

Fault Tolerant Algorithms for Orderings and Colorings *

Wayne Goddard, Stephen T. Hedetniemi, David P. Jacobs, Pradip K. Srimani
Clemson University
Department of Computer Science
Clemson, SC 29634-0974
{goddard,hedet,dpj,srimani}@cs.clemson.edu

Abstract

A k -forward numbering of a graph is a labeling of the nodes with integers such that each node has less than k neighbors whose labels are equal or larger. We obtain three self-stabilizing (s - s) algorithms for finding a k -forward numbering, provided one exists. One such algorithm also finds the k -height numbering of graph, generalizing s - s algorithms by Bruell et al. [2] and Antonoiu et al. [1] for finding the center of a tree. Another k -forward numbering algorithm runs in polynomial time. There is a strong connection between k -forward numberings and colorings of graphs. We use a k -forward numbering algorithm to obtain an s - s algorithm that is more general than previous coloring algorithms in the literature, and which k -colors any graph having a k -forward numbering. Special cases of the algorithm 6-color planar graphs, thus generalizing an s - s algorithm by Ghosh and Karaata [9], as well as 2-color trees and 3-color series-parallel graphs. We discuss how our s - s algorithms can be extended to the synchronous model.

1. Introduction

One of the most important requirements of modern distributed systems is robustness. A goal of a distributed system is that the system should function correctly in spite of intermittent faults. Ideally the global state of the system should remain in the legitimate state. Often, however, malfunctions or perturbations bring the system to some illegitimate state, and it is desirable that the system be automatically brought back to the legitimate state without the interference of an external agent.

The traditional approach to fault tolerance uses fault masking and is *pessimistic* in the sense that it assumes the

worst case scenario and protects the system against such an eventuality; validity is guaranteed in the presence of faulty processes, which necessitates restrictions on the number of faults and on the fault model. Fault masking is not free; it requires additional hardware or software, and it considerably increases the cost of the system. This additional cost may not be an economic option, especially when most faults are transient in nature and a temporary unavailability of a system service is acceptable for a short period of time.

Systems that reach the legitimate state starting from any illegitimate state in a finite number of steps are called *self-stabilizing systems* [4, 5]. This kind of property is highly desirable for any distributed system, since without having a global memory, global synchronization is achieved in finite time and thus the system can correct itself automatically from spurious perturbation or failures.

Self-stabilization [7] is an *optimistic* way of looking at system fault tolerance because it provides a built-in safeguard against transient failures that might corrupt the data in a distributed system. The objective of self-stabilization (as opposed to masking faults) is to recover from failure in a reasonable amount of time and without intervention by any external agency. Since the faults are transient (eventual repair is assumed), it is no longer necessary to assume a bound on the number of failures. Self-stabilization has potential uses in ad-hoc networks [8], sensor networks [13], and peer-to-peer communication [3].

We assume the state-reading model in which each node v can read the variables of its neighbors $N(v)$, and can make decisions based on the contents of its neighbors' local variables. The contents of a node's local variables determine its *local state*. A node may change its local state by making a *move*, that is, changing the values of its local variables. Self-stabilizing algorithms are given as a set of rules of the form **if** $p(v)$ **then** M , where $p(v)$ is a predicate and M is a move. A node v becomes *privileged* if $p(v)$ is true. When a node becomes privileged, it may execute the corresponding move (i.e., change its local variables). When no further nodes are privileged, we say that the system is *stable*.

*The authors thank the National Science Foundation for NSF Grant ANI-0218495 which helped support this research.

Unless otherwise specified, all algorithms in this paper are *self-stabilizing* or *s-s algorithms*. For most of this paper we assume that there exists a (not necessarily fair) *central daemon* which selects one of the privileged nodes to move, such that no two nodes move at the same time. Also, we present our algorithms for an *anonymous* network where nodes do not have ID's. In Section 5, however, we consider how our s-s algorithms are effected under other distributed models, namely the so-called *synchronous* distributed model and the *distributed daemon* model.

2. Overview of results

An important concept in this paper is a *k-forward numbering*. We say that a function on a node set V

$$c : V \rightarrow \mathbb{N}$$

which assigns to the nodes, not necessarily distinct, non-negative integers, is a *k-forward numbering* if each node has less than k neighbors which are assigned equal or larger numbers. That is, for any node v ,

$$|\{u \in N(v) : c(u) \geq c(v)\}| < k.$$

It will be convenient to define a *k-forward ordering* as a total ordering v_1, \dots, v_n that is a *k-forward numbering*. That is, $|\{v_j \in N(v_i) : j > i\}| < k$, for any i . It is easy to see that any tree has a 2-forward numbering. Indeed, a graph has a 2-forward numbering if and only if it is a forest. Throughout this paper, $k \geq 2$ is an integer.

