Experimental analysis of simple, distributed vertex coloring algorithms *

Irene Finocchi
DISP, Tor Vergata
finocchi@disp.uniroma2.it

Alessandro Panconesi
DSI, La Sapienza
ale@dsi.uniroma1.it

Riccardo Silvestri
DSI, La Sapienza
silvestri@dsi.uniroma1.it

Abstract

We perform an extensive experimental evaluation of very simple, distributed, randomized algorithms for $(\Delta+1)$- and so-called Brooks-Vizing vertex colorings, i.e., colorings using considerably fewer than Δ colors. We consider variants of algorithms known from the literature, boosting them with a distributed independent set computation. Our study clearly determines the relative performance of the algorithms w.r.t. the number of communication rounds and the number of colors. The results are confirmed by all the experiments and instance families. The empirical evidence shows that some algorithms are extremely fast and very effective, thus being amenable to be used in practice.

1 Introduction

In this paper we perform an extensive experimental analysis of very simple, distributed vertex coloring algorithms. Their appealing features are simplicity of implementation, paired with a rather good performance in terms of colors used, and great speed. In particular, our algorithms can quickly compute vertex colorings using many fewer than Δ colors, where Δ denotes the maximum degree of the input graph. Because of these characteristics, we expect them to be quite useful in application areas, such as parallel sparse matrices computation or protocols for wireless networks.

1.1 The algorithms

In some sense this is a study of just one algorithm. It is by changing its few parameters and by switching on and off a heuristic step that several algorithms are obtained, with quite different characteristics. We now describe the basic algorithm. The input is an undirected graph G in which every vertex u has its own unique ID and is initially given a list (or palette) of available colors, denoted as L_u^0. The colors can be assumed to be consecutive natural numbers. The algorithm simply repeats the following *basic iteration*, until all vertices are colored. The current iteration is denoted by r.

The basic iteration:

1. *Wake up!* At the beginning of the iteration every vertex is *asleep*. In parallel, each uncolored vertex u *wakes up* with probability p_u^r.

*This work has been supported in part by the IST Programmes of the EU under contract number IST-2001-33555 (COSIN), and by the Italian Ministry of University and Scientific Research (Project "ALINWEB: Algorithmics for Internet and the Web"). A preliminary version of this work appeared in the Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA'02).

2. *Try!* Each awaken vertex u, in parallel, selects a *tentative* color t_u from its list L_u^r uniformly at random.

3. *Conflict resolution.* If no neighbor of u has selected the same tentative color, t_u becomes the *final* color of u. Otherwise the attempt fails and u will try again at the next iteration.

4. *Deliverance?* Vertices that obtain a final color exit the algorithm. The others, in parallel, update their lists by removing all colors assigned as final colors to the neighbors.

5. *Feed the hungry.* If the palette of a vertex u runs out of colors, fresh new colors are introduced in the following way. Let c be the greatest color used as final color in u's neighborhood. The new list of u is set to $[\min\{c+1, \Delta+1\}]$.

6. *Back to square one.* All vertices go back to sleep.

It is apparent that the algorithm is distributed. Every iteration (also called "round") only requires $O(\log n)$ message size and can be implemented in a constant number of rounds in the synchronous, message-passing model of computation, which is the one adopted here. This is a rough but useful theoretical model that approximates the behavior of those real distributed architectures for which the cost of routing messages is typically orders of magnitude greater than that of performing local computations. For instance, it has been used by several authors to analyze algorithm performance in wireless networks [2, 24, 25] .

Whereas this simulation is, in terms of resources and programming effort, relatively inexpensive, an implementation on a real distributed architecture 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.

In the course of our experiments we also made use of the following heuristic, dubbed here the *hungarian-folk step* (HS step).

> Given a color c, let G_c be the graph induced by all vertices that selected c as their tentative color at the current iteration.
> In parallel, for all colors c, compute an independent set of G_c as follows. Let π be a random permutation of the vertices of G_c: a vertex u enters the independent set if and only if it comes first than its neighbors in the total ordering induced by π.

When this is used, it replaces the conflict-resolution step of the basic algorithm. A crucial feature of the HS step is that it can be implemented very cheaply in our distributed setting. It does not require extra communication rounds and the message size is $O(\log n)$. It is well-known that, given a graph with n vertices, the HS step computes an independent set whose expected size is at least $n/(\hat{d}+1)$, $\hat{d}$ being the average degree of the graph. The very cute analysis appears to be part of hungarian folklore [19]. Luby showed that by iterating the HS step a maximal independent set is computed in $O(\log n)$ expected many rounds [22].

The random permutation of the HS step is needed to generate a random total ordering of the vertices. Strictly speaking, this requires global communication. If n, the number of vertices of the input network, is known, then it suffices to choose a random number in a large enough interval for a high probability analysis to go through. But in practice this is not a problem, because for any realistic network a few dozen random bits are enough to generate unique ID's. In fact, our experiments strongly suggest that using the original ID's works as well as truly (pseudo) random generated ones.

1.2 Our results

In this paper we perform two kinds of study. The first concerns algorithms for $(\Delta + 1)$-vertex coloring, while the second concerns so-called *Brooks-Vizing colorings*, i.e., colorings using "many fewer" than Δ colors [10].

Algorithms for $(\Delta + 1)$-vertex coloring. In this study the only dependent variable is the *running time* of the basic algorithm, measured as the number of communication rounds (basic iterations). The independent variables are three: the *wake-up probability*, the *initial palette setting*, and the HS step. The first is set to be a constant in the interval $(0, 1]$, while for the second the possibilities are two. If the initial lists are set to $[\Delta + 1]$, for all vertices, then we have the *global setting*; otherwise, if $L_u^0 := [\deg(u) + 1]$, for all vertices u, we have the so-called *local setting*. Likewise we shall speak of the *global* or the *local* version of the algorithm. Since both versions never run out of colors the Feed-the-hungry step can be omitted. Notice that in either case the algorithm has as many as $\Delta + 1$ colors at its disposal and since in practice, as we shall see, they will always all be used, it is not meaningful to consider the number of used colors as a dependent variable. Finally, the HS step can be either set on or off, i.e., it can be used as a conflict resolution mechanism instead of Step 3.

There exist theoretical analyses of the *trivial algorithm*, corresponding to setting the wake-up probability to 1 for all vertices and iterations, and of *Luby's algorithm*, corresponding to setting the wake-up probability to $1/2$, for all vertices and iterations [15, 21]. Both algorithms compute a $(\Delta + 1)$-coloring in $O(\log n)$ many expected iterations, and in fact do so with high probability [5, 17].

The main findings of our experimental study of these $(\Delta + 1)$-coloring algorithms are the following.

- The closer the wake-up probability to 1, the faster the algorithm, regardless of other parameters. The basic trend is exemplified by Figure 2a, showing the running time as a function of the wake-up probability for a random graph with 1000 nodes and edge probability 0.1. We remark that the same behavior was exhibited in all our tests, regardless of the graph. Thus the trivial algorithm is the algorithm of choice for $(\Delta + 1)$-coloring. In particular, when compared with Luby's algorithm it consistently turned out to be 2–3 times faster.

  The available asymptotic analyses do not explain this behavior [21, 15]. In this paper (Section 3) we propose a set of intuitive, approximate recurrences that predict the behaviour of the algorithm with extreme accuracy.

- Globalization does not help, meaning that the fully distributed local version is not slower and often results in non-negligible color savings.

- As remarked, the algorithms with the hungarian-folk heuristic work just as well if vertex ID's are used instead of generating a random permutation.

- The algorithms are extremely fast. It is often the case that graphs with a thousand vertices or more are colored within 5 communication rounds.

