An Experimental Study of a Simple, Distributed Edge-Coloring Algorithm

MADHAV V. MARATHE
Los Alamos National Laboratory
ALESSANDRO PANCONESI
Universitá *La Sapienza* di Roma
and
LARRY D. RISINGER JR
Los Alamos National Laboratory

We conduct an experimental analysis of a distributed randomized algorithm for edge coloring simple undirected graphs. The algorithm is extremely simple yet, according to the probabilistic analysis, it computes nearly optimal colorings very quickly [Grable and Panconesi 1997]. We test the algorithm on a number of random as well as nonrandom graph families.

The test cases were chosen based on two objectives: (i) to provide insights into the worst-case behavior (in terms of time and quality) of the algorithm and (ii) to test the performance of the algorithm with instances that are likely to arise in practice. Our main results include the following:

(1) The empirical results obtained compare very well with the recent empirical results reported by other researchers [Durand et al. 1994, 1998; Jain and Werth 1995].
(2) The empirical results confirm the bounds on the running time and the solution quality as claimed in the theoretical paper. Our results show that for certain classes of graphs the algorithm is likely to perform much better than the analysis suggests.
(3) The results demonstrate that the algorithm might be well suited (from a theoretical as well as practical standpoint) for edge coloring graphs quickly and efficiently in a distributed setting.

Based on our empirical study, we propose a simple modification of the original algorithm with substantially improved performance in practice.

Additional Key Words and Phrases: Distributed algorithms, edge coloring, experimental analysis of algorithms, high performance computing, randomized algorithms, scheduling.

The work is supported by the Department of Energy under Contract W-7405-ENG-36, the ASCI initiative and the DOE AMS program. "An experimental study of a simple, distributed edge coloring algorithm," appeared in Proc. of 12th ACM Symposium on Parallel Algorithms and Architectures (SPAA), 2000, pp. 166–175.

Authors' addresses: Madhav V. Marathe, Basic and Applied Simulation Science (CCS-5), P.O. Box 1663, MS M997, Los Alamos National Laboratory, Los Alamos NM 87545; email: marathe@lanl.gov; Alessandro Panconesi, Dipartimento di Informatica, Universitá *La Sapienza* di Roma, via Salaria 113, 00198 Roma Italy. Most of the work was done while at BRICS, University of Århus, DK-8000 Århus C, Denmark; email: ale@di.uniroma1.it; Larry D. Risinger jr, ISR5, Los Alamos National Laboratory, P.O. Box 1663, MS J570 Los Alamos NM 87545. email: deanr@lanl.gov.

1. INTRODUCTION AND MOTIVATION

An *edge coloring* of a graph is an assignment of colors to the edges of the graph such that edges incident on the same vertex receive different colors. Equivalently, an edge coloring can be viewed as a partition of the edges into disjoint matchings. The minimum number of colors with which the graph G can be colored is called the *chromatic index* of G and is customarily denoted by $\chi'(G)$. In this paper we conduct an experimental study of a very simple distributed algorithm for the edge-coloring problem. Theoretical analysis [see Grable and Panconesi 1997] shows that this algorithm performs very well asymptotically, that is, it computes nearly optimal edge colorings very quickly. The algorithm can fail but, according to theoretical analysis, this happens with a probability that tends to 0 as the size of the input graph grows. The experimental analysis was motivated by certain applications of edge colorings in the context of scheduling (see below) and the desirable features of the algorithm (speed, quality, and simplicity of implementation). Our results show that the algorithm performs reasonably well in practice and might very well be suitable for real applications.

The kind of applications we have in mind are related to the scheduling of data transfers in large parallel architectures. The usefulness of edge coloring in this context was already noted by several authors. (See Jain et al. [1992b] for a comprehensive survey of applications arising in various areas.) Our original motivation for studying edge-coloring problems was the *gather-scatter problem* arising in the context of the Telluride project at Los Alamos [Kothe et al. 1997; Ferrell et al. 1997] . See `http://www.lanl.gov/asci/` and `http://public.lanl.gov/mww/HomePage.html` for more details about these projects.

A similar edge-coloring problem arises in other distributed computing applications [see Jain et al. 1992a, 1992b; Durand et al. 1998, 1994]. In all these applications a distributed network is required to compute an edge coloring of its own unknown topology. As a result, a simple and fast distributed algorithm for computing near-optimal colorings is preferable to a cumbersome centralized approach, even when the latter is able to compute optimal solutions. The reasons for this are twofold: (i) the task of finding good schedules for data exchange should be a small fraction of total computation time, and (ii) the amount of information that needs to be gathered at a central location will require prohibitive amounts of resources (e.g., time and memory).

The rest of the paper is organized as follows. Section 2 summarizes our results. Section 3 discusses related work. In Section 4, we describe the algorithm and discuss important implementation aspects. Section 5 provides an intuitive explanation of why the algorithm works. This provides a basis for the class of graphs on which subsequent empirical analysis is reported. In Section 7, we analyze our results on various graph classes.

2. SUMMARY OF RESULTS

In this paper, we conduct an experimental analysis of the following randomized, distributed algorithm (referred to as **S**).Initially each edge is given a list, or *palette*, of $(1+\epsilon)\Delta$ colors, where Δ is the maximum degree of the network (graph). The computation proceeds in rounds. During each round, each uncolored edge, in parallel, first picks a tentative color at random from its palette. If no neighboring edge picks the same color, the color becomes final and the algorithm stops for that edge. Otherwise, the edge's palette is updated in the obvious way—the colors successfully used by the neighbors are removed—and a new attempt is performed in the next round. Section 4 discusses the algorithm in greater detail.

Algorithm **S** is very simple and distributed, *yet* guarantees that the number of colors used is a small multiplicative factor of the optimal value, provided the maximum degree of the graph is not "too small" (see discussion below) [Grable and Panconesi 1997].

Here, we conduct an extensive empirical analysis of the algorithm **S**. The analysis is conducted by running the algorithm on a simulated *synchronous, message-passing, distributed network* environment. Such an environment is a theoretical abstraction of real distributed architectures where typically the cost of routing messages is orders of magnitude greater than that of performing local computations. In this abstract model of computation, we have a graph whose vertices correspond to processors and whose edges correspond to bidirectional communication links. There is no shared memory. Computation proceeds in *rounds*. During each round a processor performs some internal computation and exchanges information with the neighbors in the graph. The complexity of the protocol is, by definition, the number of rounds needed to compute the output correctly. In our example, this corresponds to a valid edge coloring of the network. In this model, local computation is not charged for, but communication is. Therefore, the model is somewhat orthogonal to the PRAM model.

We carry out our experimental analysis by simulating this theoretical model, as opposed to running the algorithm on a real parallel architecture, for the following reasons. Our simulation is aimed at ascertaining whether the simple stochastic process (**S**) described above shows the good behavior predicted by the analysis in real situations. Whereas this simulation is, in terms of resources and programming effort, relatively inexpensive, an implementation on a real parallel machine would be much more demanding. Moreover, a truly distributed implementation would introduce factors that would make the experimental results more informative for the specific applications considered, but probably unsuitable for drawing general conclusions about the behavior of the algorithm. There are also other reasons why our approach appears to be sound. One obvious problem with experimental studies of algorithms, in general, is that they are based on a specific implementation on a specific machine. This makes it very hard to compare different experimental results. Thus, it is useful to find a middle-ground between the world of asymptotics and real implementations, which on the one the hand provides accurate predictions on at least some important aspects of the algorithm, and on the other hand makes it possible to cleanly compare different experimental results.

For the experimental analysis, we consider (i) graph classes that bring forth the worst-case behavior of the algorithm, and (ii) graph classes that are likely to arise in practical situation. These are described in detail in the sequel (Section 5).

The input to the algorithm is a graph satisfying a certain condition on the maximum degree (denoted by Δ) of the graph. A precise statement of this condition is deferred to Section 4. Roughly, it says that the maximum degree of the graph should be high enough. In particular, it should be the case that $\Delta \gg \log n$, where n is the number of vertices of the input graph ($f(n) \gg g(n)$ if $f(n)/g(n)$ tends to infinity as n grows). The palette size—the number of colors that the algorithm is allowed to use—is also part of the input. In the analysis, we measure how the running time and the failure probability depend on the relevant variables: palette size, graph size, maximum degree of the graph, and the graph topology. Specific conclusions obtained as a result of our analysis include the following:

—**S**'s running time performance is extremely good. The algorithm colors random graphs with as many as half a million edges in about 10 communication rounds. For all practical purposes, the number of rounds is unlikely ever to exceed 15.

