A SELF-STABILIZING ALGORITHM FOR EDGE-COLORING OF GRAPHS

Łukasz KUSZNER, Adam NADOLSKI [*]

Abstract. We discuss the edge-coloring problem in a distributed model. We give a self-stabilizing algorithm for coloring the edges of a graph G with $2\Delta - 1$ colors, which runs in $O(\Delta m)$ moves, where m is the number of edges and Δ denotes the maximum vertex degree of G. Our algorithm is based on Hsu-Huang's algorithm for maximal matching.

Keywords: distributed algorithm, edge-coloring, self-stabilization.

1 Introduction

Edge-coloring of graphs is a classical problem in graph theory [15]. The primary objective of edge-coloring is to assign colors to the edges of a graph in such a way that no two neighboring edges obtain the same color. It is known that $\Delta + 1$ colors always suffice to color the edges of a graph [21] but finding an optimal coloring is NP-hard even in a centralized model [13]. In a distributed setting the situation is more difficult, though some approximation algorithms are known. The article [18] describes a fast algorithm for $(2\Delta-1)$-edge-coloring of arbitrary graphs, working in $O(\Delta \log^* n)$ rounds, where $\log^*$ denotes the iterated logarithm. In another paper on distributed graph coloring, Grable and Panconesi [10] gave a distributed algorithm that computes an edge-coloring in $O(\log \log n)$ rounds. The presented solution assumes that the edges in a graph act as processors. Such an assumption can not be justified in the standard case, where the edges in a graph model communication channels. The excellent approximation ratio, obtained in the paper, should be rather seen as a result for vertex coloring of an associated line graph. An experimental study of the problem can be found in [16]. In the paper written by Panconesi and Srinivasan [19] the authors describe an algorithm for $(1.6\Delta + O(\log^{1+\gamma} n))$-edge-coloring in $O(\log n)$

[*]Gdańsk University of Technology, Department of Algorithms and System Modeling, Narutowicza 11/12, 80–952 Gdańsk, POLAND, {kuszner,nadolski}@eti.pg.gda.pl.

time. Another result has been achieved in [4], where the authors propose an algorithm which obtains an $O(\Delta \log n)$-edge-coloring in $O(\log^4 n)$ time.

Self-stabilization, an interesting concept introduced in [6], can be seen as an approach for designing resilient distributed systems. A self-stabilizing system must be able to achieve a legitimate global state starting from any possible global state. There are several self-stabilizing algorithms for some graph-theoretical problems: ring orientation [12], center and median of trees [3], spanning tree [1]. Also some graph coloring algorithms have been described. In [20] an exact algorithm for coloring bipartite graphs was shown. Ghost and Karaata [7] gave an algorithm for coloring planar graphs with six colors. Their work has been extended in [9], where a generalization of the problem has been discussed with performance analysis. In another recent paper [11] two fast $(\Delta + 1)$-coloring algorithms for arbitrary graphs were presented.

However, all these results concern vertex-coloring of graphs. A natural question arises about the existence of self-stabilizing algorithm for edge-coloring of graphs. In [17] such an algorithm was introduced. However, in that paper the authors assumed that the sequence of moves is scheduled by a weakly fair daemon. This implies that every vertex appears in the schedule infinitely often. In this paper we provide an edge-coloring algorithm which works in a different model, where the scheduling daemon chooses any vertex from the set of active ones, without any assumptions on scheduling fairness. In our approach we assign colors to the edges in a similar way as in Hsu-Huang's algorithm [14] edges are included into a matching. That is, a color is assigned to an edge when one of the neighboring vertices proposes a color and the second one accepts that proposal.

2 The algorithm

The edge-coloring problem is defined as follows. Let $G = (V, E)$ be a simple graph with a vertex set V and an edge set E. An assignment $c : E \to \mathbb{N}$ is called an *edge-coloring* of G if every two adjacent edges are colored by different colors.