Algorithms for Brooks-Vizing colorings. The second part of our study concerns so-called *Brooks-Vizing colorings*, i.e., colorings using "many fewer" than Δ colors [10]. Here we have two dependent variables, the *running time*, measured as the number of communication rounds (basic iterations), and the *number of colors* actually used by the algorithm. The latter could exceed the initial allotment because the lists can run out of colors. The independent variables are three: the *wake-up probability*, the *shrinking factor*, and the HS step. For this study we considered the local setting only. Each vertex u is initially given a list of $\deg(u)/s$ colors, where $s > 1$ is the shrinking factor. Essentially our study focuses on three algorithms.

- `Algorithm GP`. This is the algorithm from [10]. Here the wake-up probability of vertex u at iteration r is set to be $p_u^r := |L_u^r|/\deg_r(u)$, where $\deg_r(u)$ is the number of uncolored neighbors of u at iteration r. This choice (with high probability) mantains the invariant that the number of neighbors vying for coloring at any given round equals the current size of the color lists.

- `Algorithm Hungarian-GP (HGP)`. This is the same as above, with the conflict-resolution step replaced by the hungarian-folk heuristic.

- `Algorithm Constantly-Hungarian (CH)`. Here the wake-up probability is a constant in the interval $(0, 1]$ and the hungarian-folk step replaces the conflict-resolution step.

The available theoretical evidence concerning Brooks-Vizing colorings can be summarized as follows. If G is square- or triangle-free then it is always possible to color G with $O(\Delta/\log\Delta)$ colors [18, 14]. Moreover, if G is not only triangle-free but also Δ-regular and of high enough degree (i.e., $\Delta \gg \log n$) then, with high probability algorithm `GP` colors the input graph with as few as $O(\Delta/\log\Delta)$ colors within $O(\log n)$ rounds [10]. The performance regarding the number of colors is the best possible in view of a result of Bollobás showing that there exist graphs of arbitrarily large girth whose chromatic number is $\Omega(\Delta/\log\Delta)$ [3].

The main conclusions of our study can be summarized as follows.

- The best algorithm for Brooks-Vizing colorings is the simplest of them all, namely algorithm `CH` with the wake-up probability set to 1. We called it algorithm `Trivially-Hungarian (TH)`. In all our tests it outperformed the competitors in terms of speed, while computing colorings of the same quality (same number of colors). The behaviour of algorithm `TH` is non smooth and irregular and therefore very hard to explain in terms of simple heuristic arguments, let alone by means of a rigorous probabilistic analyses. We leave these two problems as interesting and challenging open questions.

- A realistic value for the shrinking factor is a value between 4 and 6, even though in many cases greater savings can be obtained. While the running time of algorithms `GP` and `HGP` grows rather quickly as s increases, algorithm `TH`'s running time grows quite slowly. In practice, algorithm `TH` can be made to run with values of s up to 20 or more. Even if this might not result in colorings better than those obtained with smaller values of the shrinking factor, it might be worth a try since the running time stays quite small (between 20 and 30 rounds for graphs with several thousands of edges).

- The algorithms are very fast. Even using the largest (feasible) shrinking factor, graphs with hundreds or even thousands of vertices are typically colored within 20-30 rounds.

- While the theoretical analyses only deal with triangle-free graphs, the empirical evidence shows that Brooks-Vizing colorings can be quickly computed distributively also in the presence of many triangles. It would be nice if the theoretical analysis could be extended to give characterizations of classes of graphs admitting such colorings. At present the only case known to us is that of line graphs [11].

1.3 Comparison with previous work

In spite of a quite extensive literature on vertex-coloring (see for instance Culberson's bibliography [7]), there are just a few experimental studies of parallel or distributed algorithms.

Some are not quite related to the present work inasmuch as their experiments regard parallel algorithms designed for yielding very good colorings and thus they are neither simple nor fast (see, e.g., [20]).

Then, as far as we know, there are very few remaining papers experimenting with simple and fast parallel or distributed heuristics. Some of the studied heuristics derive from Luby's parallel algorithm for finding maximal independent sets [22]. The aim of the experimental study carried out by Jones and Plassmann [16] is showing that a specific parallel implementation of such an heuristic could be effective for bounded degree graphs. The performance of the heuristic is compared with that of simple sequential greedy heuristics (e.g., First Fit [12], Saturation Degree Ordering [4], Incidence Degree Ordering [6]). But no attempt is done to understand the behavior of the parallel heuristic as a function of the characteristics of the input graphs. Of a similar flavor is the experimental study by Allwright *et al.* [1], in which heuristics based on Luby's algorithm and on parallel variations of a greedy strategy are compared. The algorithms were only tested on random triangulated meshes. More recently, Gebremedhin and Manne [9] proposed a simple parallel heuristic and they studied a specific parallel implementation of it on just a few input graphs arising from numerical problems.

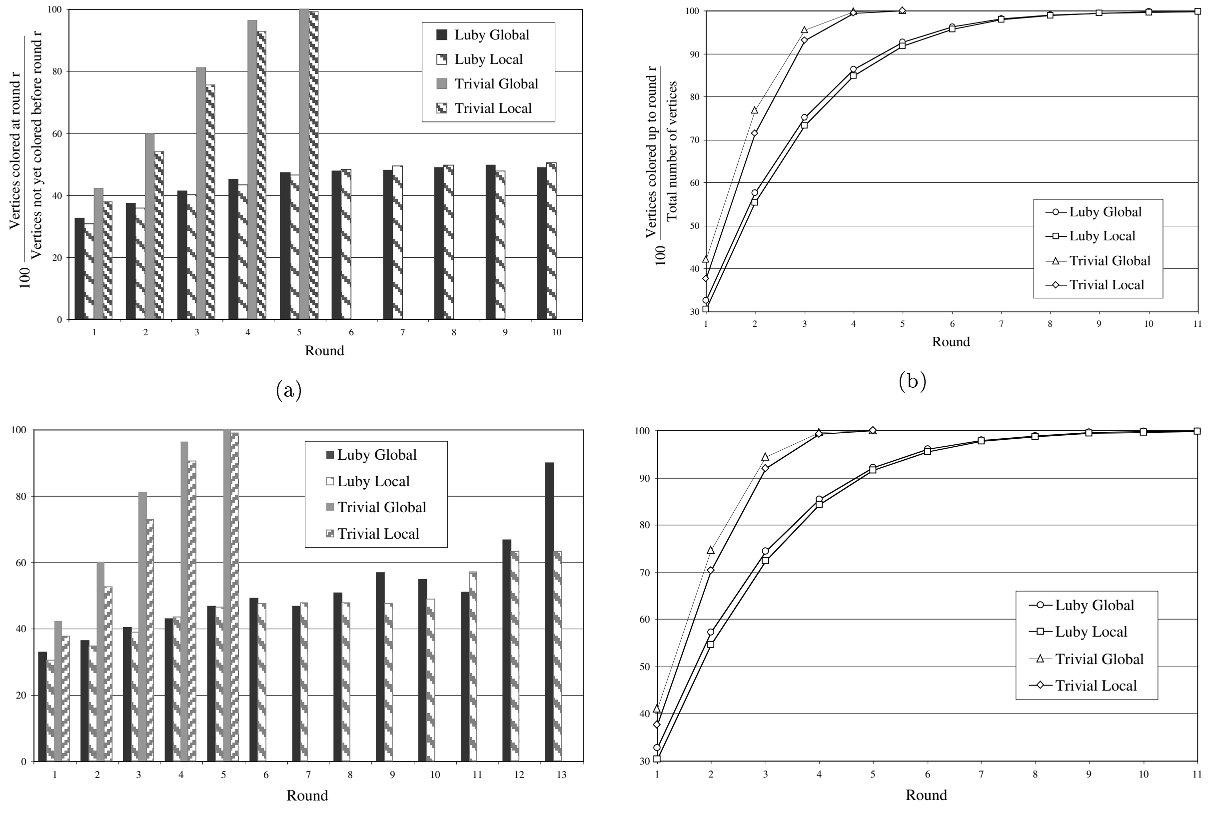
We note that the emphasis of all these studies is on testing specific parallel implementations of simple heuristics with respect to some restricted class of input graphs. Differently from this paper, the general behavior of the proposed heuristics is not analysed. For instance, which characteristics of the input graphs (e.g. maximum degree, average degree, k-partiteness, etc.) affect the number of used colors or the number of communication rounds. Under this aspect the experimental study reported here is much more similar in spirit to that of [23].