—In contrast, the failure probability is higher than anticipated and likely to make the algorithm not very useful. However, the following simple modification eliminates the problem: If some of the edges run out of colors during the course of the algorithm (this is the only way in which the algorithm can fail) they are simply given a few more fresh new colors and the algorithm continues as before. This modified algorithm is referred to as algorithm **M**. Notice that **M** remains a truly distributed algorithm, for the palette enlargement can be carried out locally. Algorithm **M** never fails but its performance must now be measured in terms of both running time and number of colors actually used. In practice, we found that **M** remains extremely fast—it never exceeds 3 or 4 additional rounds—and its color performance is satisfactory (see below).

—As remarked, for the algorithms to work well, the theoretical analysis prescribes that the initial degree be "high enough." Our analysis reveals that in some cases, this condition is critical (e.g., hypercubes) but in many other cases it can be relaxed.

—The theoretical analysis shows that asymptotically (i.e., for large n) the palette size can be pushed arbitrarily close to Δ (and hence to the optimum; recall that $\chi'(G) \geq \Delta(G)$ for all graphs G). In practice, however, we found that a more realistic value for the number of colors used by **M** is between 5% and 10% greater than Δ.

In view of the above observations, we expect the algorithms, especially algorithm **M**, to be useful if implemented on real distributed architectures.

3. RELATED WORK

The problem of finding efficient edge-colorings of a graph has been a subject of active research over the last three decades. The problem is known to be NP-complete [Hoyler 1980]. Numerous papers have been published on this topic—ranging from near-optimal sequential algorithms to randomized distributed algorithms. We refer the reader to Hochbaum [1997], Grable and Panconesi

[1997], Panconesi and Srinivasan [1992], Dubhashi et al. [1998], Jain et al. [1992a, 1992b], Durand et al. [1998], Gabow and Kariv [1982], and the references therein. We note that given a graph with maximum degree Δ, several sequential algorithms can provide solutions that color the graph using no more than $\Delta + 1$ colors. In contrast, obtaining such tight bounds in NC or under the distributed computing model is still an open question and a subject of active research.

In contrast to the theoretical work on this topic, very few papers have been published to experimentally study and tune the various theoretical algorithms and evaluate them in the context of specific scheduling and resource allocation tasks. We note that a number of heuristic methods have been proposed and experimentally tested for this and related problems. But unfortunately, these heuristics do not have worst-case performance guarantees, and in fact it is easy to devise examples where they fail. Moreover, the classes of graphs for which they fail is also not explicitly investigated. Finally, the experimental results are obtained typically only for a small class of random graphs.

The only exception that we are aware of in this context is the work of Durand, Jain and Tseytlin [Durand et al. 1998, 1994]. The algorithm described in Durand et al. [1998] is also a distributed, randomized, parameterized algorithm that can be tuned to specific applications. The work outlined in this paper differs from that in Durand et al. [1998] in the following significant ways:

(1) The algorithm of Durand et al. [1998] does not have a worst-case performance guarantee like the algorithm evaluated here. In contrast, as discussed later, the algorithm considered here has a bounded worst-case performance, both in terms of colors and the number of rounds.
(2) The algorithm given in Durand et al. [1998] computes an edge-coloring by removing one matching at a time. The number of such phases (i.e., of matching removals) is then at least Δ, so that the number of communication rounds is $\Omega(\Delta)$. In contrast, our algorithm takes $O(\log n)$ rounds, which can be, and in fact for the case we consider is, significantly less than Δ.
(3) The algorithm in Durand et al. [1998] works only for bipartite graphs (modeling client server type applications). In contrast our algorithm works for general graphs without any specific structure.
(4) Keeping the specific application in mind, the work of Durand et al. [1998] performs an experimental analysis for small graphs; namely random bipartite graphs $G(A, B, E)$ where $|A| = |B| = N \in \{16, 32, 64\}$ and with edge density 1/2. Here, in contrast, we consider classes of graphs on which the algorithm is likely to exhibit worst-case performance. Moreover, our analysis is much more extensive and we study both good and worst-case examples. It should also be noted that, because of the kind of applications we have in mind, we consider much bigger graphs.
(5) The research reported in this paper and in Durand et al. [1998] have different emphasis. Durand et al. were looking for schedules (i.e., colorings) as good as possible and were willing to spend a bit more time to obtain them. This makes sense in view of the very small graph sizes they consider. In our

case, we put more emphasis on speed rather than on schedule quality. The results they report are within 5% of the optimum, whereas our colorings lie between 5% and 15%. However, note that as opposed to our algorithms, their algorithms are somewhat optimized from an implementation point of view.

For the related problem of vertex coloring, a study similar in spirit to ours is that of Finocchi et al. [2002].

4. THE ALGORITHM

In this section, we describe the edge-coloring algorithms **S** and **M** precisely and discuss some of the issues that we expect to be clarified by the experimental analysis.

The input to the algorithms is a graph $G = (V, E)$ with maximum degree Δ, and a parameter $\epsilon > 0$. To each edge, we associate a palette of $(1+\epsilon)\Delta$ possible colors. Initially, the edge palettes are identical. Algorithm **S** repeatedly executes the following three steps, until all edges are colored or some palette runs out of colors. In the former case the algorithm terminates with success, in the latter it fails.

(1) Each uncolored edge $e = uv \in E$ chooses uniformly at random a tentative color from its associated palette.
(2) Tentative colors of the edges are checked against the colors of neighboring edges (edges that are incident on either vertex of the edge in question) for possible color collision. If a collision occurs, the edge rejects its tentative color.
(3) Edges that have chosen a valid color are marked colored, and their colors are removed from the palettes of neighboring edges.

Definition 1. In the sequel we shall refer to one execution of the three steps of the algorithm **S** and **M** as a *round*.

Clearly, the algorithm is *distributed* because the edges can work in parallel solely by exchanging information with the immediate neighbors. As discussed in Section 1, this algorithm was developed for the *synchronous, message-passing* model of computation. It is easy to see that the three steps of algorithm **S** can be implemented in $O(1)$ time in this model.

The asymptotic performance of algorithm **S** is characterized by a theorem proven in Grable and Panconesi [1997]. Here we state a simplified, somewhat less general version of the theorem, which is sufficient for our purposes. Recall that $f(n) \gg g(n)$, where f and g are functions taking values in the positive integers, means that $f(n)/g(n)$ tends to infinity as n grows.

THEOREM 1. *Let $\varepsilon > 0$ be arbitrary, but fixed. If the graph $G = (V, E)$ satisfies the condition*

$$\Delta(G) \gg \log n$$

where $n = |V|$, then, algorithm **S** *on input G and ε computes an edge coloring of G using at most $(1+\varepsilon)\Delta$ colors within*

$$O(\log n)$$

rounds in the synchronous, message-passing model of computation, with probability at least $1 - O(1)$, where $O(1)$ is a quantity that goes to 0 as n, the number of vertices, grows.

The degree condition roughly states that that Δ should be "much larger" than $\log n$. For instance, it is satisfied if $\Delta = \Omega(\log^{1+\delta} n)$, for an arbitrary, but fixed, $\delta > 0$,

As we shall see in the section devoted to analysis of the data, disappointingly for the "small" values of n we consider in this study, and they are the ones of practical interest, **S** has a high probability of failure. We have therefore also implemented and tested a simple modification of the algorithm, dubbed algorithm **M** (as in "modified"). Algorithm **M** is identical to **S** except for the first step of each round, modified as follows:

—**Modified Step 1:** Each uncolored edge $e = uv \in E$ chooses at random a tentative color from its associated palette. *If the palette is empty, one fresh new color is added to it.*

The only difference is that a new color is added if and when necessary. With this modification the algorithm is still distributed—each edge can decide locally whether to add a new color. Most importantly, algorithm **M** never fails.

