A Fault-Containing Self-Stabilizing Algorithm for 6-Coloring Planar Graphs

JI-CHERNG LIN AND MING-YI CHIU
Department of Computer Science and Engineering
Yuan Ze University
Chungli, 320 Taiwan

This paper presents the first fault-containing self-stabilizing algorithm which can 6-color any planar graph. Besides the capability to contain the fault in any single-fault situation, the proposed algorithm also has the capability to stabilize faster in single-fault situations. For single-fault situations, the worst-case stabilization time of the proposed algorithm is \(O(\Delta)\), whereas the worst-case stabilization times of all the previous self-stabilizing algorithms for 6-coloring planar graphs are \(\Omega(n)\), where \(\Delta\) is the maximum node degree, and \(n\) is the number of nodes in the system.

**Keywords:** self-stabilization, fault-containment, stabilization time, graph coloring, planar graph

1. INTRODUCTION

A distributed system consists of a set of loosely connected processors that do not share a common or global memory. Each processor has one or more local variables, the contents of which specify the local state of the processor. Local states of all processors in the system at a certain time constitute the global configuration (or, simply, configuration) of the system at that time. The main restriction of a distributed system is that each processor in the system can only access the data (i.e., read the shared data) of its neighbors. Since a distributed algorithm is an algorithm that works in a distributed system, it must abide by this main restriction. Depending on the purpose of a distributed system, a global criterion for the global configuration is defined. Those global configurations satisfying the criterion are called legitimate configurations, whereas other global configurations are called illegitimate configurations. When the system is in a legitimate configuration, the purpose of the system is fulfilled.

Dijkstra’s central demon model of computation (cf. [1]) for an algorithm in a distributed system can be described as follows:

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

\[\text{condition part} \rightarrow \text{action part}.\]

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

(b) At the initial configuration, if none of the processors is privileged, then the system is deadlocked. Otherwise, if a privileged processor exists, the central demon in the system will randomly select exactly one among all the privileged processors to make a move, in a single atomic step. The local state of the selected processor thus changes, which in the meantime results in the change of the global configuration of the system. The system will then repeat the above process to change global configurations as long as it does not encounter any deadlock situation. Thus, the behavior of the system under the action of the algorithm can be described by executions. An infinite or finite sequence of configurations \( \Gamma = (\gamma_1, \gamma_2, \ldots) \) of the system is called an execution (of the algorithm in the system) if for any \( i > 1 \), \( \gamma_i \rightarrow \gamma_{i+1} \) is obtained from \( \gamma_i \) after exactly one processor in the system makes the ith move \( \gamma_i \rightarrow \gamma_{i+1} \), and in the case that \( \Gamma \) is finite, it is further required that no node is privileged in the last configuration.

Under this central demon model, Dijkstra introduced the notion of self-stabilization of a distributed system in his classic paper [1] in 1974 (cf. also [2, 3]). Following Dijkstra’s idea, in this paper we define an algorithm to be self-stabilizing if every execution of the algorithm has a suffix in which all configurations are legitimate configurations.

Adopting Dijkstra’s computational model and his definition of self-stabilizing algorithms, Ghosh et al. introduced fault-containment of self-stabilizing algorithms in 1996-1997 [4-8]. From the viewpoint of Ghosh et al., there are two essential requirements for fault-containing self-stabilizing algorithms: the capability to contain faults, and the capability to self-stabilize faster, in situations where the system incurs only a limited number of transient faults. In [5-7], Ghosh et al. proposed fault-containing self-stabilizing algorithms for the leader election problem (for oriented rings), for the spanning tree problem, and for the BFS spanning tree problem, respectively. For single-fault situations, the worst-case stabilization times of these three algorithms are \( \Theta(1) \) moves, \( \Theta(\Delta) \) moves and \( \Theta(\Delta^2) \) moves, respectively, where \( \Delta \) is the maximum node degree in the system. In [8, 9], Ghosh et al. took a general approach and acquired a transformer that can convert any nonreactive self-stabilizing algorithm into a nonreactive fault-containing self-stabilizing algorithm. For single-fault situations, the worst-case stabilization time of any transformed algorithm is \( \Omega(n^2) \) moves, where \( n \) is the number of nodes in the system. In [10], Lin and Huang proposed a fault-containing self-stabilizing algorithm for finding a maximal independent set. Recently, in [11], Huang proposed a fault-containing self-stabilizing algorithm for the shortest path problem, which generalizes the results in [7]. For both of the latter two algorithms, the worst-case stabilization times for single-fault situations are \( \Theta(\Delta) \) moves.

Quite a few self-stabilizing graph-coloring algorithms have been proposed in the past [12-18]. The three algorithms proposed in Gradinariu and Tixeuil [12] and the two algorithms proposed in Hedetniemi et al. [13] are all self-stabilizing graph-coloring algorithms for general graphs, and use \( \Delta + 1 \) colors. In [14], Sur and Srimani presented a self-stabilizing algorithm for 2-coloring any bipartite graph. Recently, Kosowski and Kuszner [15] proposed two self-stabilizing graph-coloring algorithms. Either of the two algorithms can produce a \( 2\Delta \) coloring for any graph, and can produce an even better 2 coloring for any bipartite graph. In [16], Ghosh and Karaata introduced a self-stabilizing
A FAULT-CONTAINING SELF-STABILIZING GRAPH-COLORING ALGORITHM

algorithm under the central demon model, which can 6-color any planar graph. In [17], Huang et al. proposed a self-stabilizing algorithm under the fair demon model, which can also 6-color any planar graph. This algorithm does not need node identifiers and uses only $n$-bounded variables. In [18], Goddard et al. presented a self-stabilizing coloring algorithm under the central demon model, which can $k$-color any graph that has a $k$-forward numbering. Since any planar graph has a 6-forward numbering, this algorithm can 6-color any planar graph. None of the above three self-stabilizing planar-graph 6-coloring algorithms in [16-18] is fault-containing. Moreover, it can be verified that for single-fault situations, the worst-case stabilization time of these three planar-graph 6-coloring algorithms are $\Omega(n)$, $\Omega(n^{\sqrt{2}})$ and $\Omega(n^2)$ moves, respectively.

In this paper we will propose a fault-containing self-stabilizing algorithm that can produce a 6-coloring for any planar graph. Besides the capability to contain the fault in any single-fault situation, the proposed algorithm also has the capability to stabilize faster in single-fault situations. For single-fault situations, the worst case stabilization time of the proposed algorithm is $O(\Delta)$.

The rest of this paper is organized as follows. In section 2, the whole process of designing our fault-containing self-stabilizing algorithm is presented in roughly three steps: (1) A modified version of the algorithm in [16] is presented, which is still a self-stabilizing algorithm for 6-coloring planar graphs and not fault-containing; (2) the guard conditions of two rules in the modified algorithm are classified into various cases, to help understand the effects of the moves made by nodes that satisfy the conditions of these cases in any single-fault situation, and from this analysis a prototype of our fault-containing self-stabilizing algorithm is derived; and (3) the prototype is transformed into a distributed algorithm, which is the desired fault-containing self-stabilizing algorithm. In section 3, the correctness proof of our fault-containing algorithm is given, which consists of the verifications of (1) the no-deadlock property, (2) the self-stabilization property, and (3) the fault-containment property. In section 4, the worst-case stabilization time for single-fault situations is computed. Finally in section 5, some short remarks conclude the discussion in this paper.

2. THE ALGORITHM

In this section, we present a fault-containing self-stabilizing algorithm for 6-coloring planar graphs. In the earlier stage of this research, we tried to use the same classification approach as in [10, 11] to modify the planar-graph 6-coloring algorithm in [16] to obtain a desired fault-containing algorithm, but did not succeed. We finally decided to modify the planar-graph 6-coloring algorithm in [16] into the following Algorithm 1 first, and then apply the classification approach in [10, 11] to Algorithm 1.

Consider a distributed system whose underlying topology is a connected undirected planar graph $G = (V, E)$. Each node $i \in V$ represents a processor in the system and each edge $\{i, j\} \in E$ represents the bidirectional link connecting processors $i$ and $j$. We assume that each node has a unique identifier, and we will use the unique identifier to represent the node. We also assume that each node $i$ maintains the following two variables:

c.i: the color of node $i$ whose value is taken from the set $\{0, 1, 2, 3, 4, 5\}$.

x.i: an integer.
For every two adjacent nodes $i$ and $j$, $j$ is called a successor of $i$ and $i$ a predecessor of $j$ if $x.i < x.j \lor (x.i = x.j \land i < j)$. The following notations will be used:

$K = \{0, 1, 2, 3, 4, 5\}$: the set of all colors.

$N(i)$: the set of all neighbors of $i$.

$N_s(i)$: the set of all successors of $i$.

$N_p(i)$: the set of all predecessors of $i$.