2 More on the experimental design

The algorithms considered in the experiments were described in sufficient detail in the introduction. In the following we give more details concerning the implementation and the instances that we have been using.

Implementation details. The algorithms were implemented in ANSI C. The performance indicators used in the experiments are not affected by machine and compiler details, since we simulate a message-passing distributed network for which the running time is given by the number of communication rounds. We used the pseudo-random generator provided by the ANSI C standard function `rand`, using odd seeds and randomly generating the sequence of seeds for each test starting from a base value. To increase the statistical confidence of our results, in all the experiments we performed 5 to 10 runs of the algorithms on the same instance. Moreover, in the experiments on randomly generated graphs we used several trials per data point, i.e., we ran the algorithms on several instances by the same parameters.

Instances. We tested the algorithms on random and synthetic graph families as well as on real test sets. In addition to random graphs from $G(n,p)$, i.e., random graphs with n vertices and edge probability p, we considered uniform random k-partite graphs from $G(n,p,k)$, where vertices are randomly assigned to one of k partition elements as nearly equal in size as possible (the smallest sets being one less than the larger) and a vertex pair uv is assigned an edge with probability p, provided u and v are from distinct partition elements. We have been also using Culberson's generators for coloring instances [7] and other test sets available from the FTP site related to the second DIMACS implementation challenge [8].



(a) (b)

(c) (d)

Figure 1: $\Delta + 1$ colorings: behavior of the trivial algorithm and of Luby's algorithm, with local and global palette setting, on instances drawn from $G(16000, 0.05)$ (upper row) and from $G(3000, 0.25)$ (lower row). The x axes show the round r. In the left graphs the y-axes show the percentage of uncolored vertices that get their final color at round r. In the right graphs the y-axes show the percentage of vertices that have a final color after round r.

In Section 3 and Section 4 we discuss the behavior of the algorithms on random graphs. DIMACS benchmarks and real test sets are briefly considered in Section 5. The interested reader can find additional data with the experimental package, that is available over the Web at the URL `http://www.dsi.uniroma1.it/~finocchi/experim/coloring/`.

3 Experimental results: $\Delta + 1$ colorings

In this section we experimentally analyze distributed algorithms for computing $\Delta+1$ colorings. The words "list" and "palette" are used as synonyms.

3.1 Effects of varying the wake up probability.

We first show that the wake up probability, denoted here as w, affects speed significantly. The charts in Figure 2 have been obtained by running the distributed algorithms with global and local palettes on random instances from $G(1000, 0.1)$ and from $G(1000, 0.1, 3)$. As shown in Figure 2a and 2c, the number of rounds of each algorithm is *inversely proportional* to the wake up probability, obtaining the fastest convergence for $w = 1$. *The very same trend was observed*

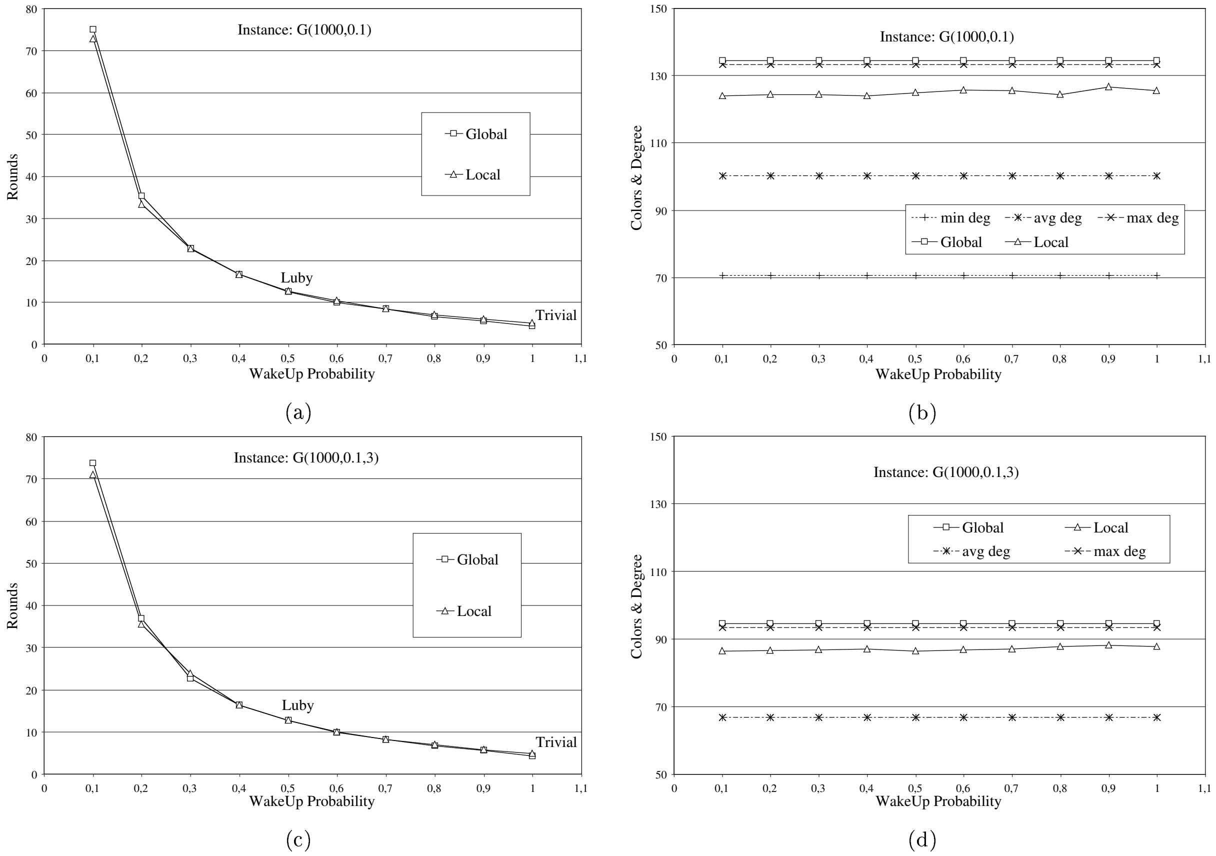


Figure 2: Behavior of the algorithms for $(\Delta + 1)$ coloring with global and local palette setting and increasing wake up probabilities. Instances are drawn from $G(1000, 0.1)$ and $G(1000, 0.1, 3)$. In terms of speed, the best wake up probability is 1. The number of colors used is independent of the wake up probability, but slightly better for local palettes.

in all our experiments. The choices $w = \frac{1}{2}$ and $w = 1$ correspond to Luby's algorithm and to the trivial algorithm, respectively, analyzed in [21, 15]. In both cases the theoretical analysis shows that the probability that a vertex colors at a certain round is at least $\frac{1}{4}$. The typical trend observed in the experiments and exemplified by Figure 2a provides strong empirical evidence that the trivial algorithm is always faster, thus suggesting that the analysis may not be tight.