Remark 1. Algorithm **M** is somewhat implementation dependent. Because of memory limitations, it is impossible to store a palette for every edge; thus we use vertex palettes instead. A vertex palette consists of the colors not used as final colors by any of the incident edges. Initially, each vertex has the full palette with $(1+\epsilon)\Delta$ colors. An edge palette now becomes simply the intersection of the vertex palettes of its two endpoints (see also next section). This results in substantial memory savings, since we ran the algorithm on graphs with a few hundred vertices but as many as half a million of edges. In fact, we could not have run our algorithms on such instances on a single processor machine without such a modification. This, however, creates some ambiguities as to how to implement algorithm **M**. We opted for the following perfectly symmetrical implementation. At the beginning of each round every vertex checks if any of its edges has an empty palette. If so, let c be the largest value among assigned and unassigned colors in the palettes of neighbors of the vertex, including the vertex itself. The new palette of the vertex is the set $\{1, \ldots, c+1\}$ minus the colors that are already assigned to edges incident on the vertex.

5. WHY THE ALGORITHM WORKS

Following Grable and Panconesi [1997], we give an intuitive explanation of why the algorithm works. The proof concerns D-regular graphs that are the most general case from a theoretical standpoint. The explanation motivates the choice of certain classes of graphs used in our experiments.

For the purposes of the discussion, we assume that a palette is assigned to each vertex instead of assigning a palette to each edge. The edge palette of an edge $e = uv$ is then equal to the intersection of the palettes of the two endpoints, that is

$$A(e) = A(u) \cap A(v) \tag{1}$$

When an edge receives its final color, the color is removed from the vertex palette of both vertices on which the edge is incident. It is easily seen that equation (1) remains valid throughout the execution of the algorithm.

The probabilistic analysis of Grable and Panconesi [1997] shows that, if we start with a D-regular graph, the process of edge and color removal generated by the algorithm is *quasi-random*. Note that at each step in the process, we remove edges that receive their final color as well as the colors that become unavailable. Informally, this means that the graph stays almost regular at all times and the vertex (not edge) palettes evolve almost independently of each other. More precisely, the vertex palettes evolve almost as if colors were removed independently at random from an identical initial palette of colors. This implies that, in expectation, the edges never run out of colors. To see why, consider two identical sets A and B of d colors each (think of them as the palettes of two neighboring vertices). If a $1 - \lambda$ fraction of colors is removed independently at random from both, the expected size of the intersection is $\lambda^2 d$. If we set $d = (1 + \epsilon)D$, by the time D colors are removed from both A and B—corresponding to the fact that all edges incident on u or v have been colored—the expected size of the intersection is strictly positive and has value $\epsilon^2/(1+\epsilon)^2 D$. This means that *in expectation* things go well: the algorithm never runs out of colors. The analysis in Grable and Panconesi [1997] shows that the random variables describing the process are sharply concentrated around their expectations.

The edges of the graph do introduce a correlation however—hence the qualification that the vertex palettes evolve "essentially" or "almost" independently. The theoretical analysis shows that, asymptotically, this effect is negligible. Here, we would like to investigate if this is indeed true in realistic situations. To gain some understanding, we have considered several kinds of topologies.

A first class of interest is *trees*. To see why, consider a graph consisting of two separate connected components A and B communicating only via an edge $e = ab$, with $a \in A$ and $b \in B$ (in graph theoretic terminology e is a bridge). Since A and B can communicate only through e, as long as e remains uncolored, the vertex palettes of a and b evolve completely independently—no approximation is involved. In a tree, every edge is a bridge. Therefore in trees, palettes of neighboring vertices evolve in a purely independent fashion as in the idealized situation described earlier. At the opposite end of the spectrum we find *cliques*, where the edges of the graph introduce all sorts of correlation between vertex palettes of neighboring vertices. Cliques are also interesting because of their high density (does density affect performance?) and because they are symmetric.

As long as an edge $e = uv$ remains uncolored it is clear that the random choices around u can affect v only via a path connecting u and v. A question of

(1) Quantities measured for algorithm **S**:
—running time, expressed as the number of rounds, as defined in Definition 1; and,
—failure probability, expressed as the percentage of failures.

(2) The two measures are functions of four independent variables:
(a) the size of the graph, expressed by n := (# vertices) and m := (# edges);
(b) the maximum degree of the graph, denoted by Δ;
(c) Initial palette size (IPS), that is the number of colors that the algorithm is allowed to use, denoted by $(1+\epsilon)\Delta$; and
(d) graph topology.

(3) The questions we are interested in:
—What is the running time?
—How small is the failure probability?
—How small can the initial palette size realistically be?
—How sensitive is the algorithm to the condition that $\Delta \gg \log n$?
—Is there a trade-off between palette size and speed, that is does the algorithm run faster if the palettes are initially bigger?
—How does the topology affect performance, that is for what kinds of graphs does it work well?
—How does the density of the graph affect performance?

(4) For algorithm **M** the dependent variables are:
—Running time; and,
—Number of colors used.

(5) Experiments were run on a Sun Sparc Ultra 1 with 200 MHz processor, with 256 Mbytes of main memory, running a standard Sun UNIX version 5. The compiler was a standard `cc` compiler using level 05 optimization option. For random graphs, each test run consisted of 5 runs on 5 different graphs, for each set of parameters. For nonrandom graphs each test run consisted of 25 algorithm runs for each set of parameters. In all our experiments, the variance of our observations is small and thus not explicitly discussed.

Fig. 1. Summary of experimental Setup.

interest is whether the *parity* of the paths affect the behavior of the algorithm. Therefore, we expect *bipartite graphs*, whether random or not, to be of interest, for they contain only even cycles.

Another class of bipartite graphs are *hypercubes*. These graphs are interesting also because they are Δ-regular graphs with $\delta = \log n$ and can therefore be used to test the criticality of the degree condition in Theorem 1. Hypercubes are also sparse, bipartite, and symmetric. We also considered *meshes* because they are likely to arise in practice. They are sparse and bipartite. Finally, we considered *random graphs* and *random bipartite graphs*.

6. EXPERIMENTAL DESIGN

To summarize the discussion of the previous sections, we will perform an experimental analysis of the algorithms **S** and **M**. The experimental set up is summarized in Figure 1.

Note that **M** never fails; thus the failure probability is irrelevant in this case. The independent variables (input parameters) are exactly the same, which includes the initial palettes' size. The question we are interested in remain the same, except that we are now interested in the final number of colors used instead of the probability of failure.

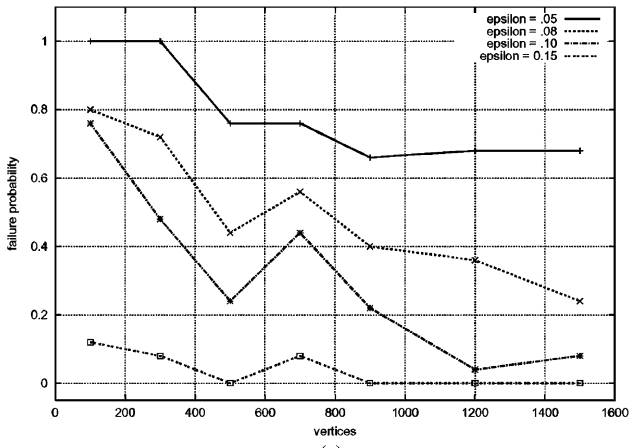
Graph topologies were chosen with the following aim in mind:

(1) graph classes that bring forth the worst-case behavior of the algorithm,
(2) Graph classes that are likely to arise in practical situations (meshes, sparse graphs, bipartite graphs, trees)

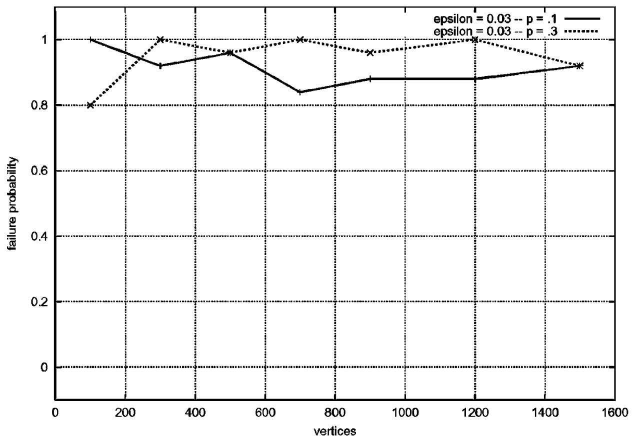
7. ANALYSIS

Some broad conclusions, which will be substantiated by the analysis of the data, presented subsequently, are as follows:

(1) Both algorithms **M** and **S** are extremely fast. We tried the algorithms on graphs with a few hundreds to several hundreds of thousands edges and both algorithms never took more than 13 rounds. Apart from graph size, speed was not affected by any of the independent variables (input parameters).
(2) Algorithm **S**'s probability of failure is quite high and likely to make it useless in real applications. In the interest of conciseness, we have not included the data on **S**'s running time. In any case, **S**'s running time is upper bounded by **M**'s running time, and the latter is extremely fast.
(3) In practice, algorithm **M** can be expected to use between 5% and 15% more colors than the optimum.
(4) Analysis of the data show that about 90–95 % of the edges get a color in the very first rounds (for sake of brevity we have not reported these data but they are available upon request). This implies that both algorithms have very low average communication cost because a large majority of the edges will exchange information with the immediate neighbors for just three or four communication rounds. We have not attempted a more precise estimation.
(5) The behavior of algorithm **S** shows a trade-off between initial palette size and failure probability (the larger the palette the smaller the probability). In contrast, for algorithm **M**, there appears to be no trade-off between initial palette size and speed, making the initial palette larger does not make the algorithm **M** any faster.
(6) The data shows the following trend. Consider two runs of algorithm **M** on the same graph. Let $(1+\epsilon_1)\Delta$ and $(1+\epsilon_2)\Delta$ be the initial number of colors, and c_1 and c_2 be the final number of colors used in the two cases. If $\epsilon_1 < \epsilon_2$ then $c_1 < c_2$. That is to say, if the algorithm starts with a smaller palette it will end up using less colors. At the same time, the running time is not affected significantly (or even detectably). This is a very appealing feature of algorithm **M**. The data shows, however, that it does not pay off to start with $\epsilon < 0.05$.
(7) Both algorithms **S** and **M** suffer on high density graphs and/or highly symmetric graphs. In particular, hypercubes, by far the worst topologies we tried, show that the condition $D \gg \log n$ cannot always be relaxed.



(a)



(b)

Fig. 2. Failure probability of **S** on graphs $G(n, 0.1)$ and $G(n, 0.3)$. (a) shows the failure probability for $G(n, 0.1)$. The four plots correspond to different IPS's (i.e., different ϵ). The performance improves as ϵ and n grow. On the right, the two plots show the failure probability attained with small IPS ($\epsilon = 0.03$) for $G(n, 0.1)$ and $G(n, 0.3)$.

Definition 1. Henceforth, IPS stands for Initial Palette Size. The IPS is always equal to $\lfloor (1+\epsilon)\Delta \rfloor$, where ϵ is an input parameter for the algorithms. Likewise, FPS will stand for Final Palette Size.

We begin by analyzing the data for random graphs.

7.1 Random Graphs

Random graphs on n vertices were generated as follows: Fix a parameter $p \in [0, 1]$, for each edge of an n-clique, a random number $r \in [0, 1]$ was picked, and the edge included if and only if $p \leq r$. This generates the distribution of random graphs known as $G(n, p)$.

Let us first consider the question of the failure probability of algorithm **S**. The results are summarized in Figure 2. They show that the probability of failure

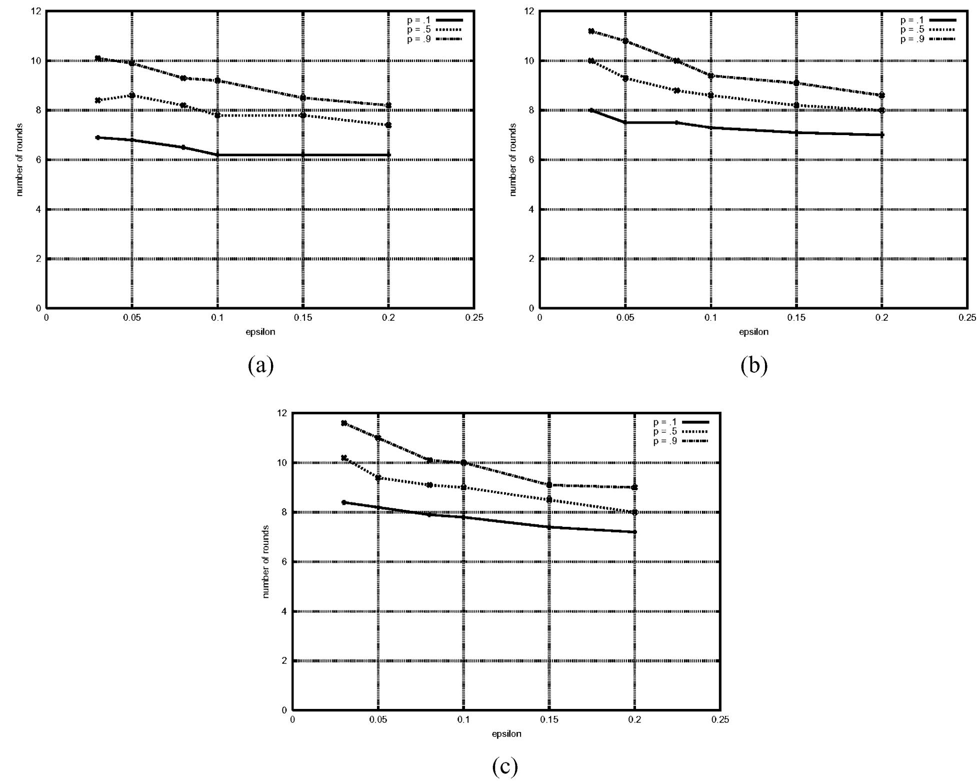
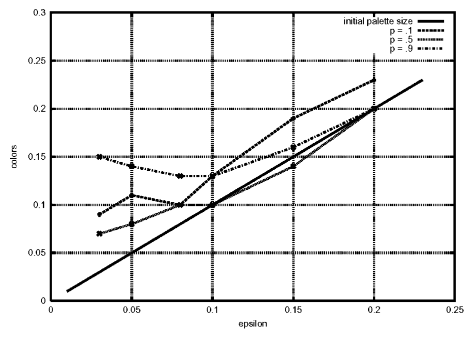


Fig. 3. Number of rounds used by **M** to color graphs in $G(n, p)$ as a function of the initial palette size. (a)–(c) show the running time for graphs in $G(100, p)$, $G(300, p)$, $G(500, p)$, respectively.

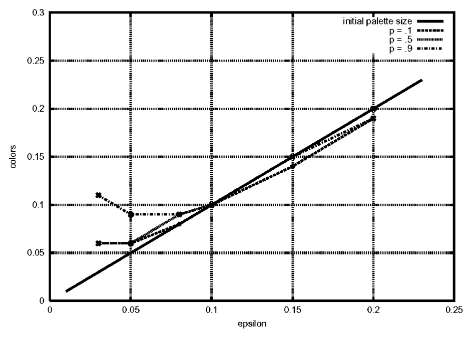
can be very high. Consistent with the theoretical analysis, the situation improves for larger values of n and for larger values of the initial palette size (i.e., greater ϵ). The trend is clear in Figure 2(a), which is relative to graphs drawn from $G(n, 0.1)$. Figure2(b) shows the results for small values of IPS ($\epsilon = 0.03$) for $G(n, 0.1)$ and $G(n, 0.3)$. Overall, the performance is far from satisfactory and for this reason we switched to the study of algorithm **M**. Recall that each test run consisted of 5 runs on 5 different graphs, for each set of parameters.

We now discuss **M**'s performance on graphs in $G(n, p)$. Figure 3 shows the number of communication rounds taken by **M** to complete the colorings for random graphs, for various values of n and p. **M** is very fast. In view of the high probability of failure of **S** this is a pleasant surprise. Its speed, confirmed by all our experiments, and the fact that **M** never fails make this algorithm quite useful. Note that the graphs we used in our tests have up to hundreds of thousand of edges and these are colored in about 10 rounds.

