 8. Average MIS size in BA model

average MIS size with various settings of p_r. When $p_r = 0$, the proposed approach yielded smaller MIS than the work by Turau [9]. However, as p_r increases, the proposed approach retakes its first place.

In BA model, the graph has n_0 nodes initially. Other nodes are incrementally added into the graph. When adding a node into the graph, we build m edges that connect this node to m nodes already in the graph. The other end of each edge is randomly determined. The probability that a new edge connects to node u is proportional to the degree of u. We tested a 100-node graph with $n_0 = 5$. Fig. 8 shows average MIS sizes with respect to m. A larger m generally implies a higher probability of a small set of nodes dominating all other nodes and thus a smaller MIS. Therefore, the performance of all methods degrades as m increases. Even though, the proposed algorithm still performs the best.

All the results can be explained by the distribution of node degrees in the tested network topologies. Node degree in a random graph or UDG has a binomial distribution [22], [23]. The BA model creates a network topology for which the distribution of node degrees follows a power law. Therefore, node degrees differ in these settings, which explains the superiority of the proposed algorithm in MIS size over the counterparts because the proposed algorithm prefers small-degree nodes in forming an MIS. However, each node has uniformly $2k$ neighbors in regular graphs ($p_r = 0$ in WS model). In this case, the preference for small-degree nodes in the formation of an MIS does not really affect the result. Consequently, other design considerations may be critical.

V. Conclusions

We have proposed the MIS game, a noncooperative graphical game design for identifying MIS in an n-node graph. The MIS game eventually enters a Nash equilibrium regardless of initial game configuration, and the game result is correct yet Pareto optimal. The MIS game have been converted into a self-stabilizing distributed algorithm running under a synchronous daemon. We have proved that the

time complexity of the algorithm is $O(n)$. The performance of the algorithm has been studied through simulations with four types of representative network topologies. The simulation results indicate that the proposed game-theoretic self-stabilizing algorithm generally outperforms other existing approaches in terms of MIS size.

ACKNOWLEDGEMENTS

The authors are grateful to anonymous reviewers for their constructive and helpful comments on the earlier version of the paper. This work has been supported by Ministry of Science and Technology, Taiwan, under contract MOST 103-2221-E-390-002.

REFERENCES

[1] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Comm. ACM*, vol. 17, no. 11, pp. 643–644, Nov. 1974.

[2] J. Cohen, A. Dasgupta, S. Ghosh, and S. Tixeuil, "An exercise in selfish stabilization," *ACM Trans. on Autonomous and Adaptive Systems*, vol. 3, no. 4, Nov. 2008.

[3] M. G. Gouda, "The theory of weak stabilization," in *Lecture Notes in Computer Science 2194*, A. Datta and T. Herman, Eds. Springer-Verlag, 2001, pp. 114–123.

[4] M. G. Gouda and H. B. Acharya, "Nash equilibria in stabilizing systems," *Theoretical Computer Science*, vol. 412, pp. 4325–4335, 2011.

[5] L.-H. Yen and Z.-L. Chen, "Game-theoretic approach to self-stabilizing distributed formation of minimal multi-dominating sets," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3201–3210, Dec. 2014.

[6] L. Jia, R. Rajaraman, and T. Suel, "An efficient distributed algorithm for constructing small dominating sets," *Distributed Computing*, vol. 15, no. 4, pp. 193–205, 2002.

[7] S. K. Shukla, D. J. Rosenkrantz, and S. S. Ravi, "Observations on self-stabilizing graph algorithms for anonymous networks," in *Proc. 2nd Workshop on Self-Stabilizing Systems*, 1995.

[8] M. Ikeda, S. Kamei, and H. Kakugawa, "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.

[9] V. Turau, "Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler," *Inform. Process. Lett.*, vol. 103, no. 3, pp. 88–93, 2007.

[10] 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 *Proc. 17th Int'l Parallel and Distributed Processing Symp.*, Apr. 2003.

[11] Z. Shi, W. Goddard, and S. T. Hedetniemi, "An anonymous self-stabilizing algorithm for 1-maximal independent set in trees," *Inform. Process. Lett.*, vol. 91, no. 2, pp. 77–83, 2004.

[12] 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, pp. 406–415, 2010.

[13] E. W. Dijkstra, "Guarded commands, nondeterminacy, and formal derivation of programs," *Comm. ACM*, vol. 18, no. 8, pp. 453–457, Aug. 1975.

[14] R. E. Tarjan and A. E. Trojanowski, "Finding a maximum independent set," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 537–546, 1977.

[15] M. M. Halldórsson and J. Radhakrishnan, "Greed is good: Approximating independent sets in sparse and bounded-degree graphs," *Algorithmica*, vol. 18, no. 1, pp. 145–163, 1997.

[16] S. M. Hedetniemi, S. Hedetniemi, D. P. Jacobs, and P. K. Srimani, "Self-stabilizing algorithms for minimal dominating sets and maximal independent sets," *Computers & Mathematics with Applications*, vol. 46, no. 5-6, pp. 805–811, Sep. 2003.

[17] M. J. Kearns, M. L. Littman, and S. P. Singh, "Graphical models for game theory," in *Proc. 17th Conf. in Uncertainty in Artificial Intelligence*, 2001, pp. 253–260.

[18] B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit disk graphs," *Discrete Mathematics*, vol. 86, no. 1-3, pp. 165–177, Dec. 1990.

[19] P. Erdös and A. Rényi, "On random graphs I," *Publications Mathematicae, Debrecen*, vol. 6, pp. 290–297, 1959.

[20] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, pp. 440–442, Jun. 1998.

[21] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, Oct. 1999.

[22] B. Bollobás, *Modern Graph Theory*. Springer-Verlag New York, 1998.

[23] R. Hekmat and P. Van Meighem, "Degree distribution and hopcount in wireless ad-hoc networks," in *Proc. 11th IEEE Int'l Conf. on Networks*, Sep. 2003, pp. 603–609.