$SN(i) = \{ j \in N(i) \mid c.i = c.j \}$ denotes the set of all neighbors of $i$, which has the same color as $i$.

$SN_s(i) = \{ j \in N_s(i) \mid c.j = c.i \}$ denotes the set of all successors of $i$, which has the same color as $i$.

$SN_p(i) = \{ j \in N_p(i) \mid c.j = c.i \}$ denotes the set of all predecessors of $i$, which has the same color as $i$.

$NC(i) = \{ c.j \mid j \in N(i) \}$ denotes the set of colors of all neighbors of $i$.

$NC_s(i) = \{ c.j \mid j \in N_s(i) \}$ denotes the set of colors of all successors of $i$.

Note that $N_s(i) \cup N_p(i) = N(i)$ and $SN_s(i) \cup SN_p(i) = SN(i)$.

**Algorithm 1**

[For each node $i \in V$]

\[ R_1: |NC(i)| \leq 5 \land |SN(i)| \geq 1 \rightarrow c.i = b, \text{ where } b \in K - NC(i); \]

\[ R_2: |NC(i)| = 6 \land |NC_s(i)| \leq 5 \land |SN_s(i)| \geq 1 \rightarrow c.i = b, \text{ where } b \in K - NC_s(i); \]

\[ R_3: |NC_s(i)| = 6 \rightarrow x.i = 1 + \max_{j \in N(i)} x.j. \]

Legitimate configurations for Algorithm 1 are defined to be all those configurations in which the following condition holds:

\[ \forall i \in V, |SN(i)| = 0 \text{ (or, equivalently, } \forall i \in V, \forall j \in N(i), c.i \neq c.j \). \]

The following lemma will be proved in Appendix 1.

**Lemma 1** Algorithm 1 is self-stabilizing.

In order to modify Algorithm 1 into a fault-containing version, we investigate all the nontrivial single-fault configurations for the system equipped with Algorithm 1. If the system starts in a legitimate configuration and incurs a single fault (one or more variables of exactly one node have their values changed), then the resulting configuration is called a single-fault configuration. A nontrivial single-fault configuration is a single-fault configuration in which the faulty node $u$ has the value of its primary variable $c.u$ corrupted, such that $|SN(u)| \geq 1$. Note that nontrivial single-fault configurations are exactly those single-fault configurations that are illegitimate. Hence trivial single-fault configurations are exactly those single-fault configurations that are legitimate. In a nontrivial single-fault configuration, the faulty node $u$ is certainly privileged by $R1$ of Algorithm 1, while some of its neighbors may also be privileged by $R1$ or $R2$ of Algorithm 1. Thus all of these
nodes are privileged to make moves to change their \( c \)-values according to Algorithm 1. However, if a “wrong” node is selected first by the central demon to make a move to change its \( c \)-value, the faulty situation gets worse, which is likely to cause the system to take a long time to self-stabilize. Therefore, restrictions should somehow be imposed on \( R1 \) and \( R2 \) of Algorithm 1 to help the central demon to select first a “right” node to change its \( c \)-value. In order to distinguish “right” nodes from “wrong” nodes, we classify the guard conditions of \( R1 \) and \( R2 \) into the following 7 cases:

1. \(|NC(i)| \leq 5 \land |SN(i)| \geq 1\) (The guard condition of \( R1 \) of Algorithm 1)
   1.1 \(|SN(i)| \geq 2\) (This case will be referred to as Case 1 in the rest of the paper)
   1.2 \(|SN(i)| = 1\) (let \( j \) be the unique node in \( SN(i) \))
      1.2.1 \(|SN(j)| \geq 2\) (Case 2)
      1.2.2 \(|SN(j)| = 1\) (Case 2.1)
      1.2.2.1 \(|NC(j)| \leq 5\) (Case 3)
      1.2.2.2 \(|NC(j)| = 6\) (Case 4)
2. \(|NC(i)| = 6 \land |NC_N(i)| \leq 5 \land |SN_N(i)| \geq 1\) (The guard condition of \( R2 \) of Algorithm 1)
   2.1 \(|SN(i)| \geq 2\) (Case 5)
   2.2 \(|SN(i)| = 1\) (let \( j \) be the unique node in \( SN(i) \))
      2.2.1 \(|NC(j)| \leq 5\) (Case 6)
      2.2.2 \(|NC(j)| = 6\) (Case 7)

Note that Cases 1-4 are exhaustive and Cases 5-7 are also exhaustive. Corresponding to these 7 cases, we obtain the following 3 lemmas. For the sake of presentation in the following lemmas as well as in the rest of the paper, for any time instant \( t \), we say that a predicate is true at \( t+ \) if it is true right after \( t \), and that a predicate is true at \( t- \) if it is true right before \( t \). Also by definition, a predicate is true at \( t \) if and only if it is true at both \( t+ \) and \( t- \); and a predicate is true in a time interval if it is true at any instant \( t \) in the interval.

In the following three lemmas, we assume that the system starts in a legitimate configuration and incurs a transient fault at a certain time instant \( t_0 \).

**Lemma 2** If, at \( t_0^+ \), there is a node satisfying the condition for Case 4, 5 or 7, then the transient fault at \( t_0 \) is not a single-fault.

In the following two lemmas, we assume that the transient fault at \( t_0 \) is a nontrivial single-fault. By Lemma 1, there must be a node executing \( R1 \) or \( R2 \) in \((t_0, \infty)\). Let \( t_1 \) be the first time instant in \((t_0, \infty)\) at which there is a node executing \( R1 \) or \( R2 \). Then, in view of Algorithm 1, if there is a node \( i \) satisfying the condition for Case 1, 2 or 3 at \( t_0^+ \), then \( i \) must be privileged by \( R1 \) at \( t_1^- \), and if there is a node \( i \) satisfying the condition for Case 6 at \( t_0^+ \), then \( i \) must be privileged by \( R2 \) at \( t_1^- \).

**Lemma 3** Suppose that, at \( t_0^+ \), there is a node \( i \) satisfying the condition for Case 1 or 2. If \( i \) executes \( R1 \) at \( t_1 \), then the system gets back to a legitimate configuration at \( t_1^+ \).

**Lemma 4** Suppose that, at \( t_0^+ \), there is a node \( i \) satisfying the condition for Case 3 or 6. If \( i \) executes \( R1 \) or \( R2 \) at \( t_1 \), then the system cannot get back to a legitimate configuration at \( t_1^+ \).
According to Lemmas 2, 3 and 4, for the fault-containment mechanism to be set, only a node that satisfies the condition in Case 1, 2, 4, 5, or 7 should be allowed to make a move to change its $c$-value. So we impose these conditions on $R1$ or $R2$ of Algorithm 1, and obtain the following prototype for our fault-containing algorithm:

**Prototype 1**

[For each node $i \in V$]

$R1$: $|NC(i)| \leq 5 \land |SN(i)| \geq 2 \rightarrow c.i. = b, b \in K - NC(i)$ (This corresponds to Case 1 in the above classification.)

$R2$: $|NC(i)| \leq 5 \land |SN(i)| = 1 \land |SN(j)| = 1$, where $j$ is the unique node in $SN(i) \rightarrow c.i. = b, b \in K - NC(i)$ (This corresponds to Case 2.)

$R3$: $|NC(i)| \leq 5 \land |SN(i)| = 1 \land |SN(j)| \geq 2 \land |NC(j)| = 6$, where $j$ is the unique node in $SN(i) \rightarrow c.i. = b, b \in K - NC(i)$ (This corresponds to Case 4.)

$R4$: $|NC(i)| = 6 \land |NC_i(j)| \leq 5 \land |SN_i(j)| \geq 1 \land |SN(i)| \geq 2 \rightarrow c.i. = b, b \in K - NC_i(i)$ (This corresponds to Case 5.)

$R5$: $|NC(i)| = 6 \land |NC_i(j)| \leq 5 \land |SN_i(j)| \geq 1 \land |SN(i)| = 1 \land |NC(j)| = 6$, where $j$ is the unique node in $SN(i) \rightarrow c.i. = b, b \in K - NC_i(i)$ (This corresponds to Case 7.)

$R6$: $|NC(i)| = 6 \rightarrow x.i. = 1 + \max \{ j \in SN_i(j) \} \rightarrow c.i. = b, b \in K - NC_i(i)$ (This is Case 3 of Algorithm 1.)