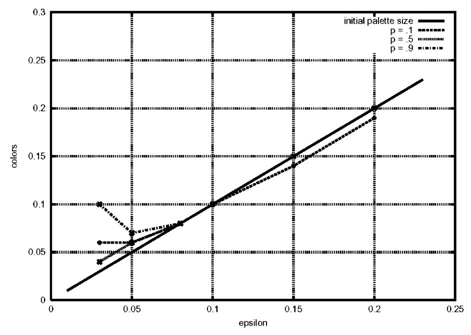
Figure 4 reports **M**'s performance in terms of colors used. The straight, thicker line represents ideal behavior, that is, an algorithm that starts with c colors and uses c colors. Since **M** uses fresh new colors when palettes are depleted, the final number of colors used can exceed the initial allotment. Overall the performance is pretty good. Figure 4 shows that the behavior is



(a)



(b)

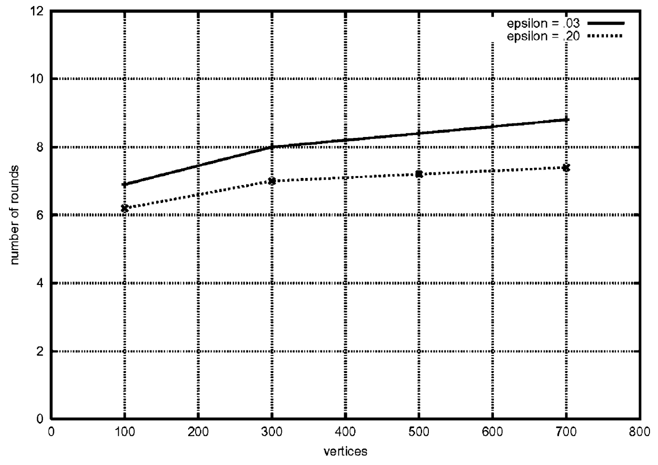


(c)

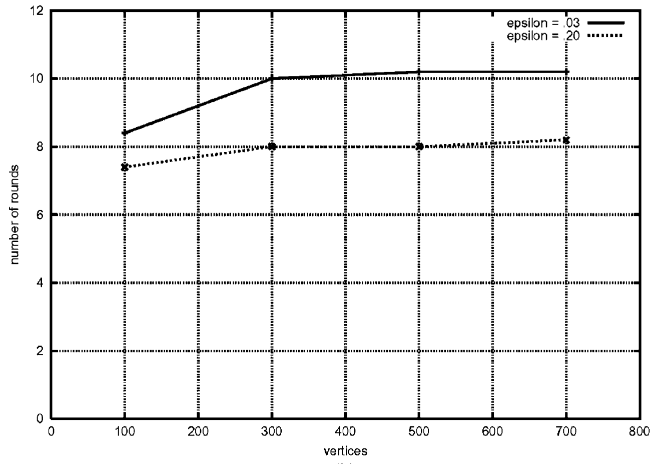
Fig. 4. Number of colors used by **M** to color graphs in $G(n, p)$ as a function of the initial palette size. (a)–(c) refer to graphs in $G(100, p)$, $G(300, p)$, and $G(500, p)$, respectively. The thick, straight line represents "ideal" behavior, that is an algorithm that starts with an IPS of c colors and uses c colors.

somewhat worse for sparse random graphs (Figure 4(a)) but it improves for denser graphs. Also, typically the performance is worse if the initial palette size is small (small ϵ), but it becomes "ideal" already for $\epsilon \approx 0.07$.

A natural question to ask is if and to what extent a bigger palette size makes algorithm **M** faster. Figure 5 compares the running time for two different values of the initial palette size, and it does so for different graph densities. Notice that the difference is almost negligible, never more than 2 rounds. Thus, it pays off to try a small initial palette size. Figure 6 quantifies the color savings that can be attained starting with a small palette. Since colors are chosen uniformly at random from the palette, and since there are many edges, if we start with a large palette size (e.g., $\epsilon = 0.20$) all colors will likely be used. This is shown clearly in the figure. The figure also shows that the performance for very small initial palette sizes is quite similar, but in any case better for smaller values. Since the resulting difference in speed is minimal, it pays off to start with a small palette size. The conclusion holds in general, since the behavior was observed for all experiments. However, in our tests **M** never used less than 1.05Δ colors.



(a)



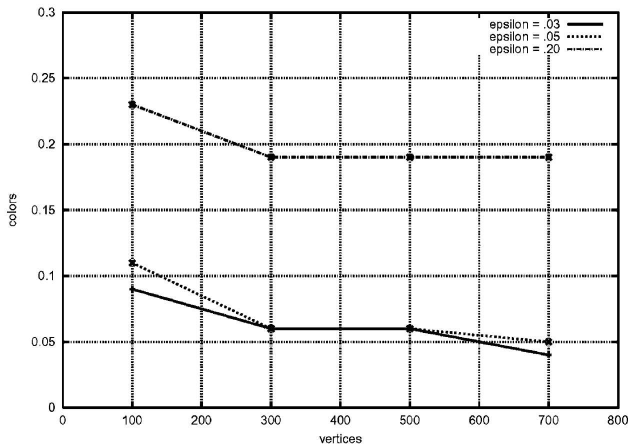
(b)

Fig. 5. Comparison of **M**'s running times with small versus large initial palette size for graphs in $G(n, p)$. (a) compares $\epsilon = 0.03$ vs. $\epsilon = 0.20$ for $G(n, 0.1)$. (b) compares $\epsilon = 0.03$ vs. $\epsilon = 0.20$ for $G(n, 0.5)$. The running time grows very slowly as n increases.

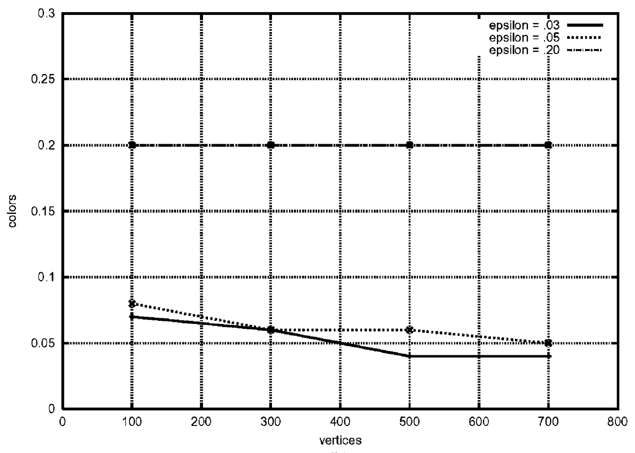
7.2 Random Bipartite Graphs

Bipartite random graphs on n vertices, n an even number, were generated as follows: Fixed a parameter $p \in [0, 1]$, for each edge of a complete $K_{\frac{n}{2},\frac{n}{2}}$ (complete bipartite graph with $n/2$ vertices on both sides of the bipartition), a random number $r \in [0, 1]$ was picked, and the edge included if and only if $p \leq r$. This generates the distribution of random graphs denoted here as $B(n, p)$.

We start by comparing the failure rates of algorithm **S** when graphs are drawn from $B(n, p)$ and $G(n, p)$. This is done in Table I. Recall that graphs in $B(n, p)$ have roughly half the number of the edges of graphs in $G(n, p)$. As the table shows, **S**'s failure probability for $B(n, p)$ graphs is even worse than that observed for $G(n, p)$, reinforcing the conclusion that algorithm **M** is the only viable alternative.



(a)



(b)

Fig. 6. Number of colors used by **M** with small versus big palette size, for graphs in $G(n, p)$. The initial palette size is $(1+\epsilon)\Delta$. (a) compares $\epsilon = 0.03$ vs. $\epsilon = 0.20$ for $G(n, 0.1)$. (b) compares $\epsilon = 0.03$ vs. $\epsilon = 0.20$ for $G(n, 0.5)$.

Running times for random bipartite graphs are displayed in two different ways in Figure 7. Figure 7(a) plots the running time against the number of vertices, while Figure7(b) plots the running time against the number of edges. The data confirm that algorithm **M** is very fast—the plots are almost flat.

