



ELSEVIER

Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Information Processing Letters 87 (2003) 251–255

Information
Processing
Letters

www.elsevier.com/locate/ipl

Linear time self-stabilizing colorings

Stephen T. Hedetniemi, David P. Jacobs, Pradip K. Srimani *

Department of Computer Science, Clemson University, Clemson, SC 29634-0974, USA

Received 7 August 2002; received in revised form 21 March 2003

Communicated by A.A. Bertossi

Abstract

We propose two new self-stabilizing distributed algorithms for proper $\Delta + 1$ (Δ is the maximum degree of a node in the graph) colorings of arbitrary system graphs. Both algorithms are capable of working with multiple type of daemons (schedulers) as is the most recent algorithm by Gradinariu and Tixeuil [OPODIS'2000, 2000, pp. 55–70]. The first algorithm converges in $O(m)$ moves while the second converges in at most n moves (n is the number of nodes and m is the number of edges in the graph) as opposed to the $O(\Delta \times n)$ moves required by the algorithm by Gradinariu and Tixeuil [OPODIS'2000, 2000, pp. 55–70]. The second improvement is that neither of the proposed algorithms requires each node to have knowledge of Δ , as is required by Gradinariu and Tixeuil [OPODIS'2000, 2000, pp. 55–70]. Further, the coloring produced by our first algorithm provides an interesting type of coloring, called a Grundy Coloring [Jensen and Toft, Graph Coloring Problems, 1995].

© 2003 Elsevier B.V. All rights reserved.

Keywords: Algorithms; Distributed systems; Self-stabilizing coloring; Grundy coloring

1. Introduction

In a distributed system the computing elements or nodes exchange information only by message passing. Every node has a set of local variables whose contents specify the local state of the node. The state of the entire system, called its *global state*, is the union of the local states of all the nodes. Each node is allowed to have only a partial view of the global state, and this depends on the connectivity of the system and the propagation delay of different messages. Yet, the objective in a distributed system is to arrive at a desir-

able global final state, or legitimate state. One of the goals of a distributed system is to function correctly, i.e., the global state of the system should remain legitimate in presence of faults (transient). Often, malfunctions or perturbations bring the system to some illegitimate state, and it is desirable that the system be automatically brought back to a desired legitimate state. *Self-stabilization*, introduced by Dijkstra [4], is the most inclusive approach to fault tolerance in distributed systems that brings the system back to a legitimate state starting from any illegitimate state (caused by any transient fault) without any intervention by an external agent. In a self-stabilizing algorithm, each node maintains its local variables, and can make decisions based on the knowledge of its neighbors' states.

In a self-stabilizing algorithm, a node may change its local state by making a *move* (specification of

* Corresponding author.

E-mail addresses: hedet@cs.clemson.edu (S.T. Hedetniemi), dpj@cs.clemson.edu (D.P. Jacobs), srimani@cs.clemson.edu (P.K. Srimani).

an action which causes a change of local state). Algorithms are given as a set of rules of the form “if $p(i)$ then M ”, where $p(i)$ is a predicate and M is a move. A node i becomes *privileged* if $p(i)$ is true. When a node becomes privileged, it may execute the corresponding move. We assume a serial model in which no two nodes move simultaneously. A central daemon selects, among all privileged nodes, the next node to move. If two or more nodes are privileged, we cannot predict which node will move next. Multiple protocols exist [12,1,2] that provide such a scheduler. Our algorithms can easily be combined with any of these protocols to work under different schedulers as well. An execution will be represented as a sequence of moves M_1, M_2, \dots , in which M_s denotes the s th move. The system’s initial state is denoted by s_0 , and for $t > 0$, the state resulting from M_t is denoted by s_t .

In this paper we propose two simple, yet very efficient, self-stabilizing algorithms that find proper colorings in an arbitrary graph. Self-stabilizing algorithms for proper colorings of graphs have been studied in the literature [6,15,14,13,9]. For example, Sur and Sriamani [15] give a self-stabilizing algorithm to 2-color any bipartite graph, and in [6], Ghosh and Karaata describe a self-stabilizing algorithm to 6-color any planar graph. Obviously the bipartite algorithm is optimal and the planar algorithm is close to optimal, given that all planar graphs are 4-colorable. But, the authors do not provide any complexity analysis. Only a recent paper of Gradinariu and Tixeuil [9] gives a self-stabilizing algorithm that finds a $(\Delta + 1)$ -coloring in any graph, and which makes $O(\Delta \times n)$ moves. One drawback of this algorithm is that each node must know the value of Δ . Our first $(\Delta + 1)$ -coloring algorithm, Algorithm 2.1, has three advantages:

- (1) no node must know the value of Δ (as opposed to the requirement in the algorithm of [9]);
- (2) the algorithm converges in $O(m)$ moves (m is the number of edges in the graph) compared to $O(\Delta \times n)$ moves of [9]; and
- (3) the coloring obtained by the algorithm is always a Grundy coloring [10] of the graph.

