Performance Comparisons of Self-stabilizing Algorithms for Maximal Independent Sets

Barton F. Cone*, Stephen T. Hedetniemi†, Lance C. Ingle‡, Ken Kennedy*

*Fulfilld.io
{bart, ken}@fulfilld.io
†School of Computing, Clemson University
hedet@clemson.edu
‡GTRI, Georgia Tech
lanceingle@gtri.gatech.edu

Abstract—Sensor networks, such as ultra-wideband sensors for the smart warehouse, may need to run distributed algorithms for automatically determining a topological layout. In this paper, we present 5 different self-stabilizing algorithms (their central and distributed counterparts) for determining maximal independent sets. The performance of the algorithms, in terms of time complexity, simulation analysis, and size of maximal independent sets found are then compared.

I. INTRODUCTION

Maximal independent sets (MIS) have several uses, one of which is in wireless networks [11], [13]. Wireless and sensor networks are ideal testbeds for self-stabilizing algorithms, and this corresponds with work that Fulfilld.io is doing with ultra-wideband (UWB) sensors for the smart warehouse. The UWB sensors are positioned such that objects can be tracked with near realtime precision allowing a variety of algorithms to be performed (path planning, route optimization, bin optimization, ...).

In this paper, we compare a variety of self-stabilizing algorithms that have been designed to find independent sets, or maximal independent sets, of nodes in distributed systems. The topology of a distributed system is modeled by an undirected graph \( G = (V, E) \) of order \( n = |V| \) of nodes, and size \( m = |E| \) of edges, where nodes represent processes and edges represent communication links between processes.

The open neighborhood of a node \( i \in V \) is the set \( N(i) = \{j | ij \in E\} \) of nodes that are adjacent to \( i \); these nodes are called the neighbors of node \( i \). The closed neighborhood of a node \( i \) is the set \( N[i] = N(i) \cup \{i\} \). The degree of a node is \( \deg(i) = |N(i)| \), and the maximum degree of a node is denoted \( \Delta(G) \).

The processes of a self-stabilizing algorithm consist of a set of variables associated with a node and a set of rules, each of which consists of a guard and a statement. In the distance-1 model, a guard is a Boolean expression over the variables in the closed neighborhood \( N[i] \) of a node \( i \). If the Boolean expression evaluates to true at a node \( i \), then node \( i \) is said to be eligible to make a move, which consists of changing the values of the node’s variables as specified by the associated statement.

The values of a node’s variables at any point define its local state. The local states of the \( n \) nodes define the global state or configuration of the system. An algorithm is called self-stabilizing if regardless of a current, initial configuration, the system is guaranteed to reach a legitimate or desired global state in a finite amount of time without any external intervention, and to remain in a legitimate state [12].

A set \( S \subseteq V \) of nodes is called independent if no two nodes in \( S \) are adjacent. In the graph theory literature, \( i(G) \) denotes the minimum cardinality of a maximal independent set, and \( \beta_0(G) \) denotes the maximum cardinality of an independent set in \( G \).

A set \( S \) of nodes is called a dominating set if every node in \( V \setminus S \) is adjacent to at least one node in \( S \). The domination number of a graph \( G \) equals the minimum cardinality of a dominating set in \( G \), while the upper domination number equals the maximum cardinality of a minimal dominating set in \( G \). The following inequality chain is well-known in domination theory [5].

\[
\gamma \leq i \leq \beta_0 \leq \Gamma
\]

We focus on comparing the performance of several self-stabilizing graph algorithms that have been published for finding maximal independent sets. In particular, we will simulate each algorithm in order to see which algorithms find either smaller or larger maximal independent sets. We are also interested in the general question of whether it is possible to modify these algorithms so that they find maximal independent sets whose cardinalities are better approximations to either \( i(G) \) or \( \beta_0(G) \). It should also be pointed out that maximal independent sets are also minimal dominating sets. Therefore, these algorithms can be used to find minimal dominating sets whose cardinalities may approximate either \( \gamma(G) \) or \( \Gamma(G) \).

II. CENTRAL ALGORITHMS

In this section, we present five self-stabilizing algorithms for finding independent sets that utilize the distance-1 model with an unfair central scheduler, whereby only one eligible node \( x \) is permitted to move at a time. All of the algorithms are silent, in that after stabilization, no node will make another move.
Since Algorithm C2 is a modification of Algorithm C1, and Algorithms C3 and C4 are further modifications of Algorithm C2, proofs of correctness and complexity are provided where relevant. Complexity results are stated for the other algorithms; the reader is referred to the original papers for their proofs.

A. Central Scheduler Algorithm 1

Algorithm C1, due to Shukla, Rosenkrantz and Ravi [11], was the first self-stabilizing algorithm for finding a maximal independent set. Each node $i$ maintains only one Boolean variable $x(i)$. When the algorithm stabilizes, the set of nodes $X = \{i \mid x(i) = 1\}$ defines a maximal independent set.