Table II has the data for the number of colors used by algorithm **M**. From the observed data, we conclude that for graphs in $B(n, p)$, a realistic estimate for the minimum number of colors attainable is $\approx 1.07\Delta$ colors in the case of sparse graphs ($p = 0.2$) and $\approx 1.06\Delta$ in the case of dense graphs. This is slightly worse than the performance we saw for graphs in $G(n, p)$.

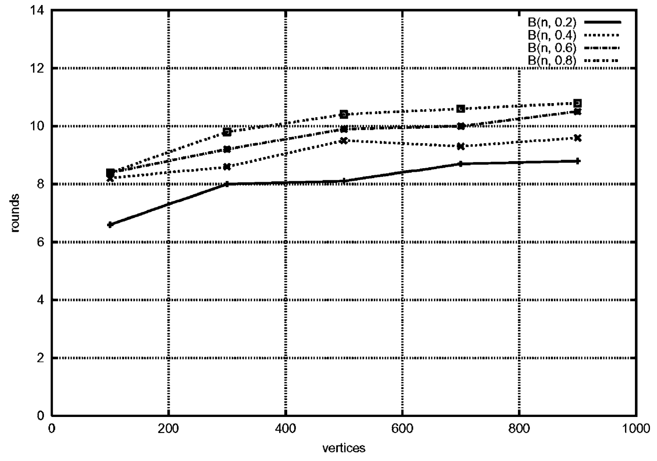
7.3 Two-Dimensional, Regular Meshes and Trees

The test runs for 2-dimensional, regular meshes are reported in Figure 8. Since each edge has exactly 6 neighbors, an IPS with 7 colors ensures that

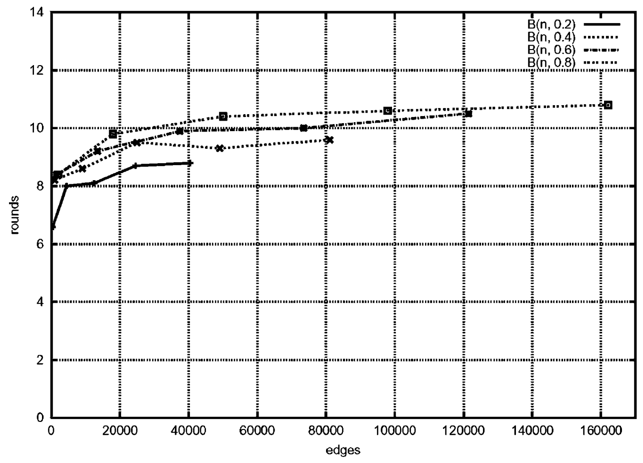
Table I. **S**'s Failure Rates for $G(n, p)$ and $B(n, p)$ when $p = 0.1$

Comparing **S**'s Failure Rates on $G(n, 0.1)$ vs. $B(n, 0.1)$		Number of Vertices (Note: $B(n, p)$ has $2n$ Vertices)				
IPS = $\lfloor(1+\epsilon)\Delta\rfloor$		100	300	500	700	900
$\epsilon = 0.05$	$G(n, 0.1)$	100%	100%	76%	76%	66%
	$B(n, 0.1)$	100%	84%	84%	96%	100%
$\epsilon = 0.08$	$G(n, 0.1)$	80%	72%	44%	56%	40%
	$B(n, 0.1)$	100%	60%	68%	52%	68%
$\epsilon = 0.10$	$G(n, 0.1)$	76%	48%	24%	44%	22%
	$B(n, 0.1)$	68%	84%	60%	56%	52%

Note that **S** consistently performs worse in the bipartite case.



(a)



(b)

Fig. 7. Number of rounds used by algorithm **M** to color graphs in $B(n, p)$ for $p \in \{0.2, 0.4, 0.6, 0.8\}$. (a) shows the number of vertices on the x-axis, while (b) shows the (average) number of edges.

Table II. Number of Colors used by Algorithm **M** to Color Graphs in $B(n, p)$

Number of Colors Used by **M** for Graphs In *B(n,p)*									
Graph Parameters:				IPS = $\lfloor(1+\epsilon)\Delta\rfloor$					
Vertices	Edges	P	Δ	$\epsilon = 0.03$	$\epsilon = 0.05$	$\epsilon = 0.08$	$\epsilon = 0.10$	$\epsilon = 0.15$	$\epsilon = 0.20$
100	505	0.2	18	0.07	0.07	0.07	0.11	0.12	0.17
	1003	0.4	28	0.14	0.11	0.12	0.11	0.19	0.21
	1503	0.6	37	0.14	0.16	0.14	0.14	0.16	0.20
	2000	0.8	47	0.14	0.13	0.12	0.13	0.16	0.20
300	4499	0.2	43	0.08	0.10	0.10	0.12	0.15	0.19
	9001	0.4	77	0.07	0.08	0.09	0.10	0.15	0.20
	13,449	0.6	106	0.07	0.08	0.09	0.10	0.14	0.19
	17,981	0.8	132	0.16	0.11	0.10	0.11	0.15	0.20
500	12,526	0.2	70	0.05	0.07	0.08	0.10	0.15	0.20
	24,981	0.4	122	0.06	0.07	0.09	0.10	0.15	0.20
	37,466	0.6	172	0.06	0.06	0.08	0.10	0.15	0.20
	50,031	0.8	218	0.12	0.08	0.09	0.10	0.14	0.20
700	24,506	0.2	93	0.05	0.07	0.09	0.11	0.15	0.20
	49,072	0.4	170	0.04	0.06	0.08	0.10	0.15	0.20
	73,510	0.6	239	0.05	0.06	0.08	0.10	0.15	0.20
	97,969	0.8	304	0.07	0.06	0.08	0.10	0.15	0.20
900	40,401	0.2	119	0.05	0.06	0.08	0.10	0.15	0.20
	81,088	0.4	213	0.05	0.06	0.08	0.10	0.15	0.20
	121,460	0.6	300	0.06	0.06	0.08	0.10	0.15	0.20
	162,088	0.8	386	0.07	0.06	0.08	0.10	0.15	0.20

The table shows the percentage of extra colors used. The algorithm starts with IPS = $\lfloor(1+\epsilon)\Delta\rfloor$ colors and ends up using with FPS = $(1+\delta)\Delta$. The table reports the value of δ, rounded to second decimal place.

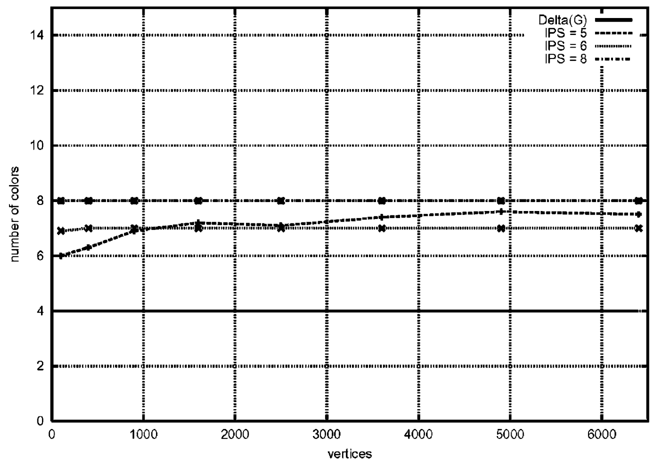
no palette will ever be depleted. The data show that sometimes the algorithm uses as few as 6 colors but this happens rarely. One might wonder whether allowing a larger IPS might significantly increase the algorithm's speed. Figure 8 shows that the answer is "no" and confirms that algorithm **M** is very fast.

We tested algorithms **S** and **M** on d-regular trees of height ℓ. These are rooted trees in which the root has height 0 and degree d and every other internal node has degree $d+1$. Therefore, $\Delta = d+1$, $n = (d^{\ell+1}-1)/(d-1)$, and $m = n-1$. In all tests $\Delta = 5$, while $1 \leq \ell \leq 8$. Recall that in trees there is no interaction among the (implicit) vertex palettes and therefore the process most resembles the idealized situation described in Section 5. Since $\Delta = 5$, nine colors are always sufficient to color any 4-regular tree. The data show that, as usual, there is no real speed up by using a larger IPS and that as few as seven colors are sometimes enough for up to a few hundred edges.