In self-stabilizing algorithms each vertex maintains variables determining its local state. The global state of the system is the union of all local states. Initially, all the local states contain arbitrary values. A vertex can change its local state by making a move. We follow the *central daemon* model [6, 7, 9, 11, 14, 20], where no two nodes move at the same time. At first glance such a model can be seen as very strong, however it is equivalent to one where only local mutual exclusion of neighboring nodes is guaranteed. Moreover, there exist protocols to convert algorithms designed for a central daemon model to weaker ones [2, 5].

The algorithm for each vertex v is defined by a set of rules of the form **if** $p(v)$ **then** A, where $p(v)$ is a predicate over local states of v and its neighbors, and A is an action changing the local state of v. We say that a rule is *active* at vertex v, if a predicate $p(v)$ associated to this rule is satisfied. Moreover, a vertex v is said to be *active* when any rule is active at v, otherwise v is *stable*. In every move the central daemon chooses an arbitrary vertex among active nodes and selects any rule which made it active. Then, an action associated with this rule is performed and the next move follows. Please note that neither fairness guarantees, nor any node and rule privileges are assumed. If all vertices in a graph are stable, we say that the system is *stable*, i.e. no further moves are made and the solution can be determined when the global state is known.

Let $D(u,v) = \deg(u) + \deg(v) - 1$ for every pair of adjacent vertices u and v. A state of any vertex v in our algorithm is an array $S_v[\,]$ consisting of $D(v)$ elements, where $D(v) = \max\{D(u,v) :\ u \in N(v)\}$ and $N(v)$ denotes the set of vertices adjacent to v. Each element of $S_v[\,]$ is a pointer, which points to null or to one of the neighbors of v.

The interpretation of a state is as follows. If $S_v[i] = u$ then vertex v suggests a color i for edge $\{u,v\}$. Observe that the proposed color i satisfies $i \le D(v) \le 2\Delta - 1$, therefore no more than $2\Delta - 1$ colors can be used. The algorithm at vertex v is given by the following six rules.

```
R0:  if $\exists_{i \le D(v)} \exists_{u \in N(v)} S_v[i] = u \wedge i > D(u,v)$
     then for $k = 1$ to $D(v)$ do
             if $S_v[k] = u$ then $S_v[k] = \text{null}$

R1:  if $\exists_{u \in N(v)} \exists_{i < j \le D(u,v)} S_v[i] = u \wedge S_v[j] = u$
     then for $k = 1$ to $D(v)$ do
             if $S_v[k] = u$ then $S_v[k] = \text{null}$

R2:  if $\exists_{u \in N(v)} \exists_{i < j \le D(u,v)} S_v[i] = u \wedge S_u[j] = v$
     then for $k = 1$ to $D(v)$ do
             if $S_v[k] = u$ then $S_v[k] = \text{null}$

R3:  if $\exists_{u \in N(v)} \exists_{i \le D(u,v)} S_v[i] = u \wedge S_u[i] = w \wedge w \ne v$
     then for $k = 1$ to $D(v)$ do
             if $S_v[k] = u$ then $S_v[k] = \text{null}$

R4:  if $\exists_{u \in N(v)} \exists_{i \le D(u,v)} \Big( S_u[i] = v \wedge S_v[i] = \text{null} \wedge \big(\forall_{\substack{j \le D(u,v) \\ j \ne i}} S_u[j] \ne v\big) \wedge$
       $\big(\forall_{j \le D(v)} S_v[j] \ne u\big)\Big)$
     then $S_v[i] = u$

R5:  if $\exists_{u \in N(v)} \Big( \big(\forall_{j \le D(u,v)} S_u[j] \ne v\big) \wedge \big(\forall_{j \le D(v)} S_v[j] \ne u\big) \wedge \big(\exists_{i \le D(u,v)} (S_v[i] =$
       $\text{null} \wedge S_u[i] = \text{null} \wedge \forall_{w \in N(v)} S_w[i] \ne v)\big)\Big)$
     then $S_v[i] = u$
```

To get a proper coloring some conditions must hold. Consider a pair of adjacent vertices u and v. First, colors proposed by u and v to edge $\{u, v\}$ must agree, i.e. $S_v[i] = u$ must imply $S_u[i] = v$. Secondly, the condition $S_v[i] = u$ and $S_u[i] = v$ must be satisfied for exactly one $i \leq D(u, v)$. To achieve this a vertex v must not:

1. propose a color to $\{u, v\}$ which is illegal for u (rule $R0$),
2. propose more than one color to $\{u, v\}$ (rule $R1$),
3. propose a color different from u for the edge connecting them (rule $R2$),
4. propose a color to $\{u, v\}$, which is used by u for another edge incident to u (rule $R3$).