Legitimate configurations for Prototype 1 are defined to be all those configurations in which the following condition holds:

$$\forall i \in V, |SN(i)| = 0.$$ 

Note that each of $R2$, $R3$ and $R5$ of Prototype 1 requires a node to collect information from some 2-hop neighbors (e.g., in the guard condition of $R3$, node $j$ is a neighbor of node $i$, and $i$ needs to know $SN(j)$ and $NC(j)$, which involve information about $i$'s 2-hop neighbors). However, in a distributed system, a node is not allowed to read information from nodes other than its direct neighbors. Therefore, in order for us to transform Prototype 1 into a distributed algorithm, auxiliary secondary variables $q$ (question) and $a$ (answer) need to be used to fulfill the job of collecting information. (The idea of applying auxiliary variables $q$ and $a$ is attributed to Ghosh et al. [5-8] and Gupta [4].) Thus our fault-containing algorithm is ready.

**Algorithm 2**

[For each node $i \in V$]

$R1$: $|NC(i)| \leq 5 \land |SN(i)| \geq 2 \rightarrow c.i. = b, b \in K - NC(i)$ (This is $R1$ of Prototype 1.)

$R2$: $|NC(i)| = 6 \land |NC_i(j)| \leq 5 \land |SN_i(j)| \geq 1 \land |SN(i)| \geq 2 \rightarrow c.i. = b, b \in K - NC_i(i)$ (This is $R4$ of Prototype 1.)

$R3$: $|SN(i)| = 1 \land a.j = 0$, where $j$ is the unique node in $SN(i) \land q.e = 0 \rightarrow q.e = 1$

$R4$: $|SN(i)| = 1 \land |NC_i(j)| \leq 5 \land q.j = 1$, where $j$ is the unique node in $SN(i) \land a.j \neq 1 \rightarrow a.j = 1$

$R5$: $|SN(i)| \geq 1 \land |NC_i(j)| = 6 \land \exists j \in SN(i)$ s.t. $q.j = 1 \land a.j \neq 2 \rightarrow a.j = 2$

$R6$: $|NC_i(j)| \leq 5 \land |SN(i)| = 1 \land a.j = 1 \land a.j = 1$, where $j$ is the unique node in $SN(i) \rightarrow a.j = 1$.
c.i. = b, b ∈ K − NC(i) (Rules R3, R4 and R6 here are devised to implement R2 of Prototype 1.)
R7: |NC(i)| ≤ 5 ∧ |SN(i)| = 1 ∧ q.i = 1 ∧ a.j = 2, where j is the unique node in SN(i) →
c.i. = b, b ∈ K − NC(i) (Rules R3, R5 and R7 here are devised to implement R3 of Prototype 1.)
R8: |NC(i)| = 6 ∧ |NCs(i)| ≤ 5 ∧ |SN(i)| ≥ 1 ∧ |SN(i)| = 1 ∧ q.i = 1 ∧ a.j = 2, where j is the
unique node in SN(i) → c.i. = b, b ∈ K − NCs(i) (Rules R3, R5 and R8 here are
devised to implement R5 of Prototype 1.)
R9: |SN(i)| = 0 ∧ q.i = 1 → q.i. = 0
R10: ∀j ∈ SN(i), q.j = 0 ∧ a.i = 0 → a.i. = 0
R11: |NCs(i)| = 6 → x.i = 1 + max j∈N(i) x.j (This is R6 of Prototype 1.)

Note that in the above algorithm, the variable q takes values in the set {0, 1}, and
the variable a takes values in the set {0, 1, 2}. Legitimate configurations for Algorithm 2
are defined to be all those configurations in which the following condition holds:

∀i ∈ V, |SN(i)| = 0 ∧ q.i = 0 ∧ a.i = 0.

To help readers comprehend the relationship between Algorithm 2 and Prototype 1, as an
example, we explain how rules R3, R4 and R6 of Algorithm 2 are derived from rule R2 of
Prototype 1. When an internal node i satisfies the guard condition for R2 of Prototype 1
(i.e., |NC(i)| ≤ 5 ∧ |SN(i)| = 1 ∧ |SN(i)| = 1, where j is the unique node in SN(i)), according
to R3 of Algorithm 2, it sets q.i = 1 to ask node j if |SN(j)| = 1. Then according to R4
of Algorithm 2, node j sets a.j = 1 to answer i that |SN(j)| = 1. Then the guard condition
of R6 holds, and hence i can make a move to change its c-value. Thus rule R2 of Proto-
type 1 is faithfully implemented. Note that in rule R3 of Algorithm 2, condition |a.j = 0,
where j is the unique node in SN(i)| is placed there as a requisite. Suppose the condition
is taken away from R3. Then in the case that |SN(j)| ≥ 2 ∧ |NC(j)| ≤ 5 and yet a.j = 1,
node i may set q.i = 1 and then go ahead to execute R6, which is not the intended result
from R2 of Prototype 1.

3. CORRECTNESS PROOF

In this section, we will prove the correctness of Algorithm 2, which consists of the
verifications of (1) the no-deadlock property, (2) the self-stabilization property and (3)
the fault-containment property.

Lemma 8 In any configuration, if there exists a node v ∈ V such that |SN(v)| ≥ 1, then
there must be a node i ∈ V such that |SN(i)| = 0 ∧ |SN(v)| ≥ 1.

Proof: Suppose not. Then for every node k ∈ V, |SN(k)| ≥ 1 or |SN(k)| = 0. In particular,
|SN(v)| ≥ 1 or |SN(v)| = 0. If |SN(v)| = 0, then |SN(v)| = |SN(v)| − |SN(v)| ≥ 1, a
contradiction. Hence |SN(v)| ≥ 1. Let v1 be a node in SN(v). Then v ∈ SN(v1), and hence |SN(v1)| ≥ 1. It follows that |SN(v)| ≥ 1. Let v2 be a node in SN(v1). Continuing in this man-
ner, we get \( v_1, v_2, \ldots \) such that \( v_1 \in SN_i(v), v_2 \in SN_i(v_1), \ldots \). Since \( V \) is finite, there exist two positive integers \( m \) and \( l \), with \( m < l \), such that \( v_m = v_l \). Hence \( v_{m+1} \in SN_i(v_m), v_{m+2} \in SN_i(v_{m+1}), \ldots, v_{l-1} \in SN_i(v_l) \). It follows that \( x.v_m < x.v_{m+1} \lor (x.v_m = x.v_{m+1} \land v_m < v_{m+1}), x.v_{m+1} < x.v_{m+2} \lor (x.v_{m+1} = x.v_{m+2} \land v_{m+1} < v_{m+2}), \ldots, x.v_{l-1} < x.v_l \lor (x.v_{l-1} = x.v_l) \). By transitivity, \( x.v_m < x.v_l \lor (x.v_m = x.v_l \land v_m < v_l) \), that is, \( x.v_m < x.v_m \lor (x.v_m = x.v_m \land v_m < v_m) \), a contradiction. Therefore there must be a node \( i \in V \) such that \( |SN_i(i)| = 0 \land |SN_i(j)| \geq 1 \).

**Theorem 1** (No deadlock) In any illegitimate configuration, there is always a node that is privileged.

**Proof:** Suppose the system is in an illegitimate configuration. Then \( \exists i \in V \) s.t. \(|SN(i)| \geq 1 \lor q.i = 1 \lor a.i \neq 0 \). We prove the theorem by cases.

**Case 1:** \( \forall v \in V, |SN(v)| = 0 \).

1.1 \( \exists i \in V \) s.t. \( q.i = 1 \). Then \( i \) can execute \( R9 \).

1.2 \( \forall v \in V, q.v = 0 \). Then \( \exists i \in V \) s.t. \( a.i \neq 0 \), and hence \( i \) can execute \( R10 \).

**Case 2:** \( \exists v \in V \) s.t. \(|SN(v)| \geq 1 \). Then, by Lemma 8, \( \exists i \in V \) s.t. \(|SN_i(i)| = 0 \land |SN_i(j)| \geq 1 \), and hence \( SN_i(i) = SN_i(j) \).

2.1 \( |SN_i(j)| = |SN_i(i)| \geq 2 \).

2.1.1 \( \forall x \in SN_i, q.x = 0 \).

2.1.1.1 \( a.i = 0 \). Then \( i \) can execute \( R10 \).

2.1.1.2 \( a.i = 0 \). Let \( j \) be a node in \( SN_i(i) \). Then \( q.j = 0 \) and \( i \in SN_j(j) \), and hence \( |SN_j(j)| \geq 1 \).

2.1.1.2.1 \( |NC_j(j)| \leq 5 \). Then \( j \) can execute \( R1 \).

2.1.1.2.2 \( |NC_j(j)| = 6 \). If \( |NC_j(j)| \leq 5 \), then since \( |SN_j(j)| \geq 1 \), \( j \) can execute \( R2 \). Otherwise, \( j \) can execute \( R11 \).

2.1.1.2.2.1 \( |NC_j(j)| \geq 1 \). Then since \( a.i = q.j = 0 \), \( j \) can execute \( R3 \).

2.1.2 \( \exists j \in SN_i(i), q.j = 1 \). Then \( i \in SN_j(j) \), and hence \( |SN_j(j)| \geq 1 \).