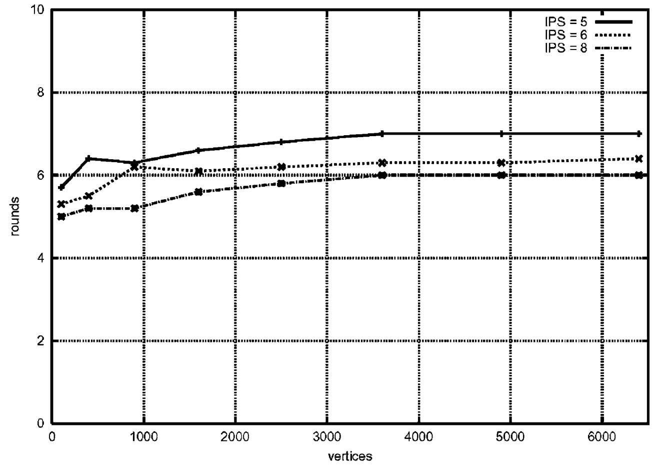
7.4 Cliques

The data for algorithm **M** are shown in Tables III–V and Figure 9. While the algorithm remains quite fast (Table III), the performance in terms of colors deteriorates when compared to other graphs.

Table III compares the outcome for cliques with that of random graphs of very high density ($p = 0.9$). The performance for dense random graphs is clearly better than that for cliques. Not only are the figures better, but the improvement



(a)



(b)

Fig. 8. (a) shows the number of colors used by algorithm **M** for 2-dimensional grids when the IPS is 5, 6, 8. (b) Shows the corresponding running times.

is faster as n grows. This perhaps indicates that it is not only density that makes things worse, but also symmetry (see discussion in Section 7.5).

7.5 Hypercubes

A *hypercube of dimension* d is a graph whose vertex set consists of all points of R^d whose coordinates are either 0 or 1. Two points $X = (x_1, \ldots, x_d)$ and $Y = (y_1, \ldots, y_d)$ are neighbors if and only if $x_i - y_i \in \{-1, 0, +1\}$ for all i. A d-hypercube therefore is a d-regular graph with $d = \log(\#$ number of points). The data for algorithm **M** are shown in Figure 10 and Table VI. Hypercubes were among the topologies chosen to see how sensitive the algorithm is to the condition $\Delta \gg \log n$, which is one of the hypothesis of Theorem 1.

Table III. Number of Iteration Required by **M** to Color 4-Regular Trees

Number of Rounds to Color 4-Regular Trees of Height ℓ						
Number of			Initial Palette Size = $\lfloor(1+\epsilon)d\rfloor$			
Levels	Vertices	Edges	$\epsilon = 0.25$	$\epsilon = 0.5$	$\epsilon = 0.75$	$\epsilon = 1.0$
1	5	4	3.0	3.4	3.2	3.1
2	21	20	4.9	4.2	4.6	3.9
3	83	84	5.4	4.8	4.5	4.0
4	341	340	6.1	5.4	5.0	5.0
5	1365	1364	6.0	5.7	5.3	5.1
6	5461	5460	6.6	6.2	6.0	5.5
7	21,845	21,844	7.0	6.5	6.1	6.0

Table IV. Number of Colors Used by **M** to Color 4-Regular Trees

Number of Colors Used by **M** for 4-Regular Trees of Height ℓ						
Number of			Initial Palette Size = $\lfloor(1+\epsilon)d\rfloor$			
Levels	Vertices	Edges	$\epsilon = 0.25$	$\epsilon = 0.5$	$\epsilon = 0.75$	$\epsilon = 1.0$
1	5	4	4.0	4.0	4.0	4.0
2	21	20	6.7	7.0	7.8	9.5
3	83	84	6.9	7.7	8.0	10.0
4	341	340	7.4	8.0	8.0	10.0
5	1365	1364	8.2	8.2	8.7	10.0
6	5461	5460	8.8	8.5	9.0	10.0
7	21,845	21,844	9.1	8.8	9.0	10.0

Here, $\Delta = 5$. The table reports the average number of colors used to color the tree.

Table V. Number of Rounds Used by Algorithm **M** to Color Cliques ($n = \Delta + 1$)

Number of Rounds Used for Cliques by Algorithm **M**							
		Initial Palette Size = IPS = $\lfloor(1+\epsilon)\Delta\rfloor$					
m	Δ	$\epsilon = 0.03$	$\epsilon = 0.05$	$\epsilon = 0.08$	$\epsilon = 0.10$	$\epsilon = 0.15$	$\epsilon = 0.20$
4950	99	10.7	10.7	10.2	10.2	9.4	8.7
44,850	299	12.0	11.8	11.1	10.6	9.6	9.0
124,750	499	12.6	12.0	11.2	10.6	9.6	9.0

The failure probability of algorithm **S** was disastrous—almost always near 100%. This is reflected in the poor performance of algorithm **M** in terms of number of colors used. This is apparent from Figure 10, which shows that not even an IPS of 1.5Δ colors suffices to complete the coloring. In our tests we tried IPS smaller than 1.5Δ but the number of colors used was always above the IPS. This was confirmed by tests with hypercubes with up to 8000 vertices (Figure 10 only shows values up to about 1000 vertices).

Recall that hypercubes do not satisfy the hypothesis of Theorem 1 (i.e., $\Delta \gg \log n$) but are "just below" that threshold. In order to test the sensitivity of the algorithm to the condition we tried the algorithm for random and random bipartite graphs with edge probability $p \approx \log n/n$. In turn this ensures that $\Delta \approx \log n$. Figure 10 shows that although algorithm **M** is sensitive to the condition, the performance is not nearly as bad as that for hypercubes. The figure only reports the data we obtained with random graphs, but the behavior with random bipartite graphs was quite similar, and hence omitted.