In short, rules $R0$–$R3$ are necessary to remove potential conflicts. The next two rules are used to assign a color to an edge. Rule $R5$ proposes a feasible color if there has not been any color proposed yet, while rule $R4$ accepts the color proposed by a neighbor. In the following sections we show in detail that a proper coloring can be associated with every stable state, which is obtained in at most $O(m\Delta)$ moves, regardless of the initial state.

3 Example

In Fig. 1 an example for cycle C_3 is shown. Each of the pictures illustrates the states of the vertices in the consecutive moves. Stable vertices are marked as empty circles, while the remaining ones are marked as solid. A vertex which is likely to make the next move is marked by an additional circle followed by the name of rule which makes it active. The state is given in brackets close to each vertex.

4 Correctness of the algorithm

According to the informal interpretation given in the previous section, a coloring is given by the following formula:

$$c(\{u, v\}) = i \iff \big(S_v[i] = u \wedge S_u[i] = v\big). \tag{1}$$

First, we prove that in every stable state condition (1) provides a legal coloring of the edges. We divide our reasoning into several lemmas.

Lemma 1. *In every stable state, every vertex has at most one color assigned to each incident edge, i.e. condition $S_v[i] = S_v[j] = u$ must not hold for any pair of adjacent vertices u, v and $i \neq j$.*

Proof. Suppose that there exists a vertex v with more than one color assigned to an incident edge $\{u, v\}$, i.e. there exist $i < j$ such that $S_v[i] = u$ and $S_v[j] = u$. Then either v is active according to rule $R1$ if $j \leq D(u, v)$, or v is active according to $R0$ if $j > D(u, v)$. This contradicts the assumption that the state is stable. □