The primary focus of this paper is s-s algorithms for finding *k-forward numberings*. Three such algorithms are given. On the surface, the algorithms appear simple in that each uses only a single nonnegative integer variable and has a single rule for comparing its value with the value of its neighbors. However, their analyses and correctness proofs are not simple. But what is most surprising is that the subtle differences in the three algorithms produce vastly different behaviors and running times.

In Section 3.1 we give the simplest of our algorithms, which tries to resolve local conflicts by always increasing a local variable to be larger than its neighbors. The algorithm finds a *k-forward numbering* if and only if one exists, but we show, by example, that the number of moves can be exponential. In Section 3.2 we give an s-s algorithm for finding a special kind of *k-forward numbering*, called a *k-height numbering*. This algorithm generalizes the algorithms of Bruell et al. [2] and Antonoiu et al. [1] for finding the center of a tree. In Section 3.3 we give a *k-forward numbering* algorithm that stabilizes in time that is polynomial in n , the network size, and W , the largest number at initialization. All three of these numbering algorithms find 2-forward numberings on trees, 3-forward numberings

on series-parallel graphs, 6-forward numberings on planar graphs, and *k-forward numberings* on partial $(k - 1)$ -trees.

A second focus of this paper is on s-s coloring algorithms. A *coloring* of a graph G is an assignment of integers (or colors) to the nodes such that no two adjacent nodes receive the same color. The *chromatic number* of a graph G is the minimum number of colors used in a coloring, and is denoted $\chi(G)$. A well-known theorem of Brooks [12] says that any for any graph G , $\chi(G) \leq \Delta(G) + 1$, where $\Delta(G)$ denotes the maximum degree of a node in G . S-s algorithms which find $(\Delta + 1)$ -colorings, under various models, are presented by Dolev and Herman [6], Gradinariu and Tixeuil [10], and Hedetniemi et al. [11]. Ghosh and Karaata in [9] give an s-s algorithm to find a 6-coloring of a planar graph, using the fact that every planar graph has a node of degree less than 6.

The *coloring number* of a graph G , denoted $col(G)$, is the smallest number k such that G has a *k-forward numbering*. Since any *k-forward numbering* can be used to greedily *k-color* a graph [12], it follows that $\chi(G) \leq col(G) \leq \Delta(G) + 1$, which is a strengthening of Brooks' Theorem. In a *list coloring*, each node is assigned a list of available colors, and the algorithm must produce a coloring, using a color from each node's list. In Section 4.1 we give a linear-time s-s list-coloring algorithm that assumes a *k-forward numbering* is given.

In Section 4.2 we give an s-s coloring algorithm that will *k-color* any graph having a *k-forward numbering*. Special cases of this algorithm 6-color any planar graph, thus generalizing an s-s algorithm of Ghosh and Karaata [9], 2-color any tree and 3-color any series-parallel graph.

In Section 5 we show that our algorithms self-stabilize even with the synchronous (maximal parallelism) model, thus generalizing results of Shukla et al. [14].

3. k-forward number

In this section, we examine three algorithms for finding *k-forward numberings*. The first algorithm, which we call **Leap**, is the natural generalization of an algorithm by Ghosh and Karaata [9]. However, we show that, while it is efficient for $k = 2$, their algorithm can take exponentially many moves. The second algorithm, which we call **Slide**, is a generalization of center-finding algorithms by Bruell et al. [2] and Antonoiu et al. [1]. This algorithm has the advantage of producing a special numbering, called a *k-height numbering*, which identifies a generalization of centers in other graphs. However the complexity of Algorithm **Slide** remains undetermined. The third algorithm, which we call **UpSlide**, is new. We show that it has a guaranteed quadratic performance. The following is trivial.