Algorithm 1 C1

$\text{RIn: if } x(i) = 0 \land (\exists j \in N(i))(x(j) = 1)$
then $x(i) = 1$

$\text{ROut: if } x(i) = 1 \land (\exists j \in N(i))(x(j) = 1)$
then $x(i) = 0$

Theorem 1: Algorithm C1 finds a maximal independent set in at most 2n moves.

Proof: See proof of Algorithm 3.1 in [6]. □

B. Central Scheduler Algorithm 2

Algorithm C2 is a modification of Algorithm C1 in which the degrees of the nodes are taken into account. In an attempt to find a larger maximal independent set, the comparison $\text{deg}(i) \geq \text{deg}(j)$ is used. We will refer to this version of the algorithm as $C2_{\beta_0}$. In order to find a smaller maximal independent set, this can be changed to $\text{deg}(i) \leq \text{deg}(j)$, which will be referred to as $C2_{\beta_1}$. The algorithm and proof of correctness are given below.

Algorithm 2 C2

$\text{RIn: if } x(i) = 0 \land (\exists j \in N(i))(x(j) = 1)$
then $x(i) = 1$

$\text{ROut: if } x(i) = 1 \land (\exists j \in N(i))(x(j) = 1 \land \text{deg}(i) \geq \text{deg}(j))$
then $x(i) = 0$

Lemma 1: When the system is stable, the set $X$ of nodes $i$ for which $x(i) = 1$ forms an independent set.

Proof: Assume that Algorithm C2 has stabilized, and suppose that the set $X$ is not independent. Then there are two adjacent nodes $i$ and $j$ for which $x(i) = x(j) = 1$. If the degree of $i$ is greater than or equal to the degree of $j$, then node $i$ is eligible to execute $\text{ROut}$. Otherwise, the degree of $j$ is greater than the degree of $i$, in which case node $j$ is eligible to execute rule $\text{ROut}$. This contradicts the assumption that the system is stable. Therefore, $X$ is an independent set. □

Lemma 2: When the system is stable, the set $X$ forms a maximal independent set.

Proof: From Lemma 1 we know that when the system is stable, $X$ is an independent set. Suppose that $X$ is not a maximal independent set. In that case, there must be a node $i$ not in $X$, none of whose neighbors are in $X$. In this case, node $i$ is eligible to execute rule $\text{RIn}$. This contradicts the assumption that the system is stable. Thus, when the system is stable, the set $X$ is a maximal independent set. □

Lemma 3: A node $i$ that executes $\text{RIn}$ will never make another move.

Proof: Suppose there is a node $i$ that executes $\text{RIn}$. By definition, this means that after node $i$ executes $\text{RIn}$, every node $j$ adjacent to $i$ has $x(j) = 0$. But since $x(i) = 1$, no node adjacent to $i$ can ever execute $\text{RIn}$. This, in turn, implies that node $i$ can never become eligible to execute $\text{ROut}$. Thus, node $i$ will never make another move. □

Lemma 4: A node $i$ that executes $\text{ROut}$ can make at most one other move.

Proof: Suppose a node $i$ executes $\text{ROut}$. Having executed this rule, the only rule that $i$ can execute is $\text{RIn}$. But from Lemma 3, it follows that if $i$ executes this rule, it will never make another move. □

Theorem 2: Algorithm C2 finds a maximal independent set in at most 2n moves.

Proof: See Lemma 2 for correctness and Lemma 4 for complexity. □

C. Central Scheduler Algorithm 3

Algorithm C3 is a modification of Algorithm C2. As with Algorithm C2, $C3_{\beta_0}$ and $C3_{\beta_1}$ have the comparison signs reversed in order to find a larger and smaller maximal independent set. The idea of Algorithm C3 is that a very weak vertex, one with a smaller degree than all of its neighbors, can always enter, or be excluded from, the maximal independent set. This is handled by a new rule $\text{RInVW}$. The performance comparison data in §IV shows that this does produce larger and smaller sets. The algorithm and proof of correctness are given below.

Algorithm 3 C3

$\text{RIn: if } x(i) = 0 \land (\exists j \in N(i))(x(j) = 1)$
then $x(i) = 1$

$\text{RlnVW: if } x(i) = 0 \land (\exists j \in N(i))(\text{deg}(i) \geq \text{deg}(j))$
then $x(i) = 1$

$\text{ROut: if } x(i) = 1 \land (\exists j \in N(i))(x(j) = 1 \land \text{deg}(i) \geq \text{deg}(j))$
then $x(i) = 0$

Lemma 5: When the system is stable, the set $X$ forms a maximal independent set.

Proof: With a slight modification, this can be seen from Lemma 2. □

Lemma 6: A node $i$ that executes $\text{RInVW}$ will never make another move.