2.1.2.1 \( |SN_j(j)| \geq 2 \).

2.1.2.1.1 \( |NC_j(j)| \leq 5 \). Then \( j \) can execute \( R1 \).

2.1.2.1.2 \( |NC_j(j)| = 6 \). If \( |NC_j(j)| \leq 5 \), then since \( |SN_j(j)| \geq 1 \), \( j \) can execute \( R2 \). Otherwise, \( j \) can execute \( R11 \).

2.1.2.2 \( |SN_j(j)| = 1 \). Then since \( 1 = |SN_j(j)| \geq |SN_j(j)| \geq 1 \), \( |SN_j(j)| = 1 \).

2.1.2.2.1 \( |NC_j(j)| \leq 5 \). In view of the condition for Case 2.1, we have that \( |SN_i(i)| \geq 2 \). Hence \( i \) can execute \( R1 \).

2.1.2.2.2 \( |NC_i(i)| = 6 \).

2.1.2.2.2.1 \( |NC_i(i)| \leq 5 \). In view of the conditions for Cases 2.1 and 2.1.2, we have that \( q.j = 1 \) and \( |SN_i(i)| = 1 \), respectively. Hence \( j \) can execute \( R7 \).

2.1.2.2.2.2 \( |NC_i(i)| = 6 \). If \( |NC_i(i)| \leq 5 \), then since \( |SN_i(i)| \geq 1 \), \( |SN_i(i)| = 1 \) and \( q.j = 1 \), we have that \( j \) can execute \( R8 \). Otherwise, \( j \) can execute \( R11 \).
2.2 \(|SN_0(i)| = |SN(i)| = 1\). Let \(j\) be the unique node in \(SN_j(i)\). Then \(i \in SN_j(i)\), and hence \(|SN_j(i)| \geq 1\).

2.2.1 \(|SN(j)| \geq 2\).

2.2.1.1 \(|NC(j)| \leq 5\). Then \(j\) can execute \(R1\).

2.2.1.2 \(|NC(j)| = 6\). If \(|NC(j)| \leq 5\), then since \(|SN_j(j)| \geq 1\), \(j\) can execute \(R2\). Otherwise, \(j\) can execute \(R11\).

2.2.2 \(|SN(j)| = 1\).

2.2.2.1 \(|NC(j)| \leq 5\).

2.2.2.1.1 \(q \neq 0 \land a_i = 0\). Then \(j\) can execute \(R3\).

2.2.2.1.2 \(q = 0 \land a_i \neq 0\). Then \(i\) can execute \(R10\).

2.2.2.1.3 \(q = 1 \land a_i \neq 0\). Then \(j\) can execute \(R6\) or \(R7\).

2.2.2.1.4 \(q = 1 \land a_i = 0\). In view of the condition for Case 2.2, we have that \(|SN(i)| = 1\). If \(|NC(i)| \leq 5\), then \(i\) can execute \(R4\). Otherwise, \(i\) can execute \(R5\).

2.2.2.2 \(|NC(j)| = 6 \land |NC(i)| \leq 6\).

2.2.2.2.1 \(q = 0 \land a_j = 0\). Then since \(|SN(i)| = 1\), \(i\) can execute \(R3\).

2.2.2.2.2 \(q = 0 \land a_j \neq 0\). Then \(j\) can execute \(R10\).

2.2.2.2.3 \(q = 1 \land a_j \neq 0\). In view of the condition for Case 2.2, we have that \(|SN(j)| = 1\). Hence \(i\) can execute \(R6\) or \(R7\).

2.2.2.2.4 \(q = 1 \land a_j = 0\). In view of the condition for Case 2.2.2, we have that \(|SN(j)| = 1\). Hence \(j\) can execute \(R5\).

2.2.2.3 \(|NC(j)| = 6 \land |NC(i)| = 6\).

2.2.2.3.1 \(q = 0 \land a_i = 0\). Then since \(|SN(j)| = 1\), \(j\) can execute \(R3\).

2.2.2.3.2 \(q = 0 \land a_i \neq 0\). Then \(i\) can execute \(R10\).

2.2.2.3.3 \(q = 1 \land a_i \neq 2\). In view of the condition for Case 2.2.2, we have that \(|SN(i)| = 1\). Hence \(i\) can execute \(R5\).

2.2.2.3.4 \(q = 1 \land a_i = 2\). If \(|NC(j)| \leq 5\), then since \(|SN(j)| \geq 1\) and \(|SN(j)| = 1\), \(j\) can execute \(R8\). Otherwise, \(j\) can execute \(R11\).

\[\square\]

**Corollary 1** If \(\gamma\) is a configuration in which no node is privileged, then \(\gamma\) is a legitimate configuration.

**Theorem 2** (Self-stabilization) Algorithm 2 is self-stabilizing.

**Proof:** We first show that any execution of Algorithm 2 is finite. Suppose not. Then there exists an infinite execution \(\Gamma = (\gamma_1, \gamma_2, \ldots)\) of Algorithm 2. If there are infinitely many moves in \(\Gamma\) that execute \(R1, R2, R6, R7, R8\) or \(R11\), let \(\gamma_1, \gamma_2, \ldots\) where \(i < i_2 < \ldots\), be all the configurations in \(\Gamma\) which result from these moves. Then, if we ignore all the \(q\)-values and \(a\)-values in these configurations, then \((\gamma_1, \gamma_2, \gamma_3, \ldots)\) becomes an infinite execution of Algorithm 1, which is impossible in view of Lemma A4. Hence there are only a finite number of moves in \(\Gamma\) that execute \(R1, R2, R6, R7, R8\) or \(R11\). Therefore, there exists a positive integer \(m\) such that in the suffix \(\Gamma' = (\gamma_m, \gamma_{m+1}, \ldots)\) of \(\Gamma\), there is no move that executes \(R1, R2, R6, R7, R8\) or \(R11\). So in \(\Gamma'\), \(c\)-values and \(x\)-values of all nodes never change.

Consider any node \(i\). Since \(|SN(i)|\) depends only on the \(c\)-values of node \(i\) and its neighbors, \(|SN(i)|\) remains fixed in \(\Gamma'\). If \(|SN(i)| \geq 1\) in \(\Gamma'\), then \(i\) can never execute \(R9\) in \(\Gamma'\), and hence \(i\) can execute \(R3\) at most once in \(\Gamma'\). If \(|SN(i)| = 0\) in \(\Gamma'\), then \(i\) can never exe-
cute $R3$ in $\Gamma'$, and hence $i$ can execute $R9$ at most once in $\Gamma'$. Thus the system executes $R3$ and $R9$ only a finite number of times in $\Gamma'$. So there exists an integer $l > m$ such that in the suffix $\Gamma'' = (\gamma_l, \gamma_{l+1}, \ldots)$ of $\Gamma'$, there is no move that executes $R1, R2, R3, R6, R7, R8, R9$ or $R11$. So in $\Gamma''$, $c$-values, $x$-values and $q$-values never change and there are only moves that execute of $R4, R5$ or $R10$ in $\Gamma''$.

Consider any node $i$. If $i$ ever executes $R10$ in $\Gamma''$, then since $\forall j \in N(i), q.j = 0$ in $\Gamma''$, $i$ can never execute $R4$ or $R5$ to change its $a$-value back to 1 or 2 in $\Gamma''$ and, thus $i$ can never execute $R10$ again in $\Gamma''$. Therefore $i$ executes $R10$ at most once in $\Gamma''$. Similarly, $i$ executes each of $R4$ or $R5$ at most once in $\Gamma''$. So in $\Gamma''$, the system executes $R4, R5$ or $R10$ only a finite number of times. Therefore $\Gamma''$ is finite, which causes a contradiction.

Hence we have shown that any execution of Algorithm 2 is finite. By the definition of a finite execution, no node is privileged in the last configuration of any finite execution. Thus by Corollary 1, the last configuration of any finite execution of Algorithm 2 is a legitimate configuration. Therefore Algorithm 2 is self-stabilizing.

In the rest of the paper, we assume that the system starts in a legitimate configuration and incurs a single fault at a certain time instant $t_0$, with the faulty node being $u$. As defined previously, if $c.u$ changes at $t_0$ such that $|SN(u)| \geq 1$ at $t_0^+$, then the single fault at $t_0$ is called a nontrivial single fault; otherwise, it is called a trivial single fault.

**Lemma 9** If the fault at $t_0$ is a trivial single fault, then the system never changes any $c$-value in $(t_0, \infty)$.