We propose a heuristic explanation that fits the experimental data extremely accurately. Let us assume that the correlation induced by the edges is negligible. That is, the vertex palettes evolve independently as true random subsets. We also assume perfect symmetry, meaning that the probability that a vertex colours itself at a certain round is the same for all vertices, the probability that a certain colour disappears from a palette at a certain round is the same for all colours and paletts, and so on. These ideal conditions are approximated very well by sparse, regular graphs initialized with local palettes. Let us denote with s the initial palette size, with c_r the probability that a vertex is coloured at round r, and with d_r the vertex degree at round r. From the assumptions above it follows heuristically that these quantities are exactly the same for all r and for all vertices and palettes. The probability that a colour is selected tentatively at any round is $\frac{1}{s}$, since the algorithm treats colours symmetrically. Assuming complete independence then

$$c_{r+1} = w \left(1 - \frac{w}{s}\right)^{d_r} \tag{1}$$

where w is the wake up probability. Then, the new degree is simply,

$$d_{r+1} = (1 - c_{r+1})d_r. \tag{2}$$

Figure 3 shows that the prediction of these simple recurrences predict the behaviour of the algorithm extremely accurately. They fail to give good enough predictions only when the edge density is beyond ~ 0.9. We will deal with this very dense case by means of refined recurrences in § 3.3 below.

Concerning the different behaviour between the local and global version of the algorithm, it is apparent that, in the global case, if the number of vertices is much larger than Δ, in the global case all available colors will be used. In the local case, on the other hand, the smaller the number of high degree vertices, the higher the probability that the final color will use less than $\Delta + 1$ colors. This intuition is confirmed by the charts in Figure 2b and 2d: using local palettes leads to slightly better colorings, though the number of colors used is always greater than the average degree of the graph. Figure 6 shows that significant color savings that can be obtained by using the local version (see below).

3.2 Running time analysis.

The $(\Delta + 1)$ coloring algorithms are extremely fast: it is often the case that graphs with a few thousand vertices or more are colored within 5 communication rounds. The charts in Figure 4 plot the running time of the trivial and Luby's algorithms with global and local palette settings against the number n of vertices of the graph. Instances are from $G(n, 0.05)$ and $G(n, 0.05, 3)$. The edge probability was kept low in order to deal with memory limitations manifesting themselves for large values of n. (As we will see later, this is not restrictive, as dense graphs appear to be no more difficult to color than sparse ones.) Note that curves grow very slowly, especially those related to the trivial algorithm, which is remarkably faster the Luby's algorithm: 5 rounds versus more than 15. As suggested also by the previous experiment, the

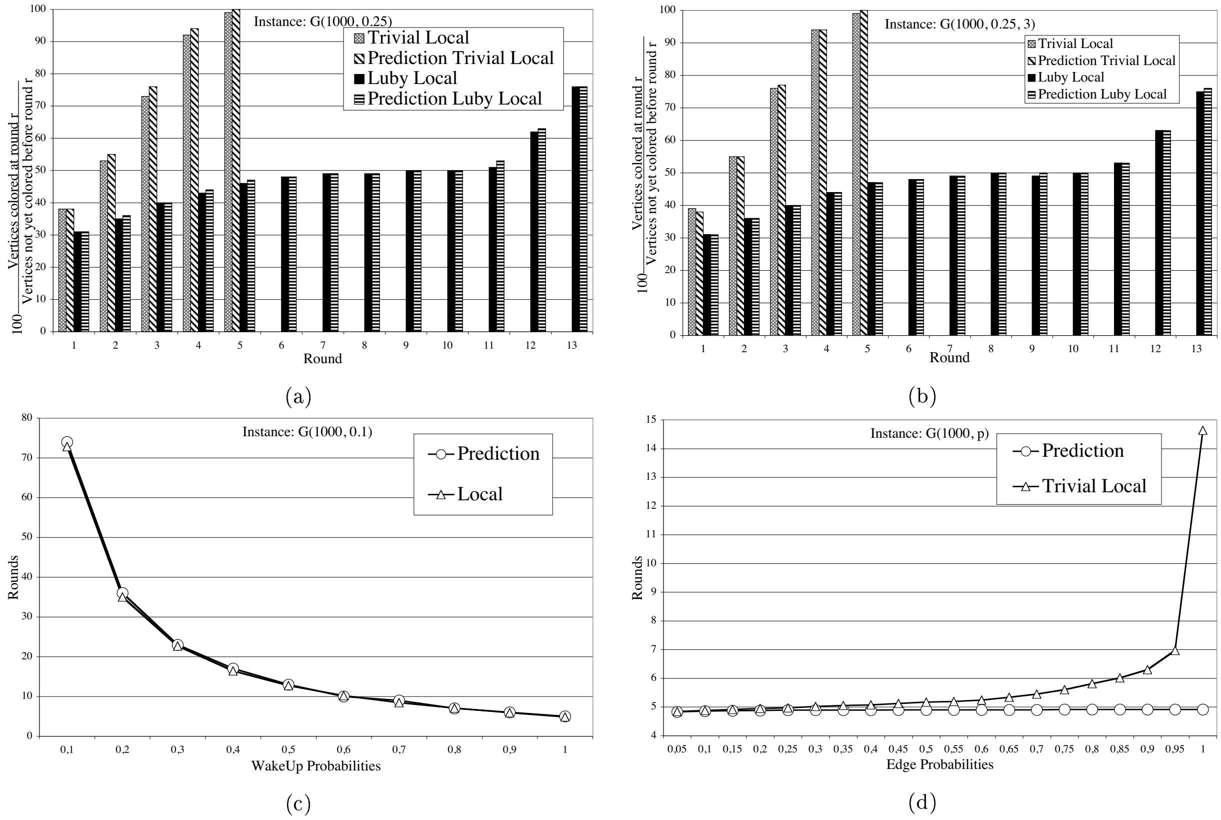


Figure 3: Simple prediction vs experiment: the simple recurrences of § 3.1 accurately predict both the probability that a vertex is coloured at a given round and the average number of rounds. They fail to give good enough predictions only when the edge density is very high as it is shown in (d).

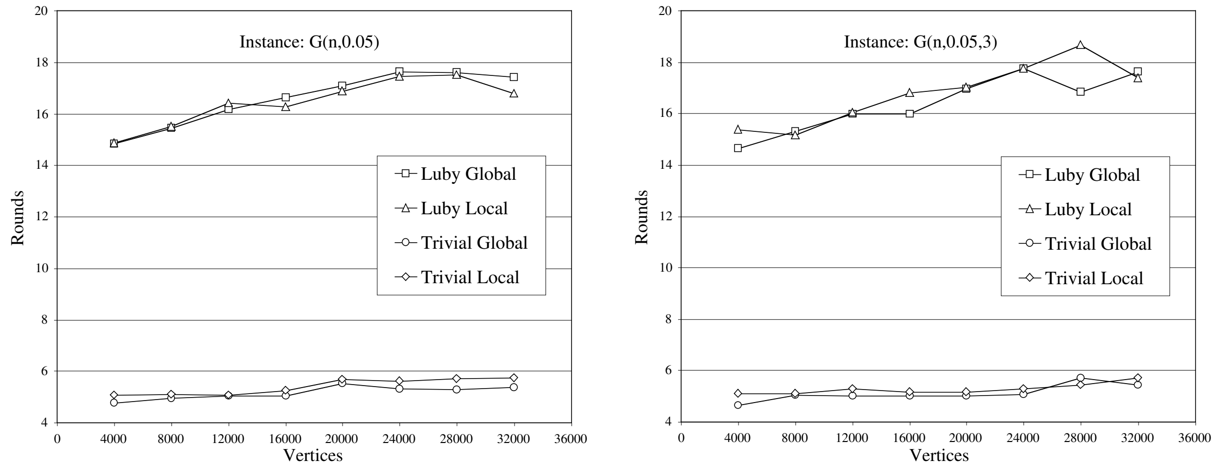


Figure 4: $(\Delta + 1)$ colorings. The running time, expressed as the number of rounds, is plotted against graph size, expressed as number of vertices. The figure shows the performance of Luby and the trivial algorithm. Note that the latter is consistently about three times faster, both with local and global palette settings.

running time benefits only at a very limited extent of the global setting: thus, it is worthwhile using the fully distributed local version of these algorithms, that often results in non-negligible color savings (see Figure 6).