Lemma 1 *A graph G has a k-forward ordering if and only if it has a k-forward numbering.*

variable: nonnegative integer $w(v)$

leap: if $|\{u \in N(v) : w(u) \geq w(v)\}| \geq k$
set $w(v) = 1 + \max\{w(u) : u \in N(v)\}$

Figure 1. Algorithm Leap

PROOF. By definition, k -forward orderings are k -forward numberings. Conversely, if $w : V \rightarrow \mathbb{N}$ is a k -forward numbering, then we may order the vertices so that

$$w(v_1) \leq w(v_2) \leq \dots \leq w(v_n).$$

It is easy to see that the total order v_1, \dots, v_n is a k -forward ordering. QED

3.1. k -forward numbering: Leap

In this section we present Algorithm Leap, shown in Figure 1, an s-s algorithm for finding a k -forward numbering. Each node v maintains a single integer variable $w(v)$. The algorithm has a single rule: if a node v has at least k neighbors having the same number or larger number, it sets $w(v)$ to be one larger than the maximum number of a neighbor. This is the natural generalization an s-s algorithm of Ghosh and Karaata [9], while avoiding the use of IDs. The fact that IDs are unnecessary was observed by Shukla et al. in [14].

Let $G = (V, E)$ be a graph, and let (v_1, \dots, v_n) be any fixed, total ordering of V . Assuming that edges $e = v_i v_j$, are always written with $i < j$, this induces a total ordering $(E, <)$ on E where, if $e = v_i v_j$ and $e' = v_{i'} v_{j'}$, then $e < e'$ if and only if either $i < i'$, or $i = i'$ and $j < j'$. Finally, we obtain a total order on the powerset 2^E in the usual lexicographic way: given $S, S' \subseteq E$, $S \neq S'$, if e is the smallest edge in $(S - S') \cup (S' - S)$, then $S < S'$ if and only if $e \in S'$. Note that E is the maximum element under this total order.

Now assume that (v_1, \dots, v_n) is a fixed k -forward ordering of G . Let $w : V \rightarrow \mathbb{N}$ be an arbitrary numbering of V . An edge $e = v_i v_j$, where $i < j$, is called a *bad* edge, relative to w , if $w(v_i) \geq w(v_j)$, and a *good* edge otherwise.

Lemma 2 Assume that G has a k -forward ordering (v_1, \dots, v_n) . In Algorithm Leap, let B_t be the set of bad edges relative to this ordering at time t . Then $B_{t+1} < B_t$, in the total order on 2^E .

PROOF. Assume that node $v = v_i$ moves at time t . Before the move, let

$$S = \{u \in N(v) \mid w(v) \leq w(u)\}.$$

Since v was privileged, $|S| \geq k$. Let $S = \{v_{j_1}, \dots, v_{j_s}\}$, and without loss of generality, we may assume

$$j_1 < j_2 < \dots < j_s.$$

Note that $j_1 < i$, for otherwise we would not have a k -forward ordering. Therefore, the move causes the edge $v_{j_1} v_i$ to change from bad to good. Let $A = \{vu \mid u \in S\}$, and $A' = \{vu \mid u \in N(v) - S\}$. We have observed that the minimum member of A changes from bad to good. The status of each edge in A' , however, remains unchanged, as does the status of all edges not incident with v . Therefore $B_{t+1} < B_t$. QED

Theorem 1 Algorithm Leap stabilizes if and only if the graph has a k -forward numbering. If it stabilizes, it produces a k -forward numbering.

PROOF. If Algorithm Leap stabilizes, then no node is privileged, so the numbers $w(v)$ form a k -forward numbering. On the other hand, assume there is a k -forward numbering. By Lemma 1, there is a k -forward ordering (v_1, \dots, v_n) . By Lemma 2 the algorithm can execute only finitely many moves. QED

Lemma 2 implies that if a node v is privileged then there exists at least one bad edge incident to v . The converse is not true; Algorithm Leap can halt with bad edges. For example, let $G = P_3$ and $k = 2$. Ordering the nodes from left to right by v_1, v_2, v_3 is a 2-forward ordering. But if $w(v_1) = 3, w(v_2) = 2, w(v_3) = 1$, then the algorithm halts, yet both edges are bad.

Quadratic for $k = 2$ We now observe that on trees, Algorithm Leap takes only $O(n^2)$ moves, when $k = 2$. Let T be a tree, and let v_1, v_2, \dots, v_n be a 2-forward ordering of its nodes, where v_n is the root, and each node has a greater index than all of its children. As before, define an edge $v_i v_j$, $i < j$, to be bad if $w(i) \geq w(j)$, and good otherwise. Let B be the set of bad edges. If v_i moves, then there is at least one bad edge, incident to one of its children, that becomes good. If v_i is not the root, after the move, there is exactly one bad edge incident to v_i , namely the edge to its parent. If v_i is the root, then after the move there are no bad edges incident to v_i . In either case, the sum

$$\Phi = \sum_{\substack{v_i v_j \in B \\ i < j}} (n - j)$$

must decrease.