Proof: Suppose a node $i$ does make another move after executing $\text{RInVW}$. In this case, the next move must be $\text{ROut}$ and $i$ must have a neighbor $j \in X$. However, for $i$ to have executed $\text{RInVW}$, it must have a smaller degree than all of its neighbors. This violates the condition of $\text{ROut}$ that a neighbor in $X$ has a degree equal to, or smaller, than $i$. Therefore, a node $i$ will not make another move after executing $\text{RInVW}. □
**Lemma 7:** A node \( i \) that executes \( \text{RIn} \) will make at most one more move.

**Proof:** For \( i \) to make another move, it must next execute \( \text{ROut} \). Additionally, for \( i \) to have executed \( \text{RIn} \), none of its neighbors could have been in \( X \) at the time. Therefore, a neighbor of \( i \) must be a very weak node and have executed \( \text{RIn} \). As seen from Lemma 6, this node will never make another move, so after \( i \) executes \( \text{ROut} \), it will be next to a very weak node in \( X \) and cannot execute any further rules. □

**Lemma 8:** A node \( i \) that executes \( \text{ROut} \) will make at most two more moves.

**Proof:** Assume a node \( i \in X \) is next to a neighbor \( j \in X \) from the initial configuration. Node \( i \) can then execute \( \text{ROut} \). From Lemma 7, node \( i \) can execute at most two more moves. □

**Theorem 3:** Algorithm C3 finds a maximal independent set in at most \( 3n \) moves.

**Proof:** See Lemma 5 for correctness and Lemmas 6 and 8 for complexity. □

**D. Central Scheduler Algorithm 4**

Algorithm C4 is a further modification of Algorithm C3. As with Algorithm C2 and C3, \( C_{4_30} \) and \( C_{4_4} \) have the comparison signs reversed in order to find a larger and smaller maximal independent set. Algorithm C3 was concerned with very weak vertices. Algorithm C4 relaxes this to examine relatively very weak vertices. A vertex is relatively very weak if it has a smaller degree than all of its neighbors that are in the set \( X \).

**Algorithm 4 C4**

\[
\begin{align*}
\text{RIn:} & \quad \text{if } x(\text{id}(i)) = 0 \land (\exists j \in N(\text{id}(i))) (x(j) = 1 \land \text{deg}(i) \geq \text{deg}(j)) \\
& \quad \text{then } x(\text{id}(i)) = 1 \\
\text{ROut:} & \quad \text{if } x(\text{id}(i)) = 1 \land (\exists j \in N(\text{id}(i))) (x(j) = 1 \land \text{deg}(i) \geq \text{deg}(j)) \\
& \quad \text{then } x(\text{id}(i)) = 0
\end{align*}
\]

A solution was developed independently by Yen, Huang, and Turau in [14] for a similar algorithm named \( A_{\text{deg}} \), which provides an approximation ratio that was bounded by \((\Delta + 2)/3\). Their solution utilizes game theory and provides several measurements for its performance compared to other algorithms. Their findings also correspond with ours in Section IV that this approach, considering the degree of the node, provides the best solution for maximal independent sets.

**E. Central Scheduler Algorithm 5**

Algorithm C5 is modeled after the Grundy coloring algorithm of Hedetniemi, Jacobs and Srimani [9], and is found in [8] by Hedetniemi, Jacobs and Kennedy as an example of a self-stabilizing algorithm that can find two disjoint independent sets. In Algorithm C5, every node is assigned a color \( x(\text{id}(i)) \in \{1, 2, 3\} \) where:

\[
\begin{align*}
R = \{i|x(\text{id}(i)) = 1\} & \quad (2) \\
B = \{i|x(\text{id}(i)) = 2\} & \quad (3)
\end{align*}
\]

and the following function is defined:

\[
\text{Color}(i) = \min(\{1, 2, 3\} - \{x(j) | j \in N(\text{id}(i)) \cap \{1, 2\})
\]

When the algorithm stabilizes, as shown in [8], \( R \) is a maximal independent set in \( G = (V, E) \), and \( B \) is a maximal independent set in the induced subgraph \( G[V - R] = (V - R, (V - R) \times (V - R) - R) \).

**Algorithm 5 C5**

\[
\begin{align*}
\text{Re-color:} & \quad \text{if } x(\text{id}(i)) \neq \text{Color}(i) \\
& \quad \text{then } x(\text{id}(i)) = \text{Color}(i)
\end{align*}
\]

**Theorem 4:** Algorithm C5 finds a maximal independent set \( R \) in \( G \) and \( B \) in \( G[V - R] \) in at most \( 3n \) moves.

**Proof:** See proof of Algorithm 3 in [8]. □

**III. DISTRIBUTED ALGORITHMS**

In this section, we present five self-stabilizing algorithms for finding maximal independent sets running under the distance-1 model with an unfair distributed scheduler, whereby any number of eligible nodes may move at the same time. The algorithms are all silent. Four algorithms are new and proofs are given for their correctness and complexity. As before, results are given for the other algorithm with the reader being referred to the original paper for its proof. For each of the algorithms, we assume that the nodes \( i \) have unique identifiers \( \text{id}(i) \).