We then propose another $(\Delta + 1)$ -coloring algorithm, Algorithm 2.2; this algorithm stabilizes in at most n moves. Algorithm 2.2 appears to be faster than all

other known self-stabilizing $(\Delta + 1)$ -coloring algorithms.

2. Self-stabilizing coloring algorithms

We model a distributed system with an undirected connected graph $G = (V, E)$, where the node set V represents the set of processors, and the edge set E represents the processor interconnections. Throughout this paper we assume $|V| = n$ and $|E| = m$. If i is a node, then $N(i)$, its *open neighborhood*, denotes the set of nodes to which i is adjacent. Every node $j \in N(i)$ is called a *neighbor* of node i . We let $d_i = |N(i)|$, the number of neighbors of node i , or its *degree*, and we let $\Delta = \max\{d_i \mid i \in V\}$.

Given a graph $G = (V, E)$, a k -coloring is a function $c: V \rightarrow \{1, 2, \dots, k\}$, where the elements in the range are called *colors*. A node i is *properly colored* if for all $j \in N(i)$, $c(i) \neq c(j)$, and c is a *proper coloring* if all nodes are properly colored. A graph G can be properly colored with k colors if and only if its node set can be partitioned into k pairwise disjoint independent sets. The minimum number of colors needed to properly color a graph G is called its *chromatic number*, denoted $\chi(G)$, and is, in general, NP-hard to compute [5,11].

2.1. Grundy colorings

Given a coloring $c: V \rightarrow \{1, 2, \dots, k\}$, a node i is called a *Grundy node* if

$$c(i) = \min\{\ell \geq 1 \mid (\forall j \in N(i))(c(j) \neq \ell)\}.$$

That is, i is colored with the smallest color not taken by any neighbor. Note that a Grundy node is by definition properly colored. If $c(i) = 1$, and if i is properly colored, then, trivially, it must be a Grundy node. A *Grundy coloring* is one in which every node is a Grundy node. While this idea seems to have originated in [8], a more recent discussion of Grundy colorings, with more references, can be found in [10].

It is shown in [10] that if a Grundy coloring uses k colors, then any node i , colored k , must have at least $k - 1$ neighbors colored $1, \dots, k - 1$ and hence the degree of the node i is at least $k - 1$.

$$\chi(G) \leq k \leq \Delta + 1.$$

R: if $c(i) \neq \min\{\ell \geq 1 \mid (\forall j \in N(i))(c(j) \neq \ell)\}$
 then set $c(i) = \min\{\ell \geq 1 \mid (\forall j \in N(i))(c(j) \neq \ell)\}$

Algorithm 2.1. GRUNDY COLORING().

However, it is known that the number of colors in a Grundy coloring can be arbitrarily larger than $\chi(G)$. For example, all trees can be 2-colored, yet for any positive integer k , there exists a tree (whose order is exponential in k) that can be Grundy colored with k colors.

Despite this potential worst case behavior, there are good reasons to seek Grundy colorings. First, any graph G has a Grundy coloring which uses $\chi(G)$ colors. And on average, a Grundy coloring does fairly well. It is known that for random graphs in which each node pair is assigned an edge with probability $\frac{1}{2}$, a Grundy coloring will use about twice as many colors as are necessary [7].

We propose a self-stabilizing algorithm, Algorithm 2.1, to produce a Grundy coloring for an arbitrary graph of order n . In this algorithm, each node i maintains a single integer variable $c(i)$, its color, where $1 \leq c(i) \leq d_i + 1$. Algorithm 2.1 has a single rule, namely if a node's color is different than the first positive integer not taken by any neighbor, then it chooses that color instead.

Note that $c(i)$ is changed whenever node i executes rule **R**. We say that a move is *increasing* if $c(i)$ increases, and *decreasing* otherwise. The following lemma is clear.