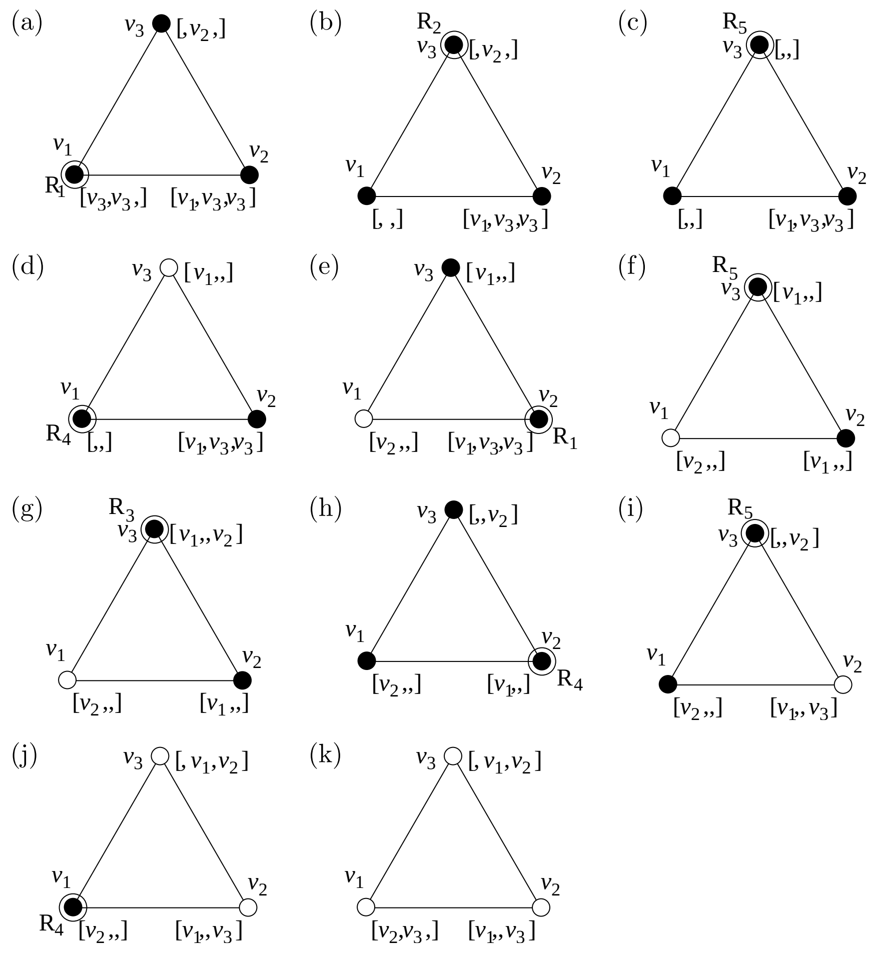

Figure 1: An example for the system graph C_3.

Corollary 1. *In every stable state, every vertex v has at most* $\deg(v)$ *different colors assigned in table $S_v[\,]$. The other values in $S_v[\,]$ are set to* null.

The next lemma follows directly from the definition of rule $R0$.

Lemma 2. *In every stable state, every color proposed by each vertex v to vertex u is also a proper color for u, i.e. if $S_v[i] = u$ then $i \leq D(u, v)$.*

In the following lemma we argue that in a stable state an edge $\{u, v\}$ is proposed the same color by both u and v.

Lemma 3. *Assume that a state of G is stable and $\{u, v\}$ is one of the edges of G. Then there exists i such that $S_v[i] = u$ and $S_u[i] = v$.*

Proof. Consider a stable state and suppose that there exists an edge $\{u, v\}$ not fulfilling the thesis of the lemma. There are three possibilities. In the first case neither u nor v proposes a color for edge $\{u, v\}$. Then, according to Lemma 1, vertex u has at most $\deg(u) - 1$ colors used in $S_u[]$. In the same way v has at most $\deg(v) - 1$ colors used. Therefore, there must exist a color i such that $1 \leq i \leq D(u, v)$ and $S_v[i] = S_u[i] =$ null. This yields that both u and v are active according to rule $R5$, a contradiction.

The second possibility occurs if only one vertex among u, v has a color for $\{u, v\}$ assigned in its table S. Suppose that $S_u[i] = v$ for some $i \leq D(u, v)$ and $S_v[j] \neq u$ for any $j \leq D(u, v)$. Then, either $S_v[i] =$ null or $S_v[i] = w$, where $w \neq u$. In the first case, by Lemma 1, $S_u[j] = v$ for no $j \neq i$, therefore v is active according to $R4$. In the second case vertex u is activated by rule $R3$.

Finally, if vertices u and v have different colors for edge $\{u, v\}$, say $S_u[i] = v$ and $S_v[j] = u$ for $i \neq j$, then both u and v are active according to $R2$. □

The lemmas we have just proved imply the following theorem.

Theorem 1. *Every stable state corresponds to a proper edge-coloring of the graph.*

Proof. According to Lemmas 1 and 3 formula (1) provides a correct definition of an assignment c. Moreover, c is a proper edge-coloring, since the identity $c(\{u, v\}) = c(\{v, w\}) = i$ for a pair of adjacent edges $\{u, v\}$, $\{v, w\}$ and a color i would imply $S_v[i] = u$ and $S_v[i] = w$. □

Now, we show that our algorithm finds a solution in a finite number of moves. The following observation is straightforward.