**A. Distributed Scheduler Algorithm 1**

Algorithm D1 was designed by Turau (see [12]) as a generalization of Algorithm C1. The algorithm includes \( \text{wait} \) and \( \text{back} \) states to take into account cases in which two adjacent nodes want to enter the independent set \( X \) at once by executing rule \( \text{RIn} \). \( \text{id} \) is used to determine which node will be allowed to enter the set.

**Algorithm 6 D1**

\[
\begin{align*}
\text{RWait:} & \quad \text{if } x(\text{id}(i)) = 0 \land (\exists j \in N(\text{id}(i))) (x(j) = 1) \\
& \quad \text{then } x(\text{id}(i)) = w \\
\text{RBack:} & \quad \text{if } x(\text{id}(i)) = w \land (\exists j \in N(\text{id}(i))) (x(j) = 1) \\
& \quad \text{then } x(\text{id}(i)) = 0 \\
\text{RIn:} & \quad \text{if } x(\text{id}(i)) = w \land (\exists j \in N(\text{id}(i))) (x(j) = 1) \land (\forall k \in N(\text{id}(i))) (x(k) \neq w \lor \text{id}(k) > \text{id}(i)) \\
& \quad \text{then } x(\text{id}(i)) = 1 \\
\text{ROut:} & \quad \text{if } x(\text{id}(i)) = 1 \land (\exists j \in N(\text{id}(i))) (x(j) = 1) \\
& \quad \text{then } x(\text{id}(i)) = 0
\end{align*}
\]

**Theorem 5:** Algorithm D1 finds a maximal independent set in at most \( \max(3n - 5, 2n) \) moves.

**Proof:** See proof of Algorithm \( A_{\text{MIS}} \) in [12]. □

**B. Distributed Scheduler Algorithm 2**

Algorithm D2 is the distributed version of Algorithm C2, the new algorithm presented earlier for finding a maximal independent set. As with Algorithm C2, the comparison \( \text{deg}(i) \)
Thus, when the system is stable, no node $i$ has no neighbors in the wait state or any neighbor in the wait state or any neighbor in the wait state.

Algorithm D2

| RWait: if $x(i) = 0 \land (\exists j \in N(i))(x(j) = 1)$ then $x(i) = w$ |
| RBack: if $x(i) = w \land (\exists j \in N(i))(x(j) = 1)$ then $x(i) = 0$ |
| RIn: if $x(i) = w \land (\exists j \in N(i))(x(j) = 1) \land (\forall k \in N(i))(x(k) \neq w \lor \deg(k) \geq \deg(i))$ then $x(i) = 1$ |
| ROut: if $x(i) = 1 \land (\exists j \in N(i))(x(j) = 1 \land \deg(i) \geq \deg(j))$ then $x(i) = 0$ |

Lemma 9: When the system is stable, the set $X$ of nodes with $x(i) = 1$ forms an independent set.

Proof: See proof of Lemma 1 with slight modification for wait state and ids.

Lemma 10: When the system is stable, no node $i$ has $x(i) = w$.

Proof: Assume that the system is stable and there exists a node $i$ with $x(i) = w$. Assume furthermore, that node $i$ has the smallest id of all nodes $j$ with $x(j) = w$. Since the system is stable, node $i$ is not eligible to execute rule RBack. This implies that no neighbor $j \in N(i)$ has $x(j) = 1$. Thus, every neighbor $j \in N(i)$ has $x(j) \in \{0, w\}$. But since the system is stable, and node $i$ is not eligible to execute rule RIn, node $i$ must have at least one neighbor, say $k \in N(i)$, with $x(k) = w$ and $\deg(k) < \deg(i)$. But this contradicts our assumption that node $i$ has the smallest id among all nodes with $x(j) = w$. Thus, when the system is stable, no node $i$ has $x(i) = w$.

Lemma 11: When the system is stable, the set $X$ forms a maximal independent set.

Proof: Assume that the system is stable and the set $X$ does not form a maximal independent set. From Lemma 9 we know that when the system is stable $X$ is an independent set. Therefore, $X$ is not a maximal independent set. This implies that there exists a node $i$ with $x(i) \in \{0, w\}$ and no neighbor $j \in N(i)$ has $x(j) = 1$. But from Lemma 6, we know that if the system is stable, then no node $i$ has $x(i) = w$. Therefore, there must exist a node $i$ with $x(i) = 0$ and all neighbors $j \in N(i)$ have $x(j) = 0$. Therefore, node $i$ is eligible to execute rule RWait, and the system is not stable.