**Proof:** Since the fault at $t_0$ is a trivial single fault, $|SN(u)| = 0$ at $t_0^+$. Hence for any $v \in V, |SN(v)| = 0$ at $t_0^+$. Let the first move of the system be made at $t_1$. Then for any $v \in V, |SN(v)| = 0$ at $t_1^+$. Hence the move at $t_1$ cannot be a move by $R1, R2, R6, R7$ or $R8$, and hence for any $v \in V$, $c.v$ does not change at $t_1$. It follows that for any $v \in V, |SN(v)| = 0$ at $t_1^+$. By the same token, we can show that any move after $t_0$ cannot be a move by $R1, R2, R6, R7$ or $R8$, that for any $v \in V$, $c.v$ never changes in $(t_0, \infty)$, and that for any $v \in V, |SN(v)| = 0$ in $(t_0, \infty)$.

**Lemma 10** If the fault at $t_0$ is a nontrivial single fault, then the system changes the $c$-value exactly once in $(t_0, \infty)$.

**Proof:** Since the fault at $t_0$ is a nontrivial single fault, $|SN(u)| \geq 1$ at $t_0^+$. By the self-stabilization property (i.e., Theorem 2), the system must change the $c$-value of at least one node in $(t_0, \infty)$. Let $t_1$ be the first time instant in $(t_0, \infty)$ at which the system changes the $c$-value of a certain node. Then

(i) for any $v \in V$, $c.v$ remains fixed in $(t_0, t_1)$.

Since $SN(v)$ depends only on the $c$-values of $v$ and its neighbors, we have that

(ii) for any $v \in V, NC(v)$ and $SN(v)$ remain fixed in $(t_0, t_1)$, and

(iii) for any $v \in V - N(u) \cup \{u\}, |SN(v)| = 0$ at $t_1^-$.

It follows that

(iv) for any $v \in V - N(u) \cup \{u\}, |SN(v)| = 0$ at $t_1^-$.

Let $i$ be the node that changes its $c$-value at $t_1$. In view of Algorithm 2, $i$ must execute $R1, R2, R6, R7$ or $R8$ at $t_1$. Hence $|SN(i)| \geq 1$ at $t_1^-$. In view of (iv) above, $i \in N(u) \cup \{u\}$.
Case 1: $i = u$. Then for any $v \in V$, $|SN(v)| = 0$ at $t_1^-$. By the same argument as in the proof of Lemma 9, we can show that for any $v \in V$, $c.v$ never changes in $(t_1, \infty)$.

Case 2: $i \in N(u)$. Since $|SN(i)| = 0$ at $t_0^-$ and for any $v \in V - \{u\}$, $c.v$ does not change at $t_0$, we have that $|SN(i)| = 0$ or 1 at $t_0^-$. In view of (ii) above, $|SN(i)| = 0$ or 1 at $t_1^-$. Hence $i$ must execute $R6$, $R7$ or $R8$ at $t_1$. It follows that $|SN(i)| = 1$ in $t_1^-$, where $j$ is the unique node in $SN(i)$ at $t_1^-$. Since $q.i = 0$ at $t_0^-$, $i$ must execute $R3$ at some instant $t_2$ in $(t_0, t_1)$. Thus $a.j = 0$ at $t_2$. Since $j \in SN(i)$ at $t_1^-$, $c.j = c.j$ at $t_1^-$. In view of (i) above, $c.i = c.j$ at $t_0^-$. Since $c.i \neq c.j$ at $t_0^-$ and $i \neq u$, we have that $j = u$. Consequently, $c.i = c.u$ at $t_0^-$, $a.u = 0$ at $t_2$ and $a.u \neq 0$ at $t_1^-$. Hence $u$ must execute $R4$ or $R5$ at some instant $t_3$ in $(t_2, t_1)$. Since $|NC(u)| \leq 5$ at $t_0^-$ and $NC(u)$ is independent of $c.u$, we have that $|NC(u)| \leq 5$ at $t_0^-$. In view of (ii) above, $|NC(u)| \leq 5$ at $t_1^-$, so $u$ must execute $R4$ at $t_3$. It follows that $|SN(u)| = 1$ at $t_1^-$. In view of (ii) above, $|SN(u)| = 1$ at $t_0^-$.

From the above in this subcase, we have that $|SN(i)| = 1$, $|SN(u)| = 1$ and $c.i = c.u$ at $t_0^-$. This, together with the fact that for every two adjacent nodes $x$ and $y$, $c.x = c.y$ at $t_0^-$, implies that for every two adjacent nodes $x$ and $y$, except $i$ and $u$, $c.x \neq c.y$ at $t_0^-$. Thus, in view of (i) above, for every two adjacent nodes $x$ and $y$, except $i$ and $u$, $c.x \neq c.y$ at $t_0^-$. Since $i$ executes $R6$, $R7$ or $R8$ at $t_1$, we get that for every two adjacent nodes $x$ and $y$, $c.x \neq c.y$ at $t_1^-$. Thus for every $v \in V$, $|SN(v)| = 0$ at $t_1^-$. By the same argument as in the proof of Lemma 9, we can show that for every $v \in V$, $c.v$ never changes in $(t_1, \infty)$. Therefore the lemma is proved.

Lemmas 9 and 10 combined give us the following theorem:

**Theorem 3 (Fault containment)** If the fault at $t_0$ is a single fault, then the system changes the $c$-value at most once in $(t_0, \infty)$.

### 4. Stabilization Times for Single-Fault Situations

We now demonstrate the efficiency of the proposed algorithm for single-fault situations.

**Lemma 11** If the fault at $t_0$ is a trivial single fault, then the system makes at most 2 moves in $(t_0, \infty)$.

**Proof:** Since the fault at $t_0$ is a trivial single fault, $|SN(u)| = 0$ at $t_0^-$. Hence $\forall v \in V$, $|SN(v)| = 0$ at $t_0^-$. By Lemma 9, $\forall v \in V$, $c.v$ never changes in $(t_0, \infty)$. Thus $\forall v \in V$, $|SN(v)| = 0$ in $(t_0, \infty)$. It follows that no node in $V$ can execute $R1$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$ or $R8$ at $(t_0, \infty)$. Since $u$ never executes $R3$ in $(t_0, \infty)$, $u$ executes $R9$ at most once in $(t_0, \infty)$. Similarly, since $u$ never executes $R4$ or $R5$ in $(t_0, \infty)$, $u$ executes $R10$ at most once in $(t_0, \infty)$. For every $i \neq u$, since $q.i = 0$ and $a.i = 0$ at $t_0^-$ and $i$ never executes $R3$, $R4$ or $R5$ in $(t_0, \infty)$, we have that $q.i = 0$ and $a.i = 0$ in $(t_0, \infty)$. Thus $i$ can never execute $R9$ or $R10$ in $(t_0, \infty)$. Since $\forall v \in V$, $|SN(v)| = 0$ in $(t_0, \infty)$, we have that $\forall v \in V$, $|NC(v)| \leq 5$ in $(t_0, \infty)$, so $\forall v \in V$, $|NC(v)| \leq 5$ in $(t_0, \infty)$. Thus no node in $V$ can execute $R11$ in $(t_0, \infty)$. Therefore the system makes at most 2 moves in $(t_0, \infty)$.  □
Lemma 12  If the fault at $t_0$ is a nontrivial single fault, then the system makes at most $5\Delta + 6$ moves in $(t_0, \infty)$, where $\Delta$ is the maximum node degree.

Proof: Since the fault at $t_0$ is a nontrivial single fault, $|SN(u)| \geq 1$ at $t_0$. By Lemma 10, there exists a unique instant $t_1 \in (t_0, \infty)$ at which the system changes the $c$-value. Hence,

(i) for any $v \in N(u) \cup \{u\}$, $c.v$ never changes in either $(t_0, t_1)$ or $(t_1, \infty)$,
(ii) for any $v \in V - N(u) \cup \{u\}$, $c.v$ never changes in $(t_0, \infty)$,
(iii) for any $v \in N(u) \cup \{u\}$, $NC(v)$ and $SN(v)$ remain fixed in both $(t_0, t_1)$ and $(t_1, \infty)$,
(iv) for any $v \in V - N(u) \cup \{u\}$, $NC(v)$ and $SN(v)$ remain fixed in $(t_0, \infty)$,
(v) $|SN(u)| \geq 1$ and $|NC(u)| \leq 5$ in $(t_0, t_1)$,
(vi) at any instant in $(t_0, t_1)$, for any $v \in SN(u)$, $|SN(v)| = 1$ and $u$ is the unique node in $SN(v)$,
(vii) for any $v \in V - SN(u) \cup \{u\}$ in $(t_0, t_1)$, $|SN(v)| = 0$ and $|NC(v)| \leq 5$ in $(t_0, \infty)$, and
(viii) for any $v \in V$, $|SN(v)| = 0$ and $|NC(v)| \leq 5$ in $(t_1, \infty)$.

In the following, our computation of stabilization time is divided into 4 parts.