Lemma 1. *After any move made by node i , $1 \leq c(i) \leq d_i + 1$.*

A main idea in our analysis is to bound the number of decreasing moves that each node can make. In the next lemma, we speak of a node i making a sequence of *consecutive decreasing moves*, if there is no intermediate increasing move made by i , but there could be intermediate moves (either increasing or decreasing) made by other nodes.

Lemma 2. *A node i can make at most $d_i + 1$ consecutive decreasing moves.*

Proof. This follows from Lemma 1. \square

Lemma 3. *A node can make an increasing move M_t only if it is not properly colored in state s_{t-1} .*

Proof. Node i can execute an increasing move if and only if for each $k \in \{1, 2, \dots, c(i)\}$, there exists a $j \in N(i)$ with $c(j) = k$. In particular, i has a neighbor colored $c(i)$. \square

Lemma 4. *After node i executes rule **R**, it becomes properly colored and remains so.*

Proof. It is clear that by executing **R**, i becomes properly colored. That it remains in this condition follows from the fact that by executing rule **R**, no node can ever destroy the proper coloring of another node. \square

We mention that although the execution of rule **R** by node i cannot destroy the proper coloring of another node, it *can* destroy the Grundy coloring of another node.

Lemma 5. *Each node i can make at most one increasing move, and that can only occur on its first move.*

Proof. This follows from Lemmas 3 and 4. \square

Lemma 6. *Each node i can make at most $d_i + 1$ moves.*

Proof. If node i never makes an increasing move, then it can make at most $d_i + 1$ moves by Lemma 2. If it does make an increasing move, by Lemma 5 this occurs only once as its first move. By Lemma 1, after this first move, $1 \leq c(i) \leq d_i + 1$. Its remaining moves must be decreasing, and there can be at most d_i of these. \square

Theorem 1. *Given a graph with n nodes and m edges, Algorithm 2.1 finds a Grundy coloring in at most $n + 2m$ moves.*

Proof. Using Lemma 6 and summing over every i , we see that any sequence of moves can have length at most $\sum_{i=1}^n (d_i + 1) = n + 2m$. \square

Corollary 1. *For planar graphs, Algorithm 2.1 constructs a Grundy coloring in at most $7n - 12$ moves.*

R: if $c(i) \in \{c(j) \mid j \in N(i)\} \vee c(i) > d_i + 1$

then {

/* recolor node i */

if $\{c(j) \mid j \in N(i)\} = \{1, 2, \dots, d_i\}$

then $c(i) = d_i + 1$

else set $c(i) \in \{1, 2, \dots, d_i\} - \{c(j) \mid j \in N(i)\}$

Algorithm 2.2. FAST COLORING().

Proof. It is well known that for planar graphs, $m \leq 3n - 6$. \square

It should be noted that an algorithm, which at first glance appears similar to Algorithm 2.1, is given in [9], and has an accompanying worst case analysis of $O(\Delta n)$. This algorithm properly colors nodes with values in the range $\{0, 1, \dots, \Delta\}$, always choosing the *largest* possible color. Our algorithm and analysis have two advantages. First, the bound of $n + 2m$ steps is an improvement. And second, in Algorithm 2.1, nodes are not required to know Δ , as in [9], nor any other global property.

2.2. $(\Delta + 1)$ -coloring in n moves

Algorithm 2.1 constructs a Grundy coloring with at most $\Delta + 1$ colors in at most $n + 2m$ moves. In this section we present a simple algorithm, Algorithm 2.2, that also constructs a proper $(\Delta + 1)$ -coloring, but which stabilizes in at most n moves. This appears to be the first $O(n)$ self-stabilizing proper coloring algorithm.

The following lemma is self-evident.

Lemma 7. *After any move made by node i , $1 \leq c(i) \leq d_i + 1$.*

Lemma 8. *After node i executes rule **R**, it becomes properly colored and remains so.*

Proof. Clearly i becomes properly colored. It remains so because a move by another node cannot destroy this property. \square

Lemma 9. *Each node can move at most once.*

Proof. A node i is privileged if and only if it is not properly colored or $c(i) > d_i + 1$. By Lemmas 7 and 8, after moving, it cannot become privileged again. \square

Theorem 2. *For any graph with n nodes, Algorithm 2.2 finds a $(\Delta + 1)$ -coloring in at most n moves.*

Proof. By Lemma 9, each node will move at most once, and clearly this will stabilize in a proper coloring. This coloring must use at most $\Delta + 1$ colors, or else some node would be privileged. \square

Remark 1. The bound is tight. Consider a path P_n with n nodes, each of which is initially colored 1, is an example of a graph for which Algorithm 2.2 can make $n - 1$ moves, if the nodes move in order from left to right.