Exponential for $k \geq 3$ We now show that Algorithm Leap can take exponential time. Set $k = 3$. We let H_n be the graph shown in Figure 2, having n pairs of

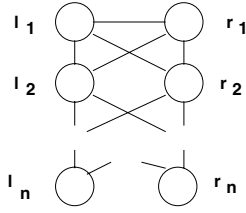


Figure 2. the graph H_n

nodes, arranged in rows. Nodes in row i , $1 \leq i \leq n-1$, are adjacent to both nodes in row $i+1$. Moreover, in row one, the nodes l_1 and r_1 are adjacent.

We assign numbers to the nodes, which represent the values $w(v)$ from Algorithm Leap. A numbering of H_n is said to be in *final form* if both numbers in row i are greater than both numbers in row $i+1$. Such a numbering is a 3-forward numbering. Next, we say that a numbering of H_n is in *initial form* if

1. Each number in row two is greater than both numbers in row one.
2. In rows $i \geq 3$, some number is greater than both numbers in row $i-1$.

Figure 3 shows H_3 in both initial and final form.

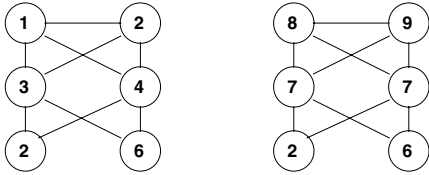


Figure 3. An initial form and final form of H_3

We now define the function

$$t_n = \begin{cases} 2 & \text{if } n = 2 \\ 2t_{n-1} + 2(n-2) & \text{if } n \geq 3 \end{cases}$$

Theorem 2 *If H_n is in initial form, there is a sequence of t_n moves that terminate in final form.*

PROOF. The proof is by induction on n . We omit the details. QED

For each $k \geq 4$, it is also possible to construct a similar family of graphs which allow exponential running times. This shows that the coloring algorithm in [9] runs in exponential time.

3.2. k -height numbering: Slide

In this section we present a variation of Algorithm Leap. It finds a special kind of k -forward numbering by taking a

more conservative approach to adjusting values. This time, if a node has too many larger neighbors, it adjusts its value to be just large enough so that the desired k -forward property holds locally. If the algorithm stabilizes, it results in a *unique* numbering, regardless of the initial conditions.

The resulting algorithm is the natural generalization of existing s-s algorithms for finding centers. Recall that the *eccentricity* of a node is the maximum length of a shortest path to some other node, and the *center* of a graph is the set of nodes having smallest eccentricity. There are either one or two central nodes in a tree. The center of a tree can be found by first numbering the leaves with zero, and then at stage $i \geq 1$, numbering with i the leaves in the subgraph of unnumbered nodes. The center of a tree consists of the node or two nodes with greatest number, when all nodes have been numbered.

This numbering can be generalized in the following way: first, number with zero, all nodes having degree less than k . Then at stage $i \geq 1$, number with i those nodes having degree less than k in the subgraph of unnumbered nodes. Such a numbering will be called the *k -height numbering* of G . The *k -height of a node v* is the number thus assigned, denoted $h_k(v)$, and the *k -height of G* , denoted $h_k(G)$, is the largest such number. When k is clear from the context, we sometimes will refer to simply the *height* of v .

Note that when the graph is a tree, and $k = 2$, this amounts to the center finding algorithm described above. Hence, all trees can be 2-height numbered. For a given k , an arbitrary graph G might not have a k -height numbering, but if k is sufficiently large, G will have a k -height numbering.

Lemma 3 *A graph has a k -height numbering if and only if it has a k -forward numbering.*

PROOF. By the construction, any k -height numbering is a k -forward numbering. Conversely, assume that G has a k -forward numbering $w : V \rightarrow \mathbb{N}$. Since w is a k -forward numbering, the nodes with smallest w value must have degree less than k . Letting S_0 denote the set of nodes having degree less than k , we number all nodes in S_0 with 0. If $V - S_0 \neq \emptyset$, let S_1 be the subset of nodes of $V - S_0$ that are adjacent to less than k members of $V - S_0$. Such a node must exist; in particular, the unnumbered node having smallest w value satisfies this property. We assign the value 1 to members of S_1 , and continue. Eventually we achieve a k -height numbering. QED

Algorithm Slide, shown in Figure 4, is an algorithm for finding a k -height numbering of a graph, if one exists. This algorithm extends the center-finding algorithms for trees [1, 2], to graphs having coloring number k .