(1) For the number of moves by $R_1$, $R_2$, $R_6$, $R_7$ or $R_8$:
Since the system changes the $c$-value exactly once in $(t_0, \infty)$, the system executes $R_1$, $R_2$, $R_6$, $R_7$ or $R_8$ exactly once in $(t_0, \infty)$.

(2) For the number of moves by $R_{11}$:
(2.1) Consider $i \in V - SN(u)$ at $t_0^+$.
In view of (v), (vii) and (viii) above, $|NC(i)| \leq 5$ in $(t_0, \infty)$. Hence $|NC, (i)| \leq 5$ in $(t_0, \infty)$, and hence $i$ can never execute $R_{11}$ in $(t_0, \infty)$.

(2.2) Consider $i \in SN(u)$ at $t_0^+$.
In view of (viii) above, $|NC(i)| \leq 5$ in $(t_1, \infty)$. Hence $|NC, (i)| \leq 5$ in $(t_1, \infty)$, and hence $i$ can never execute $R_{11}$ in $(t_1, \infty)$.

If $i$ executes $R_{11}$ at least twice in $(t_0, t_1)$, let $t_2$ be the first instant and $t_3$ be the second instant in $(t_0, t_1)$ at which $i$ executes $R_{11}$. Then $x.i$ never changes in $(t_2, t_3)$ and $x.i = 1 + \max_{j \in N(i)} x.j$ at $t_2^+$. Hence for every $j \in N(i) \setminus \{u\}$, $x.i > x.j$ at $t_2^+$. In view of the definition of legitimate configuration, $c.i = c.j$ at $t_0^+$, and hence $t_1^+$. Since $i \in SN(u)$ at $t_0^+$, $c.i = c.u$ at $t_0^+$. Hence $c.j \neq c.u$ at $t_0^+$. It follows that $j \in V - SN(u)$ at $t_0^+$. In view of (2.1) above, $j$ never executes $R_{11}$ to increase $x.j$ in $(t_2, t_3)$. Thus $x.i > x.j$ in $(t_2, t_3)$, and thus $j \in NC, (i)$ at $t_3^+$. Consequently, $|NC, (i)| \leq 1$ at $t_3^+$. Hence $i$ can never execute $R_{11}$ at $t_3^+$, a contradiction. Therefore $i$ executes $R_{11}$ at most once in $(t_0, t_1)$.

From all the above in (2.2), $i$ executes $R_{11}$ at most once in $(t_0, \infty)$.

From all the above in (2), the system executes $R_{11}$ at most $\Delta$ times in $(t_0, \infty)$.

(3) For the number of moves by $R_3$ or $R_9$:
(3.1) Consider $i \in V - SN(u) \cup \{u\}$ at $t_0^+$.
In view of (vii) above, $|SN(i)| = 0$ in $(t_0, \infty)$. Hence $i$ can never execute $R_3$ in $(t_0, \infty)$. This, together with the fact that $q.i = 0$ at $t_1^+$, implies that $q.i = 0$ in $(t_0, \infty)$. Thus $i$ can never execute $R_9$ in $(t_0, \infty)$. Consequently, $i$ never executes $R_3$ or $R_9$ in $(t_0, \infty)$.

(3.2) Consider $i \in SN(u) \cup \{u\}$ at $t_0^+$.
In view of (v) and (vi) above, $|SN(i)| \geq 1$ in $(t_0, t_1)$. Hence $i$ can never execute $R_9$ in $(t_0, t_1)$. It follows that $i$ executes $R_3$ at most once in $(t_0, t_1)$. In view of (viii) above, $|SN(i)|
= 0 in \((t_i, \infty)\). Thus \(i\) can never execute \(R3\) in \((t_i, \infty)\). It follows that \(i\) executes \(R9\) at most once in \((t_i, \infty)\). Consequently, \(i\) executes \(R3\) or \(R9\) at most twice in \((t_0, \infty)\).

From all the above in (3), the system executes \(R3\) or \(R9\) at most \(2(\Delta + 1)\) times in \((t_0, \infty)\).

(4) For the number of moves by \(R4, R5\) or \(R10:\)

(4.1) Consider \(i \in V - SN(u) \cup \{u\}\) at \(t_0^*\).

In view of (viii) above, \(|SN(i)| = 0\) in \((t_0, \infty)\). Hence \(i\) can never execute \(R4\) or \(R5\) in \((t_0, \infty)\). This, together with the fact that \(a.i = 0\) at \(t_0^*\), implies that \(a.i = 0\) in \((t_0, \infty)\). Thus \(i\) never executes \(R10\) in \((t_0, \infty)\). Consequently, \(i\) never executes \(R4, R5\) or \(R10\) in \((t_0, \infty)\).

(4.2) Consider \(i \in SN(u)\) at \(t_0^*\).

In view of (viii) above, \(|SN(i)| = 0\) in \((t_i, \infty)\). Hence \(i\) can never execute \(R4\) or \(R5\) in \((t_i, \infty)\), and hence \(i\) executes \(R10\) at most once in \((t_i, \infty)\). Therefore \(i\) executes \(R4, R5\) or \(R10\) at most once in \((t_i, \infty)\).

In view of (vi) above, \(|SN(i)| = 1\) in \((t_0, t_1)\) and \(u\) be the unique node in \(SN(i)\) in \((t_0, t_1)\). In view of (3.2) above, \(u\) executes \(R3\) at most once in \((t_0, t_1)\). Hence only three possibilities can exist: (i) \(q.u = 0\) in \((t_0, t_1)\), (ii) \(q.u = 1\) in \((t_0, t_1)\), or (iii) \(\exists t_2 \in (t_0, t_1)\) s.t. \(q.u = 0\) in \((t_0, t_2)\) and \(q.u = 1\) in \((t_2, t_1)\).

**Case 1:** \(q.u = 0\) in \((t_0, t_1)\). Then \(i\) can never execute \(R4\) or \(R5\) in \((t_0, t_1)\). Since \(a.i = 0\) at \(t_0^*\), \(a.i = 0\) in \((t_0, t_1)\). Hence \(i\) never executes \(R10\) in \((t_0, t_1)\). Consequently, \(i\) never executes \(R4, R5\) or \(R10\) in \((t_0, t_1)\).

**Case 2:** \(q.u = 1\) in \((t_0, t_1)\). Then \(i\) can never execute \(R10\) in \((t_0, t_1)\). In view of (iii) above, \(NC(i)\) remains fixed in \((t_0, t_1)\). If \(|NC(i)| \leq 5\) in \((t_0, t_1)\), then \(i\) can never execute \(R5\) in \((t_0, t_1)\), and hence \(i\) executes \(R4\) at most once in \((t_0, t_1)\). If \(|NC(i)| = 6\) in \((t_0, t_1)\), then \(i\) can never execute \(R4\) in \((t_0, t_1)\), and hence \(i\) executes \(R5\) at most once in \((t_0, t_1)\). Consequently, \(i\) executes \(R4, R5\) or \(R10\) at most once in \((t_0, t_1)\).

**Case 3:** \(\exists t_2 \in (t_0, t_1)\) s.t. \(q.u = 0\) in \((t_0, t_2)\) and \(q.u = 1\) in \((t_2, t_1)\). By the same token as in Case 1, we can prove that \(i\) never executes \(R4, R5\) or \(R10\) in \((t_0, t_2)\). By the same token as in Case 2, we can prove that \(i\) executes \(R4, R5\) or \(R10\) at most once in \((t_2, t_1)\). Consequently, \(i\) executes \(R4, R5\) or \(R10\) at most twice in \((t_0, \infty)\).

(4.3) Consider \(u\).

In view of (viii) above, \(|SN(u)| = 0\) in \((t_i, \infty)\). Hence \(u\) can never execute \(R4\) or \(R5\) in \((t_i, \infty)\), and hence \(u\) executes \(R10\) at most once in \((t_i, \infty)\). Therefore \(u\) executes \(R4, R5\) or \(R10\) at most once in \((t_i, \infty)\).

In view of (v) above, \(|SN(u)| \geq 1\) and \(|NC(u)| \leq 5\) in \((t_0, t_1)\). Hence \(u\) can never execute \(R5\) in \((t_0, t_1)\).

**Case 1:** \(|SN(u)| \geq 2\) in \((t_0, t_1)\). Then \(u\) can never execute \(R4\) in \((t_0, t_1)\). This, together with the fact that \(u\) never executes \(R5\) in \((t_0, t_1)\), implies that \(u\) executes \(R10\) at most once in \((t_0, t_1)\). Consequently, \(i\) executes \(R4, R5\) or \(R10\) at most once in \((t_0, t_1)\).