Lemma 4. *All the rules except $R4$ and $R5$ set values to* null. *Whenever rule $R4$ or $R5$ is applied to a vertex v, it changes $S_v[i]$ from* null *to a pointer to some vertex u.*

From now on we say that a state of edge $\{u, v\}$ is *stable*, whenever $S_v[i] = u$ and $S_u[i] = v$ for some $i \leq D(u, v)$ and $S_v[j] \neq u$, $S_u[j] \neq v$ for any $j \neq i$. Moreover, we say that a move is performed on edge $\{u, v\}$, if it sets $S_v[i]$ to u or $S_u[i]$ to v or if it changes $S_v[i]$ from u to null or $S_u[i]$ from v to null. From now on we bound the number of rules that are performed on a fixed edge $\{u, v\}$. It is easy to observe that whenever a state of edge $\{u, v\}$ becomes stable, no move is going to be performed on this edge any more. Moreover, after the rule $R4$ sets $S_v[i] = u$ then $\{u, v\}$ becomes stable, unless $S_u[j] = v$ for some $j > D(u, v)$, which may happen if the state of u has not been cleared by $R0$ yet. Therefore, the following lemma holds.

Lemma 5. *If a value $S_v[i] = u$ was set by rule $R4$ and $S_u[j] \neq v$ for all $j > D(u, v)$, then the state of edge $\{u, v\}$ is stable and no more moves will be performed on this edge.*

Now we describe possible sequences of moves performed on a given edge.

Lemma 6. *Let u and v be a pair of adjacent vertices. If a value $S_v[i] = u$ was set by rule $R5$ or if it was set by $R4$ while $S_u[j] = v$ for some $j > D(u, v)$, then only rule $R3$ can be applied to set this value back to* null.

Proof. First, notice that just after rule $R5$ or $R4$ has been used by v none of rules $R0$–$R3$ involving color i is active at v. Thus, rule $R4$ must be performed on $\{u, v\}$ by u or $R5$ must set $S_u[i]$ to $w \neq v$ before $S_v[i]$ is set to null by any of rules $R0$–$R3$. However, the former cannot happen as long as $S_v[i] = u$. Therefore, neither $R1$ nor $R2$ can be used to change $S_v[i]$ to null. Moreover, since i is bounded by $D(u, v)$, rule $R0$ cannot be applied to set $S_v[i]$ to null, either. This implies that only rule $R3$ can set $S_v[i] =$ null, preceded by setting $S_u[i] = w \neq v$ by $R5$. □

Observe that rules $R0$–$R2$ clear all the fields of $S_v[\,]$ equal to u if some conflicts on an edge $\{u, v\}$ are detected. Therefore, combining this observation with Lemma 6 we obtain that rules $R0$–$R2$ can be performed only twice on a given edge $\{u, v\}$ (once for u and once for v). Thus, the following upper bound holds.

Proposition 1. *The total number of uses of rules $R0$–$R2$ does not exceed $2m$.*

Proposition 1 together with Lemmas 5 and 6 yield that $R4$ can be performed only twice on a given edge $\{u, v\}$ – once before $R0$ clears pointers beyond $D(u, v)$ in $S_v[\,]$ and $S_u[\,]$ and once after that. This provides the following proposition.

Proposition 2. *The total number of performances of rule $R4$ does not exceed $2m$.*

In the proof of Lemma 6 we have already remarked that after $S_v[i]$ is set to u by rule $R5$, it can be set back to null only by rule $R3$. However, $S_u[i]$ needs to be altered to activate $R3$ in vertex v. As long as $S_v[i] = u$, rule $R5$ cannot change $S_u[i]$. Therefore, the only way to activate $R3$ in vertex v is to perform $R4$ in vertex u. This yields the following lemma.

Proposition 3. *Assume that $S_v[i]$ was set to u by rule $R5$ and afterwards it was back set to* null *by $R3$. Then, rule $R4$ must have been performed on some edge $\{u, w\}$ adjacent to $\{u, v\}$ between uses of $R5$ and $R3$.*

Now we are ready to bound the number of moves performed according to rules $R5$ and $R3$, which together with Propositions 1 and 2 gives the following theorem. It states that at most $O(m\Delta)$ moves are performed before a stable state occurs.