For a multiset of integers $S = \{s_i\}_{i=1}^r$ where

$$s_1 \geq s_2 \geq \dots \geq s_r$$

variable: nonnegative integer $w(v)$

slide: if $w(v) \neq x(v)$
 set $w(v) = x(v)$

Figure 4. Algorithm Slide

define $r_k(S) = s_k$, the k th largest value in S . Each node v in Algorithm Slide maintains a nonnegative variable $w(v)$. For a node v , we define the *consistent number* $x(v)$ by

$$x(v) = \begin{cases} 0 & \text{if } v \text{ has less than } k \text{ neighbors} \\ 1 + r_k(\{w(u) : u \in N(v)\}) & \text{otherwise.} \end{cases}$$

Note that the consistent number of a node v is precisely the smallest nonnegative number for which v will have fewer than k neighbors whose values are greater or equal.

Lemma 4 Let G be a graph whose k -height is h . During Algorithm Slide the values $w(v)$ can never exceed $W + h$, where W is the largest initial value of $w(v)$.

PROOF. By induction on h , we show that, for nodes v having k -height h , the values $w(v)$ can never exceed $W + h$. First, let v be a node of height $h = 0$. If node v never moves, then obviously the condition $w(v) \leq W$ must always exist. But if node v makes a first move, then it becomes $x(v) = 0$ permanently, since it has fewer than k neighbors. By induction, now assume that for all nodes u having height $h' < h$, the condition $w(u) \leq W + h'$ is maintained during any execution. Now let v be a node that moves, having height h . Consider its neighborhood $N(v)$: less than k of the members of $N(v)$ have height h or greater. So among the k neighbors having largest value, at least one, say u , has height $h' < h$. So we have

$$x(v) \leq w(u) + 1 \leq W + h' + 1 \leq W + h,$$

the middle inequality following from our induction assumption. QED

In Algorithm Slide a move by a node v always changes the value of $w(v)$, so we may speak of a move as either being *increasing* or *decreasing*. For a node v and a natural number r , let $d_v(r)$ denote a decreasing move by a node v to r . Let $D_v(r)$ be the number of decreasing moves made by v to r . We omit the proof of the following lemma, which is trivial.

Lemma 5 Let S be a multiset of integers and let $S' = S - \{s_i\} \cup \{s'_i\}$ for some $s_i \in S$. If $r_k(S') < r_k(S)$ then $s'_i \leq r_k(S')$.

Lemma 6 Let M_{t_1} and M_{t_2} be moves made by the same node v at times $t_1 < t_2$, and where $M_{t_2} = d_v(r)$. Then at some time in the interval (t_1, t_2) there exists a decreasing move $d_u(s)$ by a node u adjacent to v with $s < r$.

PROOF. We may assume v makes no moves in the interval (t_1, t_2) . Let S_t denote the multiset of numbers $\{w(u) : u \in N(v)\}$ at time t . Because node v moves at time t_1 , at time $t_1 + 1$, $w(v) = x(v) = r_k(S_{t_1+1}) + 1$. At time t_2 , however, v makes a decreasing move to $w(v) = x(v) = r_k(S_{t_2}) + 1 = r$. This means that in the time interval (t_1, t_2) , r_k decreased to $r - 1$. By Lemma 5 it follows that there was a move $d_u(s)$ for $s \leq r - 1$. QED

Lemma 7 For any natural number r , the sum $\sum_{v \in V} D_v(r)$ is finite.

PROOF. We induct on r . When $r = 0$ we see that each node v can make at most one move $d_v(0)$. Assuming that $D_u(s)$ is finite for all u and $s < r$, suppose by contradiction that $D_v(r)$ is not finite. By Lemma 6, between any two consecutive moves $d_v(r)$, is a decreasing move $d_u(s)$, where $s < r$ by a neighbor u . Thus for some particular u and $s < r$ there must be infinitely many moves $d_u(s)$, contradicting the induction hypothesis. QED

Bruell et al. [2] show that for $k = 2$ there can be at most $O(n^2)$ decreasing moves. However, that result exploits the fact that in trees there is a unique path between any two nodes. There does not appear to be an equivalent result for $k \geq 3$.

Theorem 3 Algorithm Slide stabilizes on a graph G if and only if G has a k -forward numbering.

PROOF. It is easy to see that if Algorithm Slide stabilizes, then each node v has fewer than k neighbors of greater or equal value, and so the numbers $w(v)$ form a k -forward numbering. Conversely, assume G has a k -forward numbering. By Lemma 3, it has a k -height numbering. Then by Lemma 4, the values $w(v)$ are bounded by $B = W + n$. Therefore, the total number of decreasing moves is exactly

$$D = \sum_{0 \leq r < B} \sum_v D_v(r),$$

which by Lemma 7, is finite. The number of increasing moves must therefore be finite since a node can make no more than B increasing moves before making a decreasing move. The number of moves is therefore finite, completing the proof. QED

Theorem 4 At stabilization, the values $w(v)$ of Algorithm Slide are a k -height numbering.