Algorithm D3

| RWait: if $x(i) = 0 \land (\exists j \in N(i))(x(j) = 1) \lor (\exists k \in N(i))(\deg(i) \geq \deg(k))$ then $x(i) = w$ |
| RBack: if $x(i) = w \land (\exists j \in N(i))(x(j) = 1) \land (\exists k \in N(i))(x(k) \neq w \lor \deg(i) \geq \deg(k))$ then $x(i) = 0$ |
| RIn: if $x(i) = w \land (\exists j \in N(i))(x(j) = 1) \land (\forall k \in N(i))(x(k) \neq 0 \lor \deg(i) \geq \deg(k))$ then $x(i) = 1$ |
| ROut: if $x(i) = 1 \land (\exists j \in N(i))(x(j) = 1 \land \deg(i) \geq \deg(j))$ then $x(i) = 0$ |

Lemma 12: If a node $i$ executes rule RIn, it will never make another move.

Proof: Assume that a node $i$ executes rule RIn. The only rule it could be eligible to execute after this is rule ROut. By contradiction, suppose that node $i$ executes ROut sometime after executing RIn. Therefore, a neighbor $j$ of $i$ must have entered the independent set after, or at the same time, as $i$. From Lemma 9 we know that a neighbor $j$ of $i$ could not have executed RIn while $i$ is in the independent set. Therefore, $j$ must have executed RIn at the same time as $i$. However, the guard of RIn requires that a node either have no neighbors in the wait state or any neighbor in the wait state must have a larger id. As the ids are unique and $i$ executed RIn, node $j$ must have a larger id than $i$ and could not enter the independent set. Thus a node $i$ will never make another move after executing RIn. □

Lemma 13: A node $i$ that executes rule RWait can make at most one move.

Proof: Suppose there is a node $i$ that executes RWait. Node $i$ is now in the wait state and can only execute RBack or RIn.

Case 1: node $i$ executes RIn. From Lemma 12 we see that node $i$ will never make another move.

Case 2: node $i$ executes RBack. For node $i$ to have executed RBack, a node $j \in N(i)$ must have executed RIn in which case node $i$ will not be able to execute another move. □

Lemma 14: A node $i$ that executes ROut will make at most two more moves.

Proof: Suppose there is a node $i$ that executes ROut from its initial state. For node $i$ to have executed ROut, there had to have been a neighbor $j$ such that $x(j) = 1$ and $\deg(i) \geq \deg(j)$. It is possible that node $j$ executes ROut at the same time as $i$. Thus, node $i$ could execute RWait, and from Lemma 13 we see that $i$ will make at most one more move. □

Lemma 15: A node $i$ that executes RBack will make at most two more moves.

Proof: Suppose there is a node $i$ that executes RBack from its initial state. For node $i$ to have executed RBack, there had to have been a neighbor $j$ such that $x(j) = 1$. If node $j$ then executes ROut (only from the initial state), the next move that $i$ can make is RWait, and from Lemma 14 we see that $i$ could make at most one more move. □

Theorem 6: Algorithm D2 finds a maximal independent set in at most $3n$ moves.

Proof: This follows immediately from Lemmas 11, 12 and 15.

C. Distributed Scheduler Algorithm 3
Lemma 17: A node $i$ that executes RInVW will not make another move.

Proof: See proof of Lemma 6 □

Lemma 18: A node $i$ that executes RIn will make at most one other move.

Proof: From Lemmas [12] and [17] if node $i$ executes RIn, the only possible move for node $i$ to make will be to execute ROut. This can only occur if a neighbor $j \in N(i)$ executes RInVW. At that point, node $i$ will not make another move as RWait will not be enabled. □

Lemma 19: A node $i$ that executes rule RWait can make at most $2 * \deg(i) - 1$ more moves.

Proof: Suppose there is a node $i$ that executes RWait. Node $i$ is now in the wait state and can execute RBack, RIn, or RInVW.

Case 1: node $i$ executes RInVW. From lemma [17] we see that node $i$ will never make another move.

Case 2: node $i$ executes RIn. From lemma [18] we see that node $i$ will make at most one more move.

Case 3: node $i$ executes RBack. For node $i$ to have executed RBack, a neighbor $j \in N(i)$ must have executed RIn or RInVW. From Lemma [17] if node $j$ executes RInVW, $j$ will not make another move nor will node $i$. However from Lemma [18] if node $j$ executed RIn, a neighbor $k \in N(j)$ where $k \neq i$ may execute RInVW. After such a move, node $j$ must execute ROut allowing $i$ to once again execute RWait. This set of moves can occur up to $\deg(i)$ times allowing for a maximum of $2 * \deg(i) - 1$ moves at node $i$ after it has executed the initial RWait. □

Lemma 20: A node $i$ that executes rule RBack can make at most $2 * \deg(i)$ more moves.

Proof: A node $i$ may initially be in the wait state with a neighbor $j \in N(i)$ initially in the set $X$. Node $i$ can then execute RBack. The next move node $i$ could make is to execute RWait, at which point from Lemma [19] node $i$ can make at most $2 * \deg(i) - 1$ more moves. □

Lemma 21: A node $i$ may execute at most 2 ROut moves.