**Case 2:** \(|SN(u)| = 1\) in \((t_0, t_1)\). Let \(i\) be the unique node in \(SN(u)\) in \((t_0, t_1)\). In view of (3.2)
above, $i$ executes $R3$ at most once in $(t_0, t_1)$. Since $q.i = 0$ at $t_0^+$, only two possibilities can exist: (i) $q.i = 0$ in $(t_0, t_1)$, or (ii) $\exists t_2 \in (t_0, t_1)$ s.t. $q.i = 0$ in $(t_0, t_2)$ and $q.i = 1$ in $(t_2, t_1)$.

**Case 2.1:** $q.i = 0$ in $(t_0, t_1)$. Then $u$ can never execute $R4$ or $R5$ in $(t_0, t_1)$. Hence $u$ executes $R10$ at most once in $(t_0, t_1)$. Consequently, $i$ executes $R4$, $R5$ or $R10$ at most once in $(t_0, t_1)$.

**Case 2.2:** $\exists t_2 \in (t_0, t_1)$ s.t. $q.i = 0$ in $(t_0, t_2)$ and $q.i = 1$ in $(t_2, t_1)$. By the same token as in Case 2.1, we can prove that $i$ executes $R4$, $R5$ or $R10$ at most once in $(t_0, t_2)$. Since $q.i = 1$ in $(t_2, t_1)$, $u$ can never execute $R10$ in $(t_2, t_1)$. This, together with the fact that $u$ never executes $R5$ in $(t_2, t_1)$, implies that $u$ executes $R4$ at most once in $(t_2, t_1)$. Consequently, $u$ executes $R4$, $R5$ or $R10$ at most twice in $(t_0, t_1)$.

From all the above in (4.3), $u$ executes $R4$, $R5$ or $R10$ at most 3 times in $(t_0, \infty)$.

From all the above in (4), the system executes $R4$, $R5$ or $R10$ at most $2\Delta + 3$ times in $(t_0, \infty)$.

From all the above, the system makes at most $1 + \Delta + 2(\Delta + 1) + 2\Delta + 3 = 5\Delta + 6$ moves in $(t_0, \infty)$. Therefore the lemma is proved.

Lemmas 11 and 12 combined give us the following main result:

**Theorem 4** For single-fault situations, the worst-case stabilization time of Algorithm 2 is $O(\Delta)$, where $\Delta$ is the maximum node degree.

## 5. CONCLUDING REMARKS

In this paper, we have proposed a fault-containing self-stabilizing algorithm which can 6-color any planar graph. For single-fault situations, the worst-case stabilization time of the proposed algorithm is $O(\Delta)$, where $\Delta$ is the maximum node degree in the system.

Including the proposed algorithm in this paper, we have used the following approach to design three fault-containing self-stabilizing algorithms:

**Step 1:** Propose (or find) a non-fault-containing self-stabilizing algorithm, say Algorithm $A$.

**Step 2:** Classify the guard conditions of some rules of Algorithm $A$ into various cases, and then derive a prototype of our fault-containing self-stabilizing algorithm.

**Step 3:** Transform the prototype into a distributed algorithm, say Algorithm $B$.

**Step 4:** If Algorithm $B$ can be proved to be a fault-containing self-stabilizing algorithm, then go to step 5. Otherwise, go back to step 1, 2 or 3, depending on how serious the problem is.

**Step 5:** Compute the worst-case stabilization time of Algorithm $B$ for single-fault situations.

In our future project, we will try to use this approach to find a transformer which can convert any non-fault-containing self-stabilizing algorithm to a fault-containing self-
stabilizing algorithm such that the worst-case stabilization time of the transformed algorithm for single fault situations is $O(\Delta)$.

REFERENCES


APPENDIX 1. A PROOF OF LEMMA 1

Let us consider the system that is equipped with Algorithm 1. For the sake of presentation, the three sets $N_p(j)$, $SN_s(j)$ and $SN_p(j)$ in a configuration $\gamma$ are denoted by $N_p(\gamma, j)$, $SN_s(\gamma, j)$ and $SN_p(\gamma, j)$, respectively. In order to prove Lemma 1 (the self-stabilization property of Algorithm 1), we define a positive integer-valued function $F$ as follows:

For any configuration $\gamma$ and any node $j$, we define $f(\gamma, j) = \sum_{k \in N_p(\gamma, j)} f(\gamma, k)$.

For any configuration $\gamma$, we define $F(\gamma) = \sum f(\gamma, j) \cdot |SN_p(\gamma, j)|$.

Note that if $N_p(\gamma, j) = \emptyset$, then $\sum f(\gamma, k)$ is defined to be 0. Hence if $N_p(\gamma, j) = \emptyset$, then $f(\gamma, j) = 1$. It is obvious that for any configuration $\gamma$ and any node $j$, $f(\gamma, j) \geq 1$. Note also that the function $f$ is well defined because if $\gamma$ is a configuration and $j_1, j_2, j_3, \ldots, j_m$ are nodes such that $j_1 \in N_p(\gamma, j_2), j_2 \in N_p(\gamma, j_3), \ldots, j_{m-1} \in N_p(\gamma, j_m)$ and $j_m \in N(j_1)$, then $j_1 \in N_p(\gamma, j_m)$ and $j_m \notin N_p(\gamma, j_1)$.

Lemma A1 If $\gamma \rightarrow \gamma'$ is an R1 move, then $F(\gamma') \leq F(\gamma) - 1$.

Proof: Let $i$ be the node that executes R1 in the move $\gamma \rightarrow \gamma'$. Then $c.i \notin NC(i)$ in $\gamma'$. Hence, for every $j \in N(i)$, $c.j \neq c.i$ in $\gamma'$, and hence $|SN_p(\gamma', i)| = 0$. For every $j \neq i$,

1. if $j \in SN_p(\gamma, i)$, then since $c.j = c.i$ in $\gamma$ and $c.j \neq c.i$ in $\gamma'$, we have $|SN_p(\gamma', j)| = |SN_p(\gamma, j)| - 1$;
2. if $j \in N_p(\gamma, i) - SN_p(\gamma, i)$, then since $c.j \neq c.i$ in both $\gamma$ and $\gamma'$, we have $|SN_p(\gamma', j)| = |SN_p(\gamma, j)|$;
3. if $j \in V - \{i\} \cup N_p(\gamma, i)$, then since the $c$-values of $j$ and all its successors do not change in the move $\gamma \rightarrow \gamma'$, we have $|SN_p(\gamma', j)| = |SN_p(\gamma, j)|$.

Note that $f(\gamma', j) = f(\gamma, j)$ and $N_p(\gamma', j) = N_p(\gamma, j)$. So we can denote them by $f(j)$ and $N_p(j)$, respectively. From (2) and (3) above, we have that for every $j \in V - \{i\} \cup SN_p(\gamma, i)$, $|SN_p(\gamma', j)| = |SN_p(\gamma, j)|$. Thus,
\[ F(\gamma) = F(\gamma') \]
\[ = \sum_{j \in V} f(\gamma, j) \cdot |SN_{\gamma}(\gamma, j)| - \sum_{j \in V} f(\gamma', j) \cdot |SN_{\gamma'}(\gamma', j)| \]
\[ = [f(\gamma, i) \cdot |SN_{\gamma}(\gamma, i)| + \sum_{j \in N_{\gamma}(\gamma, i)} f(\gamma, j) \cdot |SN_{\gamma}(\gamma, j)| + \sum_{j \in V - \{i\} \cup N_{\gamma}(\gamma, i)} f(\gamma, j) \cdot |SN_{\gamma}(\gamma, j)|] - \]
\[ [f(\gamma', i) \cdot |SN_{\gamma'}(\gamma', i)| + \sum_{j \in N_{\gamma'}(\gamma', i)} f(\gamma', j) \cdot |SN_{\gamma'}(\gamma', j)| + \sum_{j \in V - \{i\} \cup N_{\gamma'}(\gamma', i)} f(\gamma', j) \cdot |SN_{\gamma'}(\gamma', j)|] \]
\[ = f(\gamma, i) \cdot |SN_{\gamma}(\gamma, i)| + \sum_{j \in N_{\gamma}(\gamma, i)} f(j) \cdot |SN_{\gamma}(\gamma, j)| + \sum_{j \in V - \{i\} \cup N_{\gamma}(\gamma, i)} f(j) \cdot 0 \geq \]
\[ f(i) + \sum_{j \in N_{\gamma}(\gamma, i)} f(j) \cdot (-1) \]
\[ \geq f(i) - \sum_{j \in N_{\gamma}(\gamma, i)} f(j) \]

**Lemma A2** If \( \gamma \to \gamma' \) is an \( R2 \) move, then \( F(\gamma') \leq F(\gamma) - 1 \).

**Proof:** Let \( i \) be the node that executes \( R2 \) in the move \( \gamma \to \gamma' \). Then \( c.i \notin NC(\gamma', i) \) in \( \gamma' \).