3. Conclusion

It is interesting to note that the set of nodes colored 1 by Algorithm 2.1 forms a maximal independent set. Unlike Algorithm 2.1, Algorithm 2.2 does not necessarily find a maximal independent set. Algorithm 2.2 is inspired by the well-known Brooks' theorem [3] which asserts that the chromatic number of a graph is at most $\Delta + 1$, and in fact is at most $k = \Delta$, unless $k = 2$ and G has a component which an odd cycle, or $n > 2$ and K_{n+1} is a component of G . It remains an interesting question whether one can construct a self-stabilizing algorithm for coloring a graph with at most Δ colors.

Note that in each case the upper bound on the maximum number of moves made by the algorithms in the worst case is tight. Since the algorithms are self-stabilizing and the fault pattern can be arbitrary, the number of moves made by the algorithms for a given fault pattern can be as low as 1 (lower bound). Also, for a given arbitrary graph, the number of colors needed by the algorithms can be as big as $\Delta + 1$ while the chromatic number of the graph can be as low as 2 (consider the binomial trees for example).

Acknowledgements

Srimani's work was partially supported by an NSF grant # ANI-0073409.

References

- [1] G. Antonoiu, P.K. Srimani, Mutual exclusion between neighboring nodes in an arbitrary system graph tree that stabilizes using read/write atomicity, in: Euro-Par'99—Parallel Processing, Proceedings, in: Lecture Notes in Comput. Sci., Vol. 1685, Springer, Berlin, 1999, pp. 823–830.
- [2] J. Beauquier, A.K. Datta, M. Gradinariu, F. Magniette, Self-stabilizing local mutual exclusion and daemon refinement, in: DISCOO Distributed Computing 14th International Symposium, in: Lecture Notes in Comput. Sci., Vol. 1914, Springer, Berlin, 2000.
- [3] R.L. Brooks, On coloring the nodes of a network, Proc. Cambridge Philos. Soc. 37 (1941) 197–227.
- [4] E.W. Dijkstra, Self-stabilizing systems in spite of distributed control, Comm. ACM 17 (11) (1974) 643–644.
- [5] M.R. Garey, M.R. Johnson, Computers and Intractability, Freeman, New York, 1979.
- [6] S. Ghosh, M.H. Karaata, A self-stabilizing algorithm for coloring planar graphs, Distrib. Comput. 7 (1993) 55–59.
- [7] G.R. Grimmett, C.J.H. McDiarmid, On coloring random graphs, Math. Proc. Cambridge Philos. Soc. 77 (1975) 313–324.
- [8] P.M. Grundy, Mathematics and games, Eureka 2 (1939) 6–8.
- [9] M. Gradinariu, S. Tixeuil, Self-stabilizing vertex coloration and arbitrary graphs, in: 4th International Conference on Principles of Distributed Systems, OPODIS'2000, 2000, pp. 55–70.
- [10] T.R. Jensen, B. Toft, Graph Coloring Problems, John Wiley & Sons, New York, 1995.
- [11] R. Karp, Reducibility among combinatorial problems, in: R.E. Miller, J.W. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85–104.
- [12] M. Nesterenko, A. Arora, Stabilization-preserving atomicity refinement, in: DISC'99, Distributed Computing 13th International Symposium, in: Lecture Notes in Comput. Sci., Vol. 1693, Springer, Berlin, 1999, pp. 254–268.
- [13] F. Petit, V. Villain, A space-efficient and self-stabilizing depth-first token circulation protocol for asynchronous message-passing systems, in: Euro-Par'97—Parallel Processing, Proceedings, in: Lecture Notes in Comput. Sci., Vol. 1300, Springer, Berlin, 1997, pp. 476–479.
- [14] S.K. Shukla, D.J. Rosenkrantz, S.S. Ravi, Observations on self-stabilizing graph algorithms for anonymous networks, in: Proceedings of the 2nd Workshop on Self-Stabilizing Systems, 1995, pp. 7.1–7.15.
- [15] S. Sur, P.K. Srimani, A self-stabilizing algorithm for coloring bipartite graphs, Inform. Sci. 69 (1993) 219–227.