Theorem 2. *Starting from any state, the algorithm obtains a stable state after at most $m(4\Delta + 4)$ moves.*

Proof. First, we bound the number of moves performed according to rules $R5$ and $R3$. Proposition 3 implies that between assignment $S_v[i] = u$ by rule $R5$ and $S_v[i] = \text{null}$ by $R3$ a move of u according to $R4$ is performed. However, one use of $R4$ by vertex u can be involved in at most $\deg(u) - 1$ sequences of moves $R5, R3$ of vertices adjacent to u. Therefore, since the number of moves $R4$ is bounded by $2m$, the number of moves $R5$ does not exceed $2m\Delta$, including final moves $R5$ performed on each edge. In the same way the number of uses of $R3$ is bounded by $2m\Delta$.

Moreover, the total number of moves performed according to rules $R0$–$R2$ and $R3$ does not exceed $4m$ by Propositions 1 and 2. Therefore, the number of moves is bounded by $m(4\Delta + 4)$. □

5 The number of colors

We have already remarked in Section 2 that the number of colors used by the algorithm does not exceed $2\Delta - 1$. Below we provide some examples of graph classes for which this bound is tight.

Proposition 4. *The number of colors used by the algorithm on some caterpillars is equal to $2\Delta - 1$ in the worst case.*

Proof. A caterpillar is such a tree, that each vertex of degree at least 3 is adjacent to at most two vertices of degree 2 or greater. Intuitively, a caterpillar is a graph obtained by attaching pendant edges to a path.

Consider a caterpillar of degree Δ and length 2Δ, with all vertex degrees Δ or 1. Then, for every pair of adjacent vertices u, v of degree Δ we have $D(u, v) = 2\Delta - 1$. Therefore, the algorithm can color such an edge with any color from range $[1, \dots, 2\Delta - 1]$. However, there are $2\Delta - 1$ such edges, thus $2\Delta - 1$ colors must be used. □

In a similar way we prove the following proposition.

Proposition 5. *If graph G is regular, then the number of colors used by the algorithm can be equal to $2\Delta - 1$ in the worst case.*

In particular, if $\Delta = 2$, then the number of colors used by the algorithm does not exceed $3 \leq \chi' + 1$.

Corollary 2. *The number of colors used by the algorithm for even cycles and paths does not exceed $\chi' + 1$. The algorithm gives an optimal coloring for odd cycles.*

Below we prove that the number of colors is close to χ' for stars and double stars, where a double star is a tree with $n - 2$ leaves.

Proposition 6. *Stars are colored with $\chi' = \Delta$ colors. Double stars are colored with at most $\chi' + 2 = \Delta + 2$ colors.*

Proof. The number of edges of a star is equal to Δ and every two edges are adjacent. Therefore, exactly Δ colors are used. For every pendant edge $\{u, v\}$ we have $D(u, v) \leq \Delta + 1$. Thus, all pendant edges are colored with colors from the range $[1, \ldots, \Delta]$. Moreover, there is only one not pendant edge which uses at most one additional color. □

6 Concluding remarks

In practical implementations it is often desirable for the stabilization time to be dependent on the severity of the faults which appeared in the system, i.e. a minor perturbation ought to result in quicker stabilization. This is the motivation to consider another aspect of performance of self-stabilizing algorithms, namely *fault-containment*, proposed by Ghosh, Gupta, Herman and Pemmaraju [8]. Let a *k-faulty state* be a state obtained from a legitimate state by perturbing local states of k nodes. The *fault gap* is one of the measures of fault-containment introduced in [8]. It is defined as the worst case time needed to transform a 1-faulty state into a legitimate state. The fault gap is preffered to be $O(1)$, which implies that perturbing a single node causes a constant number of moves. However, it is often difficult or impossible to obtain such performance.

As we remarked in Section 4, if an edge is stable, no move is going to be performed on this edge by the algorithm. Therefore, starting from a 1-faulty state, all the moves will be performed on the edges incident to the faulty vertex only. Moreover, in the same way as in proofs presented in Section 4, we can show that each such edge will be involved in $O(1)$ moves. It follows that the fault gap of the proposed algorithm is $O(\Delta)$. More quantitative results concerning fault-containment of self-stabilizing edge-coloring can be the goal of future study.