Proof: Nodes $i$ and $j$ where $j \in N(i)$ may be initially in the set $X$. At this point, $i$ can execute ROut. From Lemma [17] if a node executes RInVW, it will never make another move, and from Lemma [18] if a node executes RIn, it will execute ROut at most one more time. □

Theorem 7: Algorithm D3 finds a maximal independent set in at most $O(\Delta n)$ moves.

Proof: See Lemma [16] for correctness and Lemmas [17] [18] [19] [20] and [21] for complexity. □

D. Distributed Scheduler Algorithm 4

The maximum number of moves for this algorithm remains an open problem. As stated for Algorithm C4, an approximation ratio was developed by Yen, Huang, and Turau in [14] showing that the central solution was bounded by $(\Delta + 2)/3$.

E. Distributed Scheduler Algorithm 5

Algorithm D5, a distributed version of Algorithm C5, is given below. In [8], no distributed version of Algorithm C5 was given. We have included one below along with a proof of its correctness and complexity. $R$, $B$, and Color are defined in §II-E. Additionally, we define the following function:

\[
\text{swn}(i) \equiv \exists j \in N(i)(\text{id}(j) < \text{id}(i) \land x(j) = w)
\]

Algorithm 10 D5

\[
\begin{align*}
\text{RWait: if } x(i) &= 3 \land x(i) \neq \text{Color}(i) \\
&\quad \text{then } x(i) = w \\
\text{RBack: if } x(i) &= w \land \text{Color}(i) = 3 \\
&\quad \text{then } x(i) = 3 \\
\text{RIn: if } x(i) &= w \land \text{Color}(i) \neq 3 \land \neg \text{swn}(i) \\
&\quad \text{then } x(i) = \text{Color}(i) \\
\text{ROut: if } x(i) &\in \{1, 2\} \land x(i) \neq \text{Color}(i) \\
&\quad \text{then } x(i) = 3
\end{align*}
\]

Lemma 22: When the system is stable, no node is in the wait state.

Proof: By contradiction, let us assume that the system is stable and there is a node $i$ in the wait state that has the smallest ID of all nodes in the wait state. In this case, node $i$ is eligible to execute RBack or RIn, and therefore, the system is not stable. □

Lemma 23: When the system is stable, $R$ is maximal independent in $G$, and $B$ is maximal independent in $G[V - R]$.

Proof: By Lemma [22] when the system is stable, no node is in the wait state. The set $R$ must be an independent set; otherwise, two nodes $i, j \in R$ would be adjacent having $x(i) = x(j) = 1$ and both would be eligible to execute ROut. A similar argument shows that $B$ must be an independent set. Additionally, $R$ must be maximal independent in $G$. If a node $j$ is not adjacent to a node in $R$, it can execute either RWait or RIn. Likewise, $B$ must be maximal independent in $G[V - R]$. □

Lemma 24: If a node $i$ executes RIn and $x(i) = 1$, then it will never make another move.

Proof: Due to the swn function, we know that all neighbors of $i$ in the wait state must have larger ids than $i$. Additionally, from the Color function, no neighbor of $i$ can have the color 1. Therefore, after $i$ executes RIn, no neighbor of $i$ will be able to set its color to 1, and $i$ will never move again. □
Lemma 25: If a node $i$ executes $\text{RIn}$ and $x(i) = 2$, then node $i$ will make at most $2 \cdot \deg(i) - 1$ more moves.

Proof: Suppose node $i$ has the largest id of all of its neighbors and executes rule $\text{RIn}$ and $x(i) = 2$. As in Lemma 24 no neighbor of $i$ will set its color to 2 while $i$ has color 2. Therefore, for $i$ to move again, a neighbor $j$ with $x(j) = 1$ must execute $\text{ROut}$. This is possible in the initial state if $j$ is adjacent to a neighbor also colored 1. At this point, $i$ can execute $\text{ROut}$ and then $\text{RWait}$ as it has no neighbors colored 1.

Case 1: If $i$ next executes $\text{RIn}$ and sets $x(i) = 1$, from Lemma 24, we know that $i$ will never make another move.

Case 2: A neighbor $k$ of $i$ sets $x(k) = 2$. Node $i$ can now execute $\text{RBack}$. Assume $k$ is also adjacent to a neighbor $l$ with $x(l) = 1$ from the initial state. As with $j$, $l$ may execute $\text{ROut}$ allowing $k$ to set its color to 1. At this point, node $i$ can again execute $\text{RWait}$ to attempt to set $x(i) = 2$. This process can repeat for $\deg(i) - 1$ times. At the end, $i$ will either execute $\text{RBack}$ or $\text{RIn}$. □

Lemma 26: A node $i$ may execute at most $\deg(i)$ $\text{RWait}$ moves.

Proof: This follows from Lemma 25 □

Lemma 27: A node $i$ may execute at most $\deg(i) + 1$ $\text{RBack}$ moves.