PROOF. Omitted.

<p>variable: nonnegative integer $w(v)$</p> <p>upslide: if v has at least k larger or equal neighbors set $w(v) = 1 + r_k(\{w(u) : u \in N(v)\})$</p>

Figure 5. Algorithm UpSlide

3.3. Polynomial time k -forward numbering: UpSlide

In this section we use the best features of Leap and Slide to give Algorithm UpSlide, a third s-s algorithm for finding a k -forward numbering. Algorithm UpSlide, shown in Figure 5, is similar to Algorithm Slide except that nodes only make *increasing* moves. The advantage is that it runs in time polynomial n , the size of the graph, and W , the largest initial value.

Lemma 8 *Let G be a graph whose k -height is h . During Algorithm UpSlide the values $w(v)$ can never exceed $W + h$, where W is the largest initial value of $w(v)$.*

PROOF. The proof is identical to that of Lemma 4. QED

Theorem 5 *If G has a k -forward numbering, then Algorithm UpSlide stabilizes in $n(W + n)$ moves.*

PROOF. By Lemma 3, G has a k -height numbering. Since the height of nodes are bounded by n , by Lemma 8, each node can move at most $W + n$ times. Thus the sum of all moves is at most $n(W + n)$. QED

Theorem 6 *Algorithm UpSlide stabilizes if and only if the graph has a k -forward numbering. If it stabilizes, it produces a k -forward numbering.*

PROOF. Clearly if Algorithm UpSlide has stabilized, then G has a k -forward numbering. The other direction follows from Theorem 5. QED

We observe that the running time of Algorithm UpSlide indeed depends on both n and W , even when $k = 2$. Consider the path P_4 on four nodes in which end nodes have been given a large integer W and internal nodes have been given zero and one. The internal nodes will alternately move, each time incrementing themselves by only two, and thus taking about W moves. Furthermore, we can show examples where Algorithm UpSlide takes $\Omega(n(W + n))$, so the bound in Theorem 5 is the best possible.

4. Application: coloring

In this section we present a self-stabilizing (list) coloring algorithm that assumes a k -forward numbering is already given. In the Section 4.2 we will drop this assumption.

4.1. List coloring and the coloring number

Recall that the coloring number $col(G)$ is the smallest number k such that G has a k -forward numbering. Its significance is that

$$\chi(G) \leq col(G) \leq \Delta(G) + 1, \quad (1)$$

which is a strengthening of Brooks' Theorem. The second inequality in (1) is obvious since any total ordering is a $(\Delta(G) + 1)$ -forward ordering. The first inequality is easy to prove. The book by Jensen and Toft [12] contains more discussion of the coloring number¹.

Suppose that associated with each node v is a list $L(v)$ of distinct colors. The minimum k such that, for any assignment of lists of k distinct colors, there always exists a coloring c on V , where $c(v) \in L(v)$ for all $v \in V$, is called the *choice number* of the graph [12], and is denoted $choice(G)$. Clearly, $\chi(G) \leq choice(G)$. It is easy to show that $choice(G) \leq col(G)$.

Algorithm ColorByNumber, shown in Figure 6, assumes that each node v has a list $L(v)$ of k colors, and a k -forward numbering w is known. We regard these as constants associated with each node. Note, however, that we obtain an ordinary k -coloring algorithm by defining, for each v , $L(v) = \{1, \dots, k\}$.

For a node v and k -forward numbering w , define the set $F_w(v)$ as those colors used by neighbors of v having numbers the same or larger than v 's:

$$F_w(v) = \{c(u) : u \in N(v) \wedge w(v) \leq w(u)\}.$$

The algorithm produces a k -coloring by giving the responsibility of rectifying monochromatic edges to the smaller-numbered node.

Theorem 7 *Given any graph G and k -forward numbering, and any assignment of lists $L(v)$ of size k to the vertices, Algorithm ColorByNumber stabilizes, producing a list coloring.*

PROOF. First, note that the algorithm is well-defined in the sense that the set $L(v) - F_w(v)$ is always nonempty: since w is a k -forward numbering, there is always a color that may be selected. Next, it is easy to see that the system is stable if and only if the values $c(v)$ form a coloring. Finally, we

¹There $col(G)$ is defined in an equivalent way, but in terms of k -back orderings.