7 Acknowledgements

This research has been supported in parts by the Ministry of Education and Science, grants no. 3T11F 006 28 and 4T11C 047 25. Additionally, the second author is supported by the Foundation for Polish Science.

References

[1] G. Antonoiu and P. K. Srimani: A self-stabilizing distributed algorithm to construct an arbitrary spanning tree of a connected graph, *Comput. Math. Appl.* **30**, 1995, 1–7.

[2] J. Beauquier, A. K. Datta, M. Gradinariu and F. Magniette: Self-stabilizing local mutual exclusion and daemon refinement. *Chicago J. Theor. Comput. Sci.*, 2002.

[3] S. C. Bruell, S. Ghosh, M. H. Karaata, and S. V. Pemmaraju: Self-stabilizing algorithms for finding centers and medians of trees. *SIAM J. Comput.* **29**, 1999, 600–614.

[4] A. Czygrinow, M. Hańćkowiak and M. Karoński: Distributed $O(\Delta \log(n))$-edge-coloring algorithm, *ESA*, 2001, 345–355.

[5] S. Dolev: Self-Stabilization. MIT Press, 2000.

[6] E. W. Dijkstra: Self-stabilizing systems in spite of distributed control. *Communication of the ACM* **17**, 1974, 643–644.

[7] S. Ghosh and M. H. Karaata: A self-stabilizing algorithm for coloring planar graphs. *Distributed Computing* **7**, 1993, 55–59.

[8] S. Ghosh, A. Gupta, T. Herman and S. Pemmaraju: Fault-containing self-stabilizing algorithms. *Proc. 15th Ann. ACM Symp. on Principles of Dist. Comp.* 1996, 45–54.

[9] W. Goddard, S. T. Hedetniemi, D. Pokrass Jacobs and P. K. Srimani: Fault tolerant algorithms for orderings and colorings. *IPDPS*, 2004.

[10] D. Grable and A. Panconesi: Nearly optimal distributed edge colouring in $O(\log \log n)$ rounds. *Random Structures and Algorithms* **10**, 1997, 385–405.

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

[12] J.-H. Hoepman: Self-stabilizing ring orientation using constant space. *Inform. and Comput.* **144**, 1998, 18–39.

[13] I. Holyer: The NP-completeness of edge-coloring. *SIAM J. Comput.* **10**, 1981, 718–720.

[14] S. C. Hsu and S. T. Huang: A self-stabilizing algorithm for maximal matching. *Inform. Process. Lett.* **43**, 1992, 77–81.

[15] M. Kubale: Introduction to Computational Complexity and Algorithmic Graph Coloring. GTN, Gdańsk, 1998.

[16] M. V. Marathe, A. Panconesi, and L. D. Risinger: An experimental study of a simple, distributed edge coloring algorithm. *Proceedings of the twelfth annual ACM symposium on Parallel Algorithms and Architectures*, ACM Press, 2000, 166–175.

[17] T. Masuzawa and S. Tixeuil: A self-stabilizing link-coloring protocol resilient to unbounded byzantine faults in arbitrary networks. Technical Report 1396, Laboratoire de Recherche en Informatique, 2005.

[18] A. Panconesi and R. Rizzi: Some simple distributed algorithms for sparse networks. *Distributed Computing* **14**, 2001, 97–100.

[19] A. Panconesi and A. Srinivasan: Randomized Distributed Edge Coloring via an Extension of the Chernoff-Hoeffding Bounds. *SIAM J. Comput.* **26**, 1997, 350-368.

[20] S. Sur and P. K. Srimani: A self-stabilizing algorithm for coloring bipartite graphs. *Inform. Sci.* **69**, 1993, 219–227.

[21] V. G. Vizing: On an estimate of the chromatic class of a p-graph. *Diskret. Analiz* **3**, 1964, 25–30.

Received November, 2005