3.3 Experiments with different graph densities.

In order to analyze how the number of rounds and colors depends on the density of the graph, we ran the trivial algorithm with global and local palettes on several graph instances of varying density. We consider the trivial algorithm only, because it consistently proved itself to be the best. Figure 5 reports the results for random instances from $G(1000, p)$ and $G(1000, p, 4)$, with edge probability p increasing from 0.05 to 1. Although global palettes guarantee faster convergence (Figure 5a and 5c), the effect is once again rather negligible. Notice that the algorithms are extremely fast even on dense instances.

The number of rounds does not depend very much on the density of the graph except for densities beyond 0.9. After this point the number of rounds grows considerably. In particular cliques require significantly more rounds to be colored. This behavior has been confirmed also on random k-partite graphs: on these instances the number of rounds stays almost constant. This is consistent with what was observed earlier since the edge density of these graphs cannot come too close to 1. Figure 5b and 5d show the percentage of rounds saved when the HS step is used. The saving is roughly 25%, with both local and global palette, and therefore significant.

We observed that the number of colors increases linearly with the density, following the trend of the maximum degree curve, and that the benefit of using local palettes is bigger for sparser graphs, as shown by Figure 6. The percentage of colors saved is approximately 7% for $p = 0.1$ and decreases linearly with the density down to 0 for $p = 1$. Colorings obtained on cliques by local and global palettes are obviously the same.

We now give a heuristic explanation that has very good predictive power also in the case of very dense instances. It is not surprising that the simple recurrences 1 and 2 do not yield good predictions when the edge density is very high. The dependence among the palettes grows with the edge density and when the edge density is at the maximum (i.e. the graph is complete) the palettes are fully dependent. Thus, to improve the prediction, it is necessary to take into account how the palette of a vertex depends on the palettes of its neighbours. To this end we estimate the expected number of colours in the intersection of the palettes of two adjacent vertices. Let us denote with s_r the size of a palette and with t_r the size of the intersection of the palettes of two adjacent vertices, at the end of round r. Let f_r be the probability that two adjacent vertices select the same tentative colour at round r. We approximate f_r by $t_{r-1}/(s_{r-1})^2$. To find an approximate recurrence for s_r and t_r we reason as follows. Let S be the palette of a vertex v at the beginning of round r and let R be the set of the final colours obtained by the neighbours of v at round r. The palette of v at the end of round r is $S' = S - R$ and so $|S'| = |S| - |T|$ where $T = S \cap R$. We need to estimate $|T|$. Let X^S and Y^R be two indipendent and uniformely distributed random variables on S and R respectively. That is, for any $c \in S$ and any $c' \in R$, $\Pr(X^S = c \wedge Y^R = c') = 1/(|S||R|)$. We believe that $\Pr(X^S = Y^R)$ can be approximated by f_r. On the other hand, we can compute it as follows:

$$\Pr(X^S = Y^R) \;=\; \sum_{c \in T} \Pr(X^S = c \wedge Y^R = c) \;=\; \sum_{c \in T} \frac{1}{|S||R|} \;=\; \frac{|T|}{|S||R|}.$$

Thus, $|T| \;=\; \Pr(X^S = Y^R)|S||R| \;\approx\; f_r|S||R|$. Hence,

$$|S'| \;=\; |S| - |T| \;\approx\; |S| - f_r|S||R| \;=\; |S|(1 - f_r|R|).$$

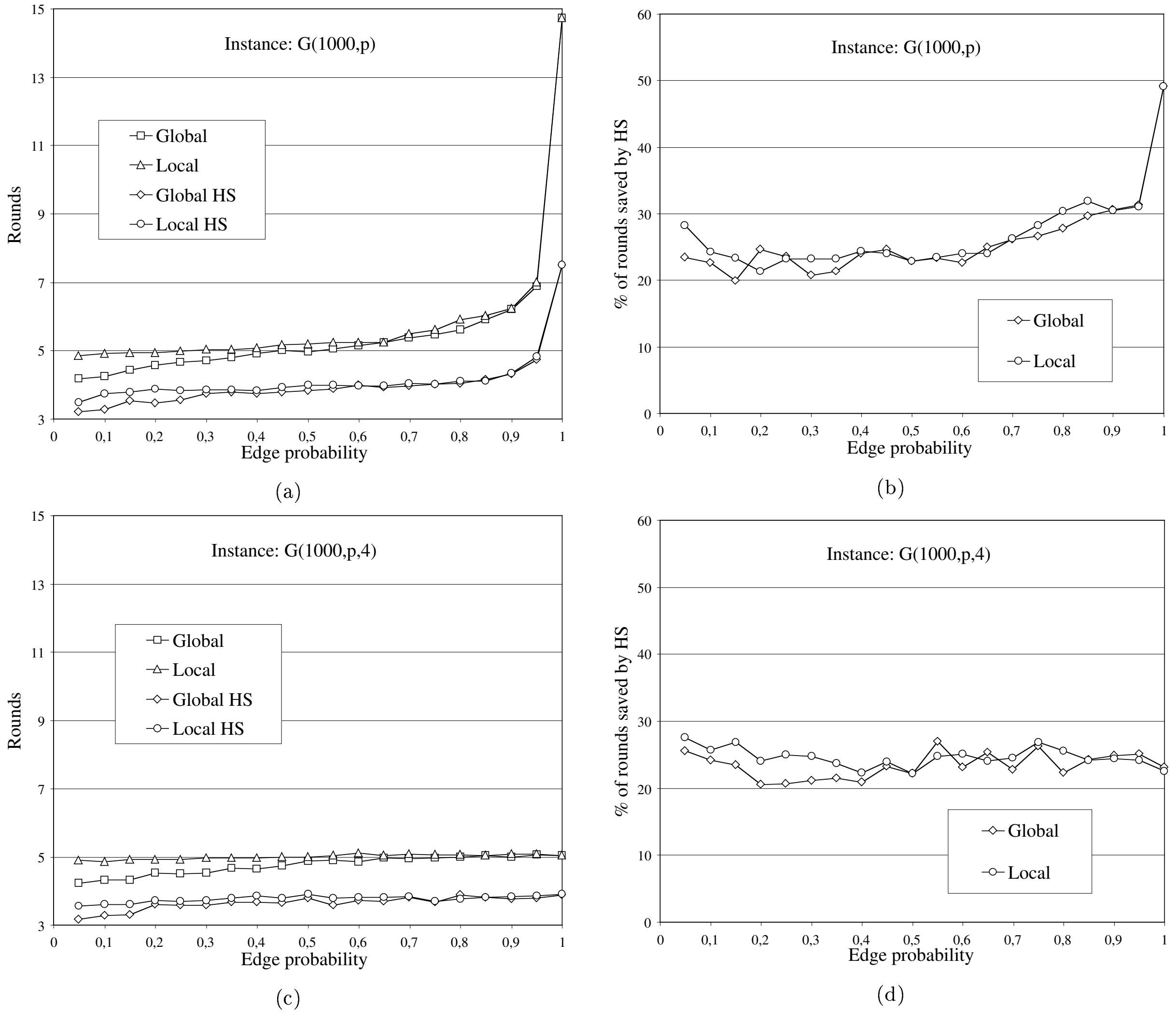


Figure 5: $(\Delta + 1)$ colorings: behavior of the trivial algorithm on instances drawn from $G(1000, p)$ and $G(1000, p, 4)$, with increasing edge probability. The number of rounds (y-axis) does not depend very much on the density (x-axis), except beyond the point $p = 0.9$. The HS step saves approximately 25% rounds.