<p>variable: color $c(v)$ constant: k-forward number $w(v)$ constant: list of k distinct colors $L(v)$</p> <p>recolor: if $c(v) \in F_w(v)$ set $c(v)$ to any color in $L(v) - F_w(v)$</p>

Figure 6. Algorithm ColorByNumber

claim that the system must stabilize. We will show that for any v , $m(v)$, the number of moves made by v , is finite. For each node v we define

$$\begin{aligned} N_=(v) &= \{u \in N(v) \mid w(u) = w(v)\} \\ N_>(v) &= \{u \in N(v) \mid w(u) > w(v)\} \\ N_{\geq}(v) &= N_=(v) \cup N_>(v) \end{aligned}$$

Let us say that a move, made by a neighbor u of v , is *injurious* if $c(u)$ becomes $c(v)$. Besides its initial move, a node v can make at most one move for every injurious move made by a member of $N_{\geq}(v)$. But moves made by members of $N_=(v)$ are never injurious. Hence

$$m(v) \leq 1 + \sum_{u \in N_{>(v)}} m(u). \quad (2)$$

We now order the nodes v_1, v_2, \dots, v_n such that

$$w(v_1) \geq w(v_2) \geq \dots \geq w(v_n).$$

By induction on i we claim that each $m(v_i)$ is finite. Clearly $m(v_1) \leq 1$. Inequality (2) completes the induction. QED

In [10] and [11], s-s algorithms for $(\Delta + 1)$ -colorings are given that run in n moves. Note that *any* numbering, say assigning all numbers $w(v)$ to be equal, is a $(\Delta + 1)$ -forward numbering. In this case, the set $F_w(v)$ is the full set of colors in $N(v)$. It is clear that in this case our algorithm runs in at most n moves. Therefore Algorithm ColorByNumber obtains a $(\Delta + 1)$ -list-coloring, providing a generalization of the s-s algorithms in [10] and [11].

Algorithm ColorByNumber extends some of the results of Shukla et al. [14]. In particular, since a ring has a 3-forward numbering, Algorithm ColorByNumber can 3-list color rings without requiring an orientation, and since trees have 2-forward numberings, Algorithm ColorByNumber can 2-list-color trees.

4.2. Coloring algorithm

By composing Algorithm ColorByNumber with any of the k -forward numbering algorithms we can obtain a general s-s algorithm for list k -coloring a graph G , which

<p>variable: nonnegative integer $w(v)$ variable: color $c(v)$ constant: list of colors $L(v)$</p> <p>upslide: if v has at least k larger or equal neighbors set $w(v) = 1 + r_k(\{w(u) : u \in N(v)\})$ recolor: if $c(v) \in F_w(v)$ and $N_{\geq}(v) < k$ set $c(v)$ to any color in $L(v) - F_w(v)$</p>
--

Figure 7. Algorithm k-Coloring

does not require an a-priori k -forward numbering. This algorithm is shown in Figure 7. As before, one obtains an ordinary k -coloring algorithm by taking the lists to be $\{1, \dots, k\}$. The algorithm stabilizes if and only if G has coloring number less than or equal to k . Our approach is modeled after Ghosh and Karaata [9], who obtain a self-stabilizing algorithm for 6-coloring planar graphs. The idea is to combine the two rules **upslide** and **recolor** from Algorithms UpSlide and ColorByNumber, respectively. Rule **upslide** is exactly the k -forward numbering rule from Algorithm UpSlide. Rule **recolor** is the list coloring rule of Algorithm ColorByNumber with an additional boolean constraint; a node must have less than k larger neighbors before it can change its color. This ensures that a new color is available.

Lemma 9 *If G has a k -forward ordering, Algorithm k-Coloring makes at most $nW + n^2$ upslide moves.*

PROOF. The **upslide** moves use only variable w and this variable is not modified by the **recolor** moves. Therefore this is obvious in light of Theorem 5. QED

Lemma 10 *Algorithm k-Coloring can make only a finite number of consecutive recolor moves.*

PROOF. Assume that there is an arbitrary numbering w , and we will consider a sequence of consecutive recoloring moves. Some nodes v can not move at all because $|N_{\geq}(v)| \geq k$. But in any case, after its first move, a node v can move at most once for every injurious move made by a member of $N_{>(v)}$. Thus the same induction argument used in Theorem 7 may be applied to show that $m(v)$ is finite. QED

Theorem 8 *If G has a k -forward ordering, Algorithm k-Coloring stabilizes, and produces a k -coloring.*