Proof: A node $i$ may initially be in the wait state when it has neighbors colored 1 and 2. At this point, $i$ may execute $\text{RBack}$. From Lemma 26 we see that there may be $\deg(i)$ $\text{RWait}$ moves, each of which may be followed by an $\text{RBack}$ move. □

Lemma 28: A node $i$ may execute at most 2 $\text{ROut}$ moves.

Proof: A node in the initial state with color 1 or 2 may execute $\text{ROut}$ if it is adjacent to a neighbor with the same color. From Lemma 24 we know that after a node executes $\text{RIn}$ and sets $x(i) = 1$, it will never move again. However, if it sets $x(i) = 2$, it may execute one more $\text{ROut}$ rule if a neighbor $j$ where $x(j) = 1$ from the initial configuration executes $\text{ROut}$. □

Theorem 8: Algorithm D5 finds a maximal independent set $R$ in $G$ and a maximal independent set $B$ in $G[V - R]$ in $O(\Delta n)$ moves.

Proof: See Lemma 23 for correctness and Lemmas 24, 25, 26, 27 and 28 for complexity. □

IV. PERFORMANCE COMPARISONS

In the previous two sections, we have presented several self-stabilizing algorithms for finding maximal independent sets, using both central and distributed schedulers. We implemented each algorithm and run simulations on four classes of graphs: trees (connected and acyclic graphs), bipartite graphs (connected graphs having no cycles of odd length), unit disk graphs (the intersection graphs of unit disks randomly placed in the plane) and arbitrary connected graphs. In each case we randomly generated 5,000 graphs of order $n = 500$, for each of these four types. For algorithms using the central scheduler, we selected at random any of the currently eligible nodes to move, and for algorithms using the distributed scheduler, we randomly selected a subset of the set of currently eligible nodes to move. In each case we recorded the number of moves made before the algorithm stabilized, and the size of the independent set found by the algorithm.

Our simulation results indicate that the central scheduler algorithms usually find larger independent sets than the distributed scheduler algorithms, see Table I, and generally require fewer moves, see Table II, to reach a stable state. The second result was expected, as distributed algorithms often include wait states. However, we were not expecting the first result. Of the algorithms studied, Algorithm C2, and its distributed version D2, are faster than the previously published algorithms C1 and D1. In terms of set size, the following inequality was observed for finding large maximal independent sets:

\[ C_4, D_4 \geq C_3, D_3 \geq C_2, D_2 \geq C_1 = C_5, D_1 = D_5 \quad (4) \]

and the following inequality for small maximal independent sets:

\[ C_4, D_4 < C_3, D_3 < C_2, D_2 < C_1 = C_5, D_1 = D_5 \quad (5) \]

Algorithm C4, and the distributed D4, consistently produced the largest and smallest maximal independent sets. This is also consistent with findings in [14]. Algorithm C5, and its distributed version D5, produced sets of the same size as C1 and the D1 but made far more moves.

V. CONCLUSIONS

With the introduction of distributed sensors in environments such as the smart warehouse, the need for algorithms that can be run on these sensors in order to create topological layouts, such as maximal independent sets, will only increase. This paper has presented a variety of algorithms and approaches for determining fast methods for finding a MIS set as well as methods for determining the smallest/largest MIS set.

The complexity of Algorithm D4, as well as a fixed bound for Algorithm C4, are still open problems and of interest to the authors.

REFERENCES

[1] K. Alzoubi, P.J. Wan, O. Frieder, Maximal independent set, weakly-connected dominating set, and induced spanners in wireless ad hoc networks, International Journal of Foundations of Computer Science 14 (2)(2003)287-303.

[2] W. Goddard, S. T. Hedetniemi, D. P. Jacobs and P. K. Srimani, Self-stabilizing protocols for maximal matching and maximal independent sets for ad hoc networks. Proc. Fifth IPDPS Workshop on Advances in Parallel and Distributed Computational Models, Nice, France, pp. 162, 2003.

[3] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, P. K. Srimani and Z. Xu, Self-stabilizing graph protocols. Parallel Process. Lett. 18:189-199, 2005.

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

[5] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater, Fundamentals of Domination in Graphs, Marcel-Dekker, New York, 1998.
| Algorithm | Tree % Diff. | Bipartite % Diff. | Unit Disk % Diff. | Connected % Diff. |
|-----------|-------------|------------------|------------------|------------------|
| C1        | 267.5       | 167.6            | 61.1             | 95.1             |
| C2\beta_0 | 282.5       | 178.5            | 65.9             | 102.8            |
| C2_i      | 247.3       | 154.4            | 58.1             | 86.5             |
| C3\beta_0 | 296.9       | 185.7            | 68.0             | 106.4            |
| C3_i      | 232.5       | 149.9            | 57.6             | 85.0             |
| C4\beta_0 | 297.2       | 186.5            | 71.3             | 107.6            |
| C4_i      | 221.1       | 148.1            | 75.2             | 88.4             |
| C5        | 267.2       | 167.3            | 61.1             | 95.0             |
| D1        | 264.2       | 166.7            | 60.9             | 94.4             |
| D2\beta_0 | 272.8       | 174.3            | 64.5             | 100.6            |
| D2_i      | 253.2       | 157.7            | 58.4             | 87.9             |
| D3\beta_0 | 296.7       | 185.8            | 67.5             | 106.3            |
| D3_i      | 233.3       | 150.8            | 57.8             | 85.6             |
| D4\beta_0 | 297.1       | 186.4            | 71.3             | 107.6            |
| D4_i      | 221.1       | 147.5            | 75.2             | 88.4             |
| D5        | 263.3       | 166.3            | 60.9             | 94.2             |