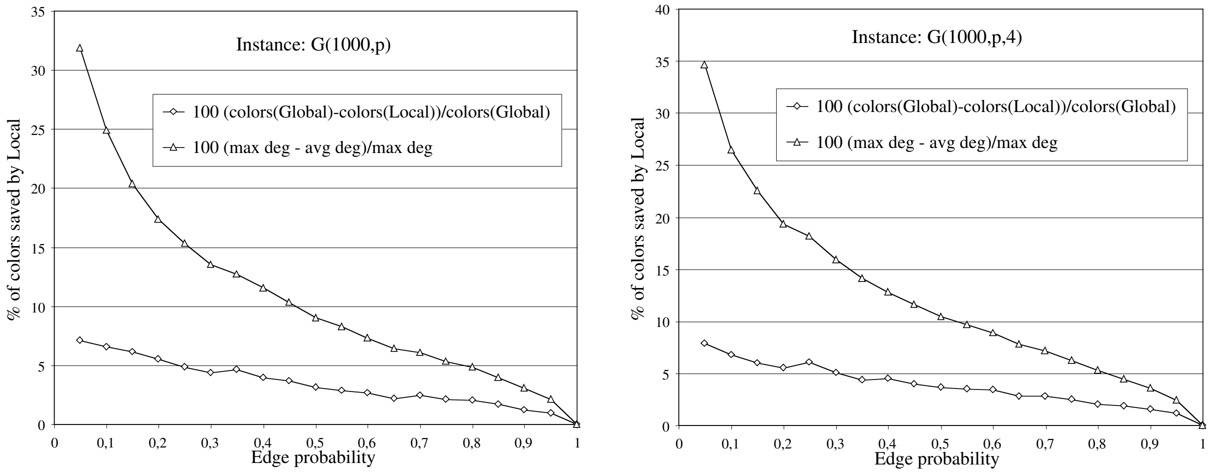


Figure 6: $(\Delta + 1)$ colorings: behavior of the trivial algorithm on instances with increasing edge probability drawn from $G(1000, p)$ and $G(1000, p, 4)$. The benefit of local palettes is greater on sparser instances.

This leads us to the following recurrence:

$$s_r = s_{r-1}(1 - f_r \rho_r)$$

where $\rho_r = |R|$ is the number of distinct final colours obtained by the neighbours of a given vertex at round r. By means of a similar argument we also get an approximate recurrence for t_r:

$$t_r = t_{r-1}(1 - f_r \overline{\rho}_r)$$

where $\overline{\rho}_r$ is the number of distinct final colours obtained by the neighbours of either of two given adjacent vertices at round r. It remains to find an approximation for ρ_r and $\overline{\rho}_r$. Firstly consider ρ_r. We already have an approximation for the number of neighbours of a given vertex that get a final colour at round r, that is, $c_r d_{r-1}$. So it suffices to estimate the number of *distinct* colours among the $c_r d_{r-1}$ obtained final colours. Let $m = c_r d_{r-1}$ and assume that the m vertices are numbered from 1 to m. We can represent that situation by means of the following random variables, for any $i, j \in \{1, 2, \ldots, m\}$,

$$X_{i,j} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ get the same final colour} \\ 0 & \text{otherwise} \end{cases}$$

It is easy to verify that the number D of distinct colours is given by

$$D = \sum_{k \geq 1, \{i_1, \ldots, i_k\} \subseteq \{1,2,\ldots,m\}} X_{i_1,i_2} \cdots X_{i_{k-1},i_k} \prod_{j \in \{1,2,\ldots,m\} - \{i_1,\ldots,i_k\}} (1 - X_{i_1,j})$$

where $X_{i_1,i_2} \cdots X_{i_{k-1},i_k}$ is valued 1 when $k = 1$. By assuming that all the variables are identically distributed with $\Pr(X_{i,j} = 1) = p$ and that they are fully indipendent, for the expected value of D it holds that

$$\begin{aligned} \mathrm{E}(D) &= \textstyle\sum_{\emptyset \neq C \subseteq \{1,2,\ldots,m\}} p^{|C|-1}(1-p)^{m-|C|} \\ &= \textstyle\sum_{k=1}^{m} \binom{m}{k} p^{k-1}(1-p)^{m-k} \\ &= \frac{1-(1-p)^m}{p} \end{aligned}$$

The probability p can be approximated by $(1-ed)f_r$, where ed is the edge density of the graph. Thus an approximation for ρ_r is given by

$$\rho_r \approx \frac{1-(1-(1-ed)f_r)^{c_r d_{r-1}}}{(1-ed)f_r}$$

To obtain an approximation for $\overline{\rho}_r$, by similar arguments, it suffices an estimate for the number of the neighbours of either of two given adjacent vertices that get a final colour at round r. It can be estimated by $c_r d_{r-1} + c_r(1-td)d_{r-1}$, where td is defined by the following average with respect to the graph:

$$\text{average of } \frac{\text{number of triangles incident on a given edge}}{n-2}$$

Intuitively td provides the fraction of neighbours of a vertex v that are also neighbours of a neighbour u of v. Thus an approximation for $\overline{\rho}_r$ is

$$\overline{\rho}_r \approx \frac{1-(1-(1-ed)f_r)^{c_r d_{r-1}(2-td)}}{(1-ed)f_r}$$

Now we put together all the recurrences:

$$\begin{aligned}
f_r &= \tfrac{t_{r-1}}{(s_{r-1})^2} \\
c_r &= w(1-w\cdot f_r)^{d_{r-1}} \\
\rho_r &= \tfrac{1-(1-(1-ed)f_r)^{c_r d_{r-1}}}{(1-ed)f_r} \\
s_r &= s_{r-1}(1-f_r\rho_r) \\
\overline{\rho}_r &= \tfrac{1-(1-(1-ed)f_r)^{c_r d_{r-1}(2-td)}}{(1-ed)f_r} \\
t_r &= t_{r-1}(1-f_r\overline{\rho}_r) \\
d_r &= (1-c_r)d_{r-1}
\end{aligned}$$

with $s_0 = s$, $t_0 = s$, $d_0 =$ initial average degree, and $r = 1, 2, \ldots$

Figure 7 shows that these refined recurrences predict algorithm behaviour extremely accurately even when the edge density is very high. The refined recurrences provide us with an explanation for the "unreasonable" prediction power of the simple recurrences. Indeed, even though the palette size s_r reduces, the number of shared colours t_r reduces more so that the ratio $t_r/(s_r)^2$ keeps very close to $1/s$. This keeps true as far as the edge density is not too high. When the edge density is very high, t_r tends to be closer to s_r and the ratio $t_r/(s_r)^2$ keeps very close to $1/s_r$. As a consequence the probability of conflict gets higher and higher. Experimental data not reported here shows that the refined recurrences predict both the average size of palettes and the average number of shared colours quite accurately.

4 Experimental results: Brooks-Vizing colorings

In this section we experimentally analyze distributed algorithms for computing colorings with significantly fewer than Δ colors. We tested the local version only, i.e.