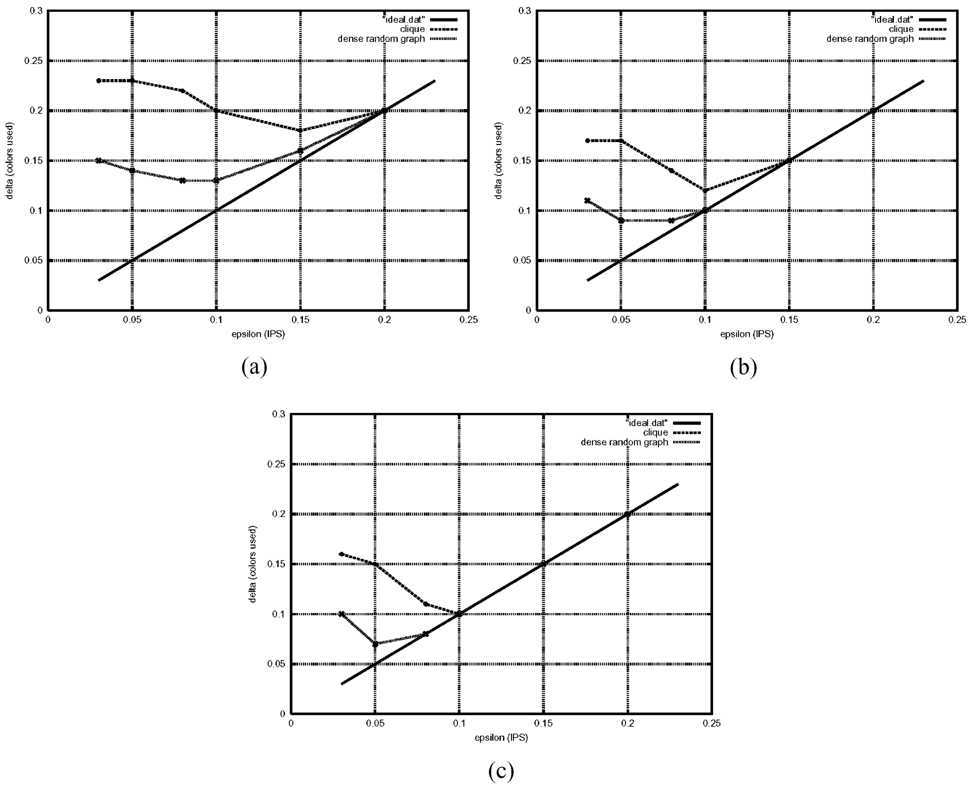


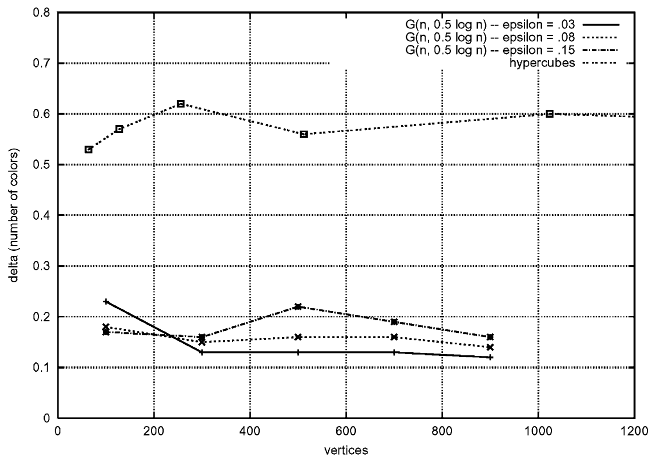
Fig. 9. Extra number of colors used by algorithm **M** to color cliques. (a) Compares the outcome for cliques with 100 vertices with that for $G(100, 0.9)$, for different values of the IPS. (b) and (c) Do the same with cliques of 300 and 500 vertices and $G(300, 0.9)$ and $G(500, 0.9)$. As always algorithm **M** starts with $(1+\epsilon)\Delta$ colors and ends up using $(1+\delta)\Delta$ colors.

One possible explanation for the bad performance with hypercubes is that the algorithm suffers when graphs are "symmetric." Recall that cliques, highly symmetric graphs, proved to be bad instances too (whereas very dense random graphs were just fine). A mathematical abstraction capturing the intuitive notion of symmetry in the case of graphs is the group of automorphisms of a graph. Intuitively, the larger the automorphism group the more "symmetric" the graph. Both hypercubes and cliques are very symmetric in this sense and the algorithms perform poorly on them. Vice versa random graphs typically have small automorphism group, and the algorithm performs well on them.

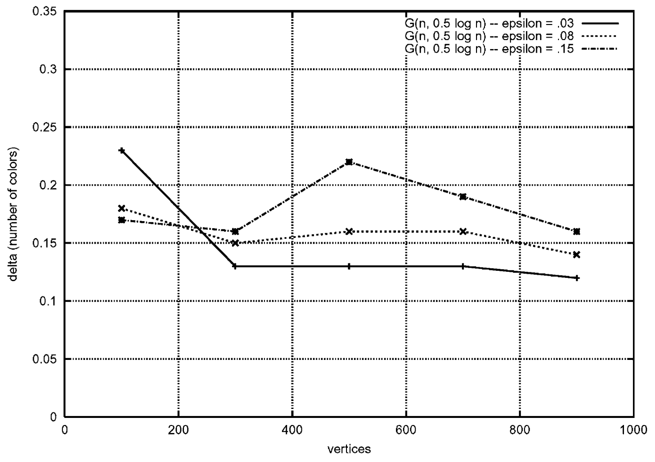
Although the performance is poor in terms of colors, **M** remains quite fast even with hypercubes, as shown in Table VI.

8. CONCLUDING REMARKS AND FUTURE WORK

We have conducted a detailed experimental analysis of a simple and theoretically "good" distributed algorithm for edge-coloring simple graphs. Our results show that a simple modification of the original algorithm is indeed an attractive candidate to be included in practical libraries of coloring algorithms.



(a)



(b)

Fig. 10. Number of colors used by **M** to color hypercubes and graphs in $G(n, p)$ with $\Delta \approx \log n$ (i.e. $p \approx \log n$). Although the performance for hypercubes is much worse, figure (b) indicates that if the condition $\Delta \gg \log n$ is not satisfied **M**'s performance in terms of colors used deteriorates.

Table VI. Number of Iteration by **M** to Color Hypercubes

Number of Iterations by Algorithm **M** to Color d-Hypercubes					
			Initial Palette Size = $\lfloor(1+\epsilon)\Delta\rfloor$		
$d = \Delta$	$n = 2^d$	m	$\epsilon = 0.5$	$\epsilon = 1.0$	$\epsilon = 1.5$
6	64	192	6.0	5.4	4.6
7	128	448	6.1	5.4	5.6
8	256	1024	6.7	6.0	5.2
9	512	2304	7.2	6.0	6.0
10	1024	5120	7.0	6.3	6.0
11	2048	11,264	7.3	6.3	6.0
12	4096	24,576	7.3	6.6	6.1
13	8192	53,248	7.8	6.9	6.2

As discussed earlier, there is no significant speed-up when the IPS gets larger.

ACKNOWLEDGMENTS

We thank Olaf Lubeck, Robert Ferrell and Joseph Kanapka for introducing us to PGSLib and working with us closely during the initial parts of the project. We thank Dr. Ravi Jain (Telecordia) for making available a copy of his papers. Finally, we thank Prof. T. Luzack, Prof. S.S. Ravi Dr. Vance Faber and Prof. Aravind Srinivasan for several discussions on related topics.

REFERENCES

Dubhashi, D., Grable, D., and Panconesi, A. 1998. Nearly-optimal, distributed edge-coloring via the nibble method. *TCS 203,* 4, 225–251. A special issue for the best papers of the 3rd European Symposium on Algorithms (ESA 95).

Durand, D., Jain, R., and Tseytlin, D. 1994. Distributed Scheduling Algorithms to Improve the Performance of Parallel Data Transfers. In *ACM SIGARCH Computer Architecture News*. ACM Press, 35–40. Special issue on Input/Output in Parallel Computer Systems.

Durand, D., Jain, R., and Tseytlin, D. 1998. Applying randomized edge-coloring algorithms to distributed communications: An experimental study. *TCS 203,* 4, 225–251. A special issue for the best papers of the 3rd European Symposium on Algorithms (ESA 95).

Ferrell, R., Kothe, D., and Turner, J. 1997. PGSLib: A library for portable, parallel, unstructured mesh simulations. In *Presented at the 8th SIAM Conference on Parallel Processing for Scientific Computing*.

Finocchi, I., Panconesi, A., and Silvestri, R. 2002. An experimental study of simple, distributed vertex colouring algorithms. In *Proceedings of the Thirteenth ACM-SIAM Symposium on Discrete Algorithms (SODA 02)*. ACM Press, 245–269. To appear in Algorithmica.

Gabow, H. and Kariv, O. 1982. Algorithms for edge-coloring bipartite graphs and multigraphs. *SIAM J. Comput. 1,* 11, 117–129.

Grable, D. and Panconesi, A. 1997. Nearly optimal distributed edge-coloring in $o(\log\log n)$ rounds. *RSA 10,* 3 (May), 385–405. Also In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA), 1997*, pp. 278–285.

Hochbaum, D., Eds. 1997. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, Boston, MA.

Hoyler, I. 1980. The NP-completeness of edge colorings. *SIAM J. Comput. 10*, 718–720.

Jain, R., Somalwar, K., Werth, J., and Browne, J. 1992a. Scheduling parallel I/O operations in multiple bus systems. *J. Parallel Distrib. Comput. 16,* 4, 352–362.

Jain, R., Werth, J., Browne, J., and Sasaki, G. 1992b. A graph-theoretic model for the scheduling problem and its application to simultaneous resource scheduling. In *Computer Science and Operations Research: New Developments in Their Interfaces*. O. Balci, R. Shander, and S. Zerrick, Eds. Penguin Press.

Jain, R. and Werth, J. 1995. Analysis of approximate algorithms for edge-coloring bipartite graphs. *IPL 54,* 3, 163–168.

Kothe, D., Ferrell, R., Turner, J., and Mosso, S. 1997. A high resolution finite volume method for efficient parallel simulation of casting processes on unstructured meshes. In *Presented at the 8th SIAM Conference on Parallel Processing for Scientific Comput*. LANL Report LA-UR-97-30, 14–17.

Panconesi, A. and Srinivasan, A. 1992. Fast randomized algorithms for distributed edge coloring. *SIAM J. Comput. 26,* 2, 350–368. Also in Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC) 1992.