Hence, for every \( j \in N_{\gamma}(\gamma, i) \), \( c.j \neq c.i \) in \( \gamma' \), and hence \( |SN_{\gamma'}(\gamma', i)| = 0 \). For every \( j \neq i \),

1. if \( j \in SN_{\gamma}(\gamma, i) \) and \( c.j = c.i \) in \( \gamma' \), then \( |SN_{\gamma'}(\gamma', j)| = |SN_{\gamma}(\gamma, j)| \);
2. if \( j \in SN_{\gamma}(\gamma, i) \) and \( c.j \neq c.i \) in \( \gamma' \), then \( |SN_{\gamma'}(\gamma', j)| = |SN_{\gamma}(\gamma, j)| - 1 \);
3. if \( j \in N_{\gamma}(\gamma, i) \) - \( SN_{\gamma}(\gamma, i) \) and \( c.j = c.i \) in \( \gamma' \), then \( |SN_{\gamma'}(\gamma', j)| = |SN_{\gamma}(\gamma, j)| + 1 \);
4. if \( j \in N_{\gamma}(\gamma, i) \) - \( SN_{\gamma}(\gamma, i) \) and \( c.j \neq c.i \) in \( \gamma' \), then \( |SN_{\gamma'}(\gamma', j)| = |SN_{\gamma}(\gamma, j)| \);
5. if \( j \in V - \{i\} \cup N_{\gamma}(\gamma, i) \), since the \( c \)-values of \( j \) and all its successors do not change in \( t \), we have that \( |SN_{\gamma}(\gamma, j)| = |SN_{\gamma}(\gamma, j)| \).

Note that \( f(\gamma', j) = f(\gamma, j) \) and \( N_{\gamma}(\gamma', j) = N_{\gamma}(\gamma, j) \). So we can denote them by \( f(j) \) and \( N(j) \), respectively. From (1)-(4) above, we have that for every \( j \in N_{\gamma}(\gamma, i) \), \( |SN_{\gamma}(\gamma, j)| - |SN_{\gamma'}(\gamma', j)| \geq -1 \). Thus,

\[ F(\gamma) - F(\gamma') \]
\[ = \sum_{j \in V} f(\gamma, j) \cdot |SN_{\gamma}(\gamma, j)| - \sum_{j \in V} f(\gamma', j) \cdot |SN_{\gamma'}(\gamma', j)| \]
\[ = [f(\gamma, i) \cdot |SN_{\gamma}(\gamma, i)| + \sum_{j \in N_{\gamma}(\gamma, i)} f(\gamma, j) \cdot |SN_{\gamma}(\gamma, j)| + \sum_{j \in V - \{i\} \cup N_{\gamma}(\gamma, i)} f(\gamma, j) \cdot |SN_{\gamma}(\gamma, j)|] - \]
\[ [f(\gamma', i) \cdot |SN_{\gamma'}(\gamma', i)| + \sum_{j \in N_{\gamma'}(\gamma', i)} f(\gamma', j) \cdot |SN_{\gamma'}(\gamma', j)| + \sum_{j \in V - \{i\} \cup N_{\gamma'}(\gamma', i)} f(\gamma', j) \cdot |SN_{\gamma'}(\gamma', j)|] \]
\[ = f(\gamma, i) \cdot |SN_{\gamma}(\gamma, i)| + \sum_{j \in N_{\gamma}(\gamma, i)} f(j) \cdot |SN_{\gamma}(\gamma, j)| + \sum_{j \in V - \{i\} \cup N_{\gamma}(\gamma, i)} f(j) \cdot 0 \geq \]
\[ f(i) + \sum_{j \in N_{\gamma}(\gamma, i)} f(j) \cdot (-1) \]
\[ \geq f(i) - \sum_{j \in N_{\gamma}(\gamma, i)} f(j) \]
Lemma A3  Any execution \( \Gamma \) of Algorithm 1 contains a finite number of \( R3 \) moves.

Proof: Suppose that \( \Gamma \) contains infinitely many \( R3 \) moves. Then there exists a node \( i \) in \( V \) such that \( i \) executes \( R3 \) infinitely many times in \( \Gamma \) (this property will be denoted by (*) in the following). Consider any such \( i \).

Claim  There exist at least 6 neighbors of \( i \) each of which executes \( R3 \) infinitely many times in \( \Gamma \).

Proof of Claim: Suppose that there are at most 5 neighbors of \( i \) each of which executes \( R3 \) infinitely many times in \( \Gamma \). Then there is a suffix \( \Gamma' \) of \( \Gamma \) such that at most 5 neighbors of \( i \) execute \( R3 \) in \( \Gamma' \). Let \( \gamma_j \rightarrow \gamma_{j+1} \) and \( \gamma_k \rightarrow \gamma_{k+1} \) be two consecutive \( R3 \) moves by \( i \) in \( \Gamma' \). Then all neighbors of \( i \) are predecessors of \( i \) in \( \gamma_{j+1} \) and at least 6 neighbors of \( i \) become successors of \( i \) in \( \gamma_k \). So at least 6 neighbors of \( i \) should execute \( R3 \) to change their \( x \)-values in \((\gamma_{j+1}, \gamma_{j+2}, \ldots, \gamma_k)\), a contradiction. Therefore there are at least 6 neighbors of \( i \) each of which executes \( R3 \) infinitely many times in \( \Gamma \).

Let \( V' = \{ j \in V | j \text{ executes } R3 \text{ infinitely many times in } \Gamma \} \). Then, from (*) above, we have \( V' \neq \phi \). Let \( G' = (V', E') \) be the subgraph of \( G = (V, E) \) induced from \( V' \) (i.e. edge \{a, b\} \in E' \iff \{a, b\} \in E \text{ and } a, b \in V' \). Then \( G' \) is a planar graph (since \( G \) is planar) and each node has a degree of at least 6 (from the claim above). This leads to a contradiction, since in any planar graph, there exists a node with a degree of at most 5. Therefore the lemma is proved.

Lemma A4  Any execution \( \Gamma \) of Algorithm 1 is finite.

Proof: By Lemma A3, there is a suffix \( \Gamma' \) of \( \Gamma \) such that there is no \( R3 \) move in \( \Gamma' \). Then any move in \( \Gamma' \) is either an \( R1 \) move or an \( R2 \) move. This together with Lemmas A1 and A2 implies that each move in \( \Gamma' \) causes the decrease of the value of the function \( F \) by at least 1. Since \( F \) is bounded below by 0, we have that \( \Gamma' \) contains a finite number of moves. Hence \( \Gamma \) also contains a finite number of moves.

Lemma A5  If \( \gamma \) is a configuration in which no node is privileged, then \( \gamma \) is a legitimate configuration.

Proof: Suppose that \( \gamma \) is not a legitimate configuration. Then there is a node \( i \) such that \( |SN(i)| \geq 1 \) in \( \gamma \). If \( |SN(i)| = 0 \) in \( \gamma \), then \( |SN(j)| \geq 1 \) in \( \gamma \). Let \( j \) be a node in \( SN(i) \) in \( \gamma \). Then \( |SN(j)| \geq 1 \) in \( \gamma \). Hence we have shown that there must be a node \( v \) such that \( |SN(v)| \geq 1 \) in \( \gamma \). If \( |NC(v)| \leq 5 \) in \( \gamma \), then \( v \) is privileged by \( R1 \) in \( \gamma \) if \( |NC(v)| = 6 \) in \( \gamma \), then \( v \) is privileged by \( R3 \) in \( \gamma \) and if \( |NC(v)| = 6 \) and \( |NC(v)| \leq 5 \) in \( \gamma \), then \( v \) is privileged by \( R2 \) in \( \gamma \). Thus, in any case, \( v \) is privileged in \( \gamma \), which causes a contradiction. Therefore \( \gamma \) is a legitimate configuration.
Proof of Lemma 1: By Lemma A4, any execution of Algorithm 1 is finite. By the definition of a finite execution, no node is privileged in the last configuration of any finite execution. Thus by Lemma A5, the last configuration of any finite execution is a legitimate configuration. Therefore Algorithm 1 is self-stabilizing.

Ji-Cherng Lin (林基成) received his B.S. degree in Mathematics from National Tsing Hua University, Taiwan in 1983, and M.S. and Ph.D. degrees in Computer Science from National Tsing Hua University, Taiwan in 1985 and 1991, respectively. He is now an Associate Professor of the Department of Computer Science and Engineering of Yuan Ze University, Taiwan. His current research interest is focused mainly on self-stabilizing algorithms.

Ming-Yi Chiu (邱明義) received his M.S. degree in Computer Science and Engineering from Yuan Ze University, Taiwan in 2001. He is now pursuing a Ph.D. degree in the Department of Computer Science and Engineering of Yuan Ze University, Taiwan. His research interest is in the area of self-stabilizing algorithms.