> *each vertex u is initially given a list of* $\deg(u)/s$ *colors, s being the shrinking factor.*

The algorithms tested were:

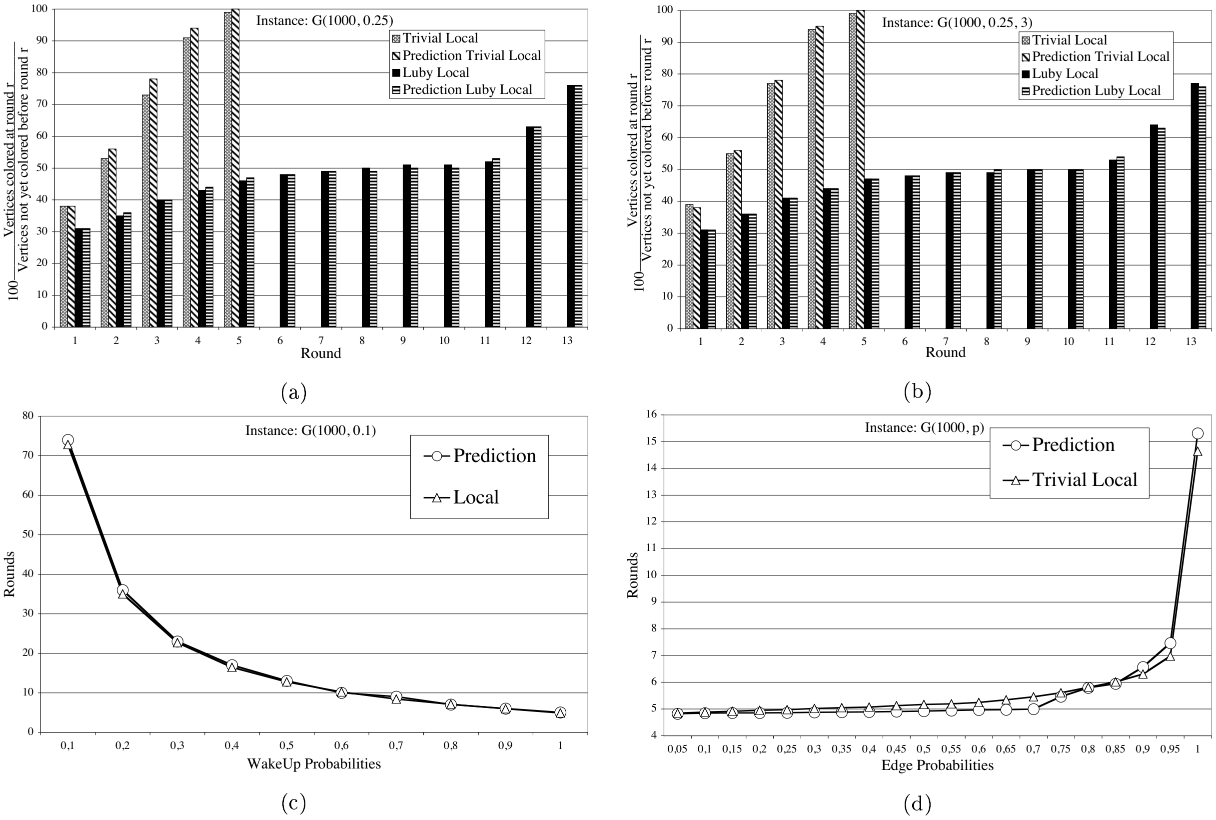


Figure 7: Refined prediction vs experiment: the refined recurrences of § 3.3 accurately predict both the probability that a vertex is coloured at a given round and the average number of rounds. As shown in (d) the prediction is very good even when the edge density is very high.

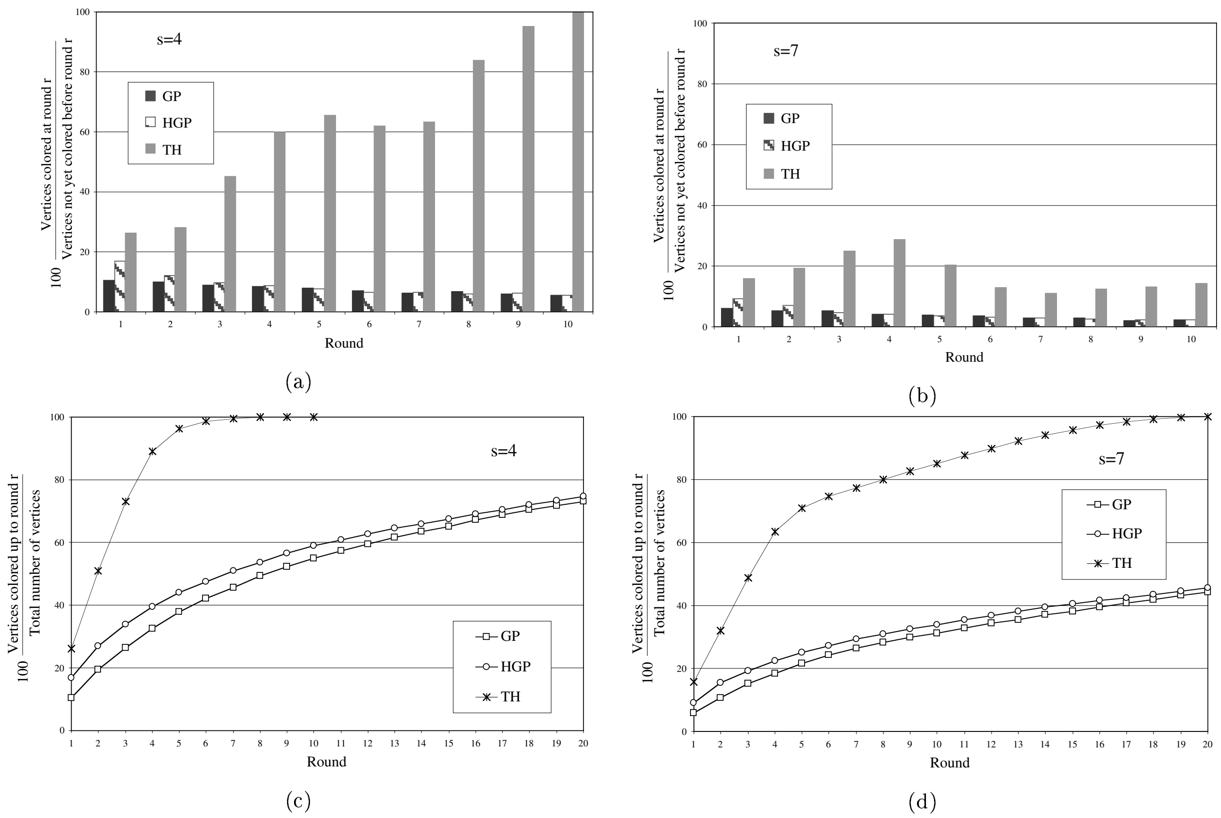


Figure 8: Brooks-Vizing colorings: behaviour of algorithms GP, HGP and TH on instances drawn from $G(500, 0.25)$. The left and right plots correspond to the shrinking factor $s = 4$ and $s = 7$, respectively. The y-axes of the upper plots show, for each round r, the percentage of uncolored vertices that get their final color at round r. The y-axes of the lower plots show, for each round r, the percentage of vertices that have a final color after round r.

- Algorithm `GP`, in which the wake up probability at round r is set to be

$$\frac{\textit{current palette size}}{\textit{current degree}}$$

  where the current degree is the number of neighbours yet to color.

- Algorithm `HGP` (hungarian GP), which is the same as algorithm GP with the introduction of the Hungarian Step for color conflict resolution;

- Algorithm `CH` (constantly hungarian), in which the wake-up probability is set to a constant value denoted as w. This algorithm always uses the Hungarian Step for color conflict resolution. Recall that algorithm `CH` with $w = 1$ is called algorithm TH (trivially hungarian).