PROOF. It follows from Lemma 9 and Lemma 10 that Algorithm k-Coloring must stabilize. At this time, the values $w(v)$ must form a k -forward numbering, or else some node

could execute the **upslide** rule. The values $c(v)$ must form a coloring, for otherwise, if there were a monochromatic edge vu with $w(v) \leq w(u)$, since the k -forward numbering is in place, node v could execute the **recolor** move. QED

Any forest has a 2-forward numbering. It is also known [12] that series-parallel graphs have 3-forward numberings, and planar graphs 6-forward numberings. Therefore we have:

Corollary 1 Any forest is 2-colored by Algorithm k-Coloring.

Corollary 2 Any series-parallel graph is 3-colored by Algorithm k-Coloring.

Corollary 3 Any planar graph is 6-colored by Algorithm k-Coloring.

Corollary 4 For any surface S there is a constant c_S such that all graphs embeddable on that surface are c_S -colored by Algorithm k-Coloring.

PROOF. It is known [12] that the coloring number for all such graphs is bounded. QED

Let G be a graph having maximum degree Δ . Brooks' Theorem guarantees the existence of a $(\Delta+1)$ -coloring, and if G is not an odd cycle or a complete graph, there exists a Δ -coloring. It is easy to show that graphs having maximum degree $k = \Delta$, that are not regular, must have a k -height numbering. Thus we have

Corollary 5 Any non-regular graph having maximum degree Δ is Δ -colored by Algorithm k-Coloring.

5. Other models: distributed daemon and synchronous

In the distributed daemon model, the daemon may select an arbitrary subset of privileged nodes. The serial-model algorithms of this paper can be converted to s-s algorithms under the distributed daemon model at the cost of IDs and time.

In the synchronous model, all privileged nodes move simultaneously. All three of our k -forward numbering algorithms run correctly under this model. We know that at least one of these, Algorithm Slide, converges very fast.

Theorem 9 Under the synchronous model, Algorithm Slide stabilizes if and only if its graph G has a k -forward numbering. In this case, it stabilizes in at most n steps.

References

- [1] G. Antonoiu and P.K. Srimani. A self-stabilizing distributed algorithm to find the center of a tree graph. *Parallel Algorithms and Applications*, 10:237–248, 1997.
- [2] S.C. Bruell, S. Ghosh, M.H. Karaata, and S.V. Pemmaraju. Self-stabilizing algorithms for finding centers and medians of trees. *SIAM Journal on Computing*, 29(2):600–614, 1999.
- [3] M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *3rd IEEE International Conference on Peer-to-Peer Computing*, (P2P '03). Linköping, Sweden, September 2003.
- [4] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.
- [5] E. W. Dijkstra. A belated proof of self-stabilization. *Distributed Computing*, 1(1):5–6, 1986.
- [6] S. Dolev and T. Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago Journal of Theoretical Computer Science*, 3(4), 1997.
- [7] S. Dolev. *Self-stabilization*, MIT Press, 2000.
- [8] S. Dolev, E. Schiller, and J. Welch. Random walk for self-stabilizing group communication in ad-hoc networks. In *Proc. 21st IEEE Symposium on Reliable Distributed Systems*, pages 70–79, 2002.
- [9] S. Ghosh and M.H. Karaata. A self-stabilizing algorithm for coloring planar graphs. *Distributed Computing*, 7:55–59, 1993.
- [10] M. Gradinariu and S. Tixeuil. Self-stabilizing vertex coloration of arbitrary graphs. In *4th International Conference On Principles Of Distributed Systems, OPODIS'2000*, pages 55–70. Studia Informatica Universalis, 2000.
- [11] S. T. Hedetniemi, D. P. Jacobs, and P.K. Srimani. Linear-time self-stabilizing colorings. *Information Processing Letters*, 87:251–255, 2003.
- [12] T.R. Jensen and B. Toft. *Graph Coloring Problems*. Wiley, 1994.
- [13] S. S. Kulkarni and U. Arumugam. Collision-free communication in sensor networks. In *Proceedings Self-Stabilizing Systems: 6th International Symposium, SSS 2003*, San Francisco, pages 17 – 31, Springer-Verlag, Heidelberg, 2003
- [14] S. Shukla, D. Rosenkrantz, and S. Ravi. Developing self-stabilizing coloring algorithms via systematic randomization. In *Proceedings of the International Workshop on Parallel Processing*, pages 668–673, Bangalore, India, 1994. Tata-McGraw Hill, New Delhi.