### TABLE I

**AVERAGE SIZE OF MAXIMAL INDEPENDENT SETS FOUND**

| Algorithm | Complexity | Tree % Diff. | Bipartite % Diff. | Unit Disk % Diff. | Connected % Diff. |
|-----------|------------|-------------|------------------|------------------|------------------|
| C1        | 2n         | 205.4       | 253.5            | 239.1            | 275.9            |
| C2\beta_0 | 2n         | 182.0       | 200.8            | 215.4            | 225.7            |
| C2_i      | 2n         | 197.9       | 240.0            | 237.5            | 259.3            |
| C3\beta_0 | 3n         | 274.0       | 259.6            | 237.9            | 261.1            |
| C3_i      | 3n         | 253.3       | 274.3            | 246.9            | 276.2            |
| C4\beta_0 | 3n         | 285.8       | 288.2            | 300.0            | 302.8            |
| C4_i      | 3n         | 389.7       | 396.6            | 344.9            | 382.5            |
| C5        | max(3n-5,2n) | 388.1     | 422.2            | 393.8            | 474.6            |
| D1        | 3n         | 383.4       | 381.8            | 351.8            | 389.5            |
| D2\beta_0 | 3n         | 376.9       | 394.5            | 369.3            | 416.7            |
| D2_i      | 3n         | 490.1       | 449.8            | 387.1            | 429.2            |
| D3\beta_0 | O(\Delta n) | 444.7     | 452.8            | 388.9            | 450.4            |
| D3_i      | O(\Delta n) | 511.8     | 487.7            | 481.1            | 484.6            |
| D4\beta_0 | O(\Delta n) | 659.6     | 655.8            | 550.9            | 624.8            |
| D4_i      | O(\Delta n) | 646.1     | 628.6            | 513.4            | 688.6            |

### TABLE II

**WORST-CASE COMPLEXITY AND AVERAGE NUMBER OF MOVES**

[6] S. M. Hedetniemi, S. T. Hedetniemi, D. P. Jacobs and P. K. Srimani, Self-stabilizing algorithms for minimal dominating sets and maximal independent sets. *Comput. Math. Appl.* 46 (2003) 803–811.

[7] S. M. Hedetniemi, S. T. Hedetniemi, K. E. Kennedy, and A. A. McRae, Self-stabilizing Algorithms for Unfriendly Partitions into two Disjoint Dominating Sets. *Parallel Processing Letters* (March 2013).

[8] S. T. Hedetniemi, D. P. Jacobs and K. E. Kennedy, Linear-time self-stabilizing algorithms for disjoint independent sets. *The Computer Journal* (10 September 2012).

[9] S. T. Hedetniemi, D. P. Jacobs and P. K. Srimani, Linear time self-stabilizing colorings. *Inform. Process. Lett.* 87:251-255, 2003.

[10] M. Ikeda, S. Kamei and H. Kakugawa, A space-optimal self-stabilizing algorithm for the maximal independent set problem. *Proc. 3rd Int. Conf. on Parallel and Distributed Computing, Applications and Technologies*, 70-74, 2002.

[11] S. Shukla, D. Rosenkrantz and S. Ravi, Observations on self-stabilizing graph algorithms for anonymous networks. *proc. 2nd Workshop on Self-Stabilizing Systems*, 7.1-7.5, 1995.

[12] V. Turau. Linear self-stabilizing algorithms for the independent and dominating set problems. *Inform. Process. Lett.* 103 (2007) 88-93.

[13] Peng Wang, Stephan Bohacek, On the practical complexity of solving the maximum weighted independent set problem for optimal scheduling in wireless networks. *WICON’08* (17-19 November 2008).

[14] Li-Hsing Yen, Jean-Yao Huang and Volker Turau, Designing Self-Stabilizing Systems Using Game Theory. *ACM Transactions on Autonomous and Adaptive Systems*, Volume 11, Issue 3, Article 18, 2016.
Fig. 1. Comparison of the average number of moves for the C4/D4 algorithms on random trees of various orders.

Fig. 2. Comparison of the % differences for the average number of moves of the C4/D4 algorithms with C1/D1 on random trees of various orders.