Speed and parsimony. Our algorithms are very fast. A typical experimental outcome that illustrates this general conclusion is displayed in Figure 9. In the figure the running times of algorithms GP and TH are plotted (y-axis) against n, the number of vertices (x-axis). Instances were drawn from $G(n, 0.05)$ and $G(n, 0.05, 3)$, for growing n. Notice that while both algorithms

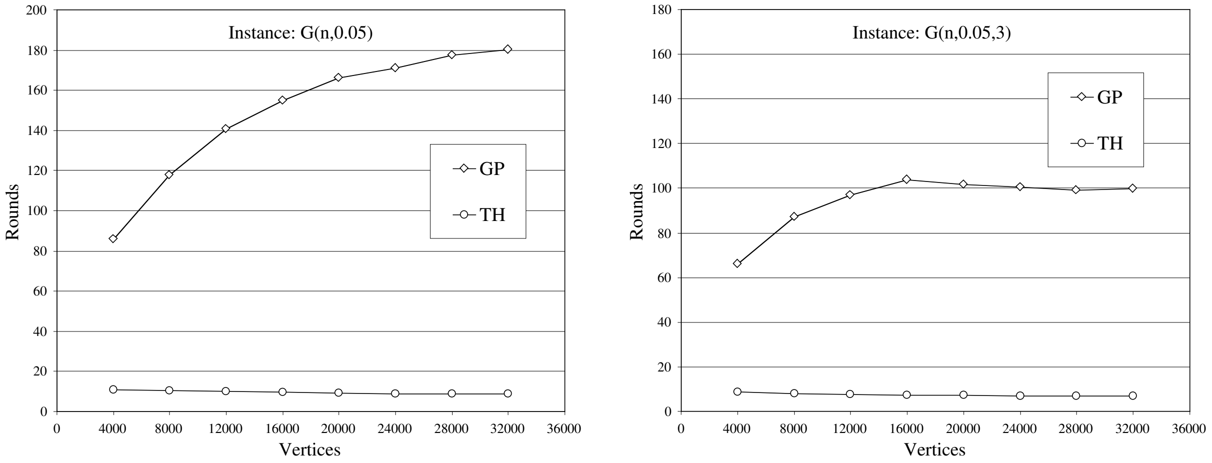


Figure 9: Brooks-Vizing colorings. Running times of algorithms GP and TH for instances drawn from $G(n, 0.05)$ and $G(n, 0.05, 3)$ for increasing values of n. The shrinking factor is $s = 5$.

are quite efficient, TH outperforms GP. In these tests the algorithms were computing colorings using roughly $\Delta/5$ colors. *In general all our tests show that TH ouperforms all other algorithms in speed, while using the same number of colors.*

We now address two questions. First, how much slower do our algorithms become as the initial palette size becomes smaller? And second, do we really save colors if we start with a smaller palette size? Recall that fresh new colors are introduced during the execution of the algorithms if and when a palette is depleted.

Figure 10a shows the running time of the algorithms with three shrinking factors $s = 2, 3, 7$. Each chart plots the number of rounds (y-axis) against the wake-up probability (x-axis). Corresponding to algorithms GP and HGP are straight lines since their wake-up probability is set by the algorithm. The curve corresponding to algorithm CH shows clearly that the best choice for the wake-up probability is 1, i.e. algorithm TH is the fastest. Clearly, as s increases (i.e. as the initial palette size decreases) the algorithms are bound to slow down. Figure 10a shows that while this deterioration is somewhat marked for algorithms GP and HGP, this is not so for algorithm CH (and in particular for TH). The same figure also shows that the hungarian step is not very useful for algorithm GP (the running times of HGP and GP are basically the same), while it is tremendously effective for CH and its simplest version TH. The charts in Figure 10a have been obtained by running the algorithms on random instances from $G(500, 0.25)$. The same trends emerge from Figure 10b, obtained by running the algorithms on 3-partite random instances from $G(500, 0.25, 3)$.

Figure 11 displays the results of a more systematic experiment in the same vein, reinforcing the same conclusions. The running time of algorithms GP, HGP and TH is plotted (y-axis) against the shrinking factor (x-axis). TH is confirmed to be much faster than GP and HGP. Note also that its running time increases less rapidly than that of the others. For a comparison, the figure also shows the running time of the $(\Delta+1)$-coloring algorithm Local HS, corresponding to algorithm TH with $s = 1$. Instances were drawn from $G(500, 0.25)$ and $G(500, 0.25, 4)$.

We also performed other experiments with different graph families and the results were basically the same. To summarize, *while all algorithms for Brooks-Vizing colorings proved to be quite fast, algorithm TH is clearly the fastest.* Its running time remains low even for large

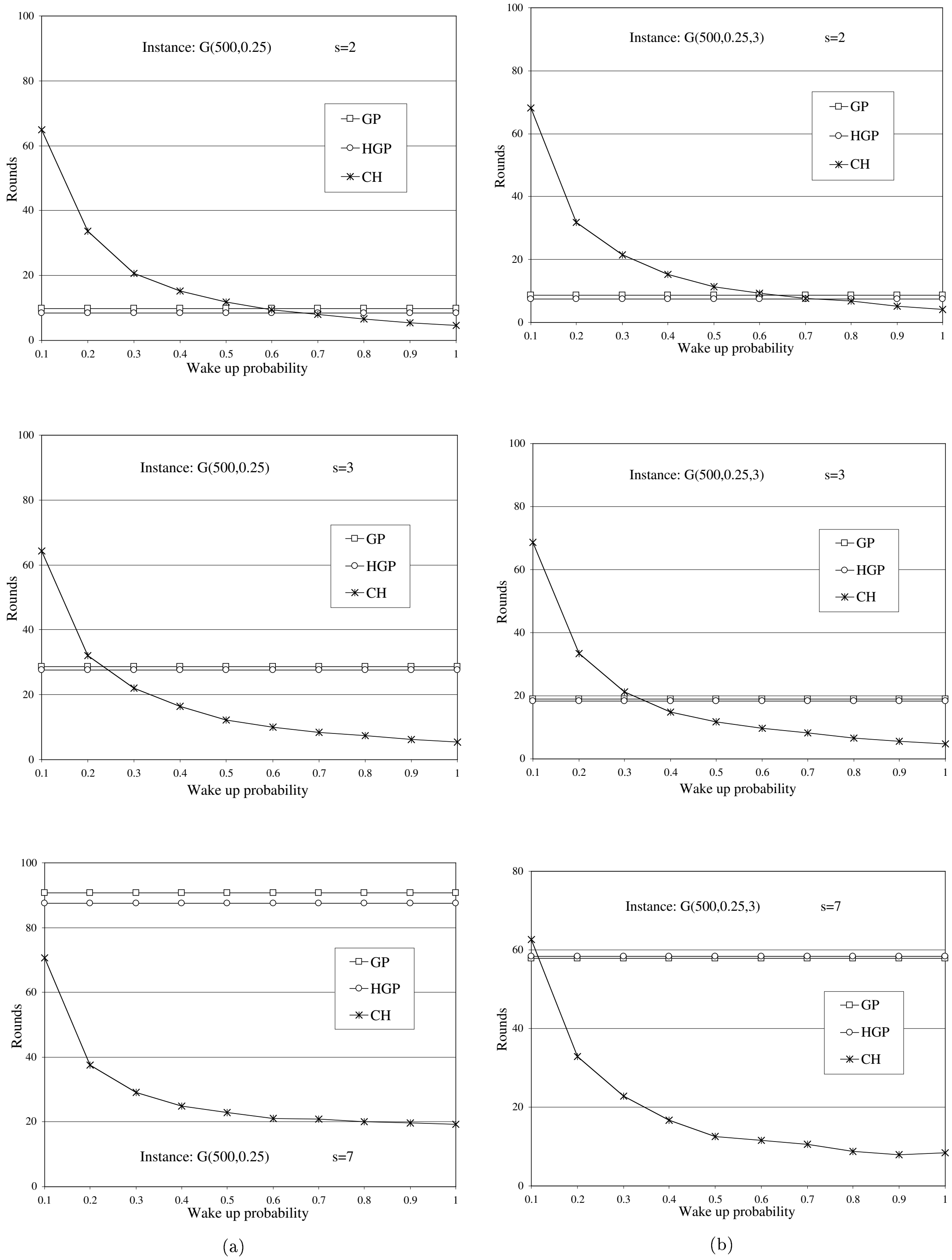


Figure 10: Brooks-Vizing colorings. Running times of algorithms GP, HGP and CH on instances drawn from $G(500, 0.25)$ (left column) and $G(500, 0.25, 3)$ (right column) for increasing values of the wake up probability. The three plots on each column correspond to the shrinking factor $s = 2, 3, 7$. Recall that algorithm TH is algorithm CH with the wake-up probability set to 1. The plots clearly show that this choice of the wake-up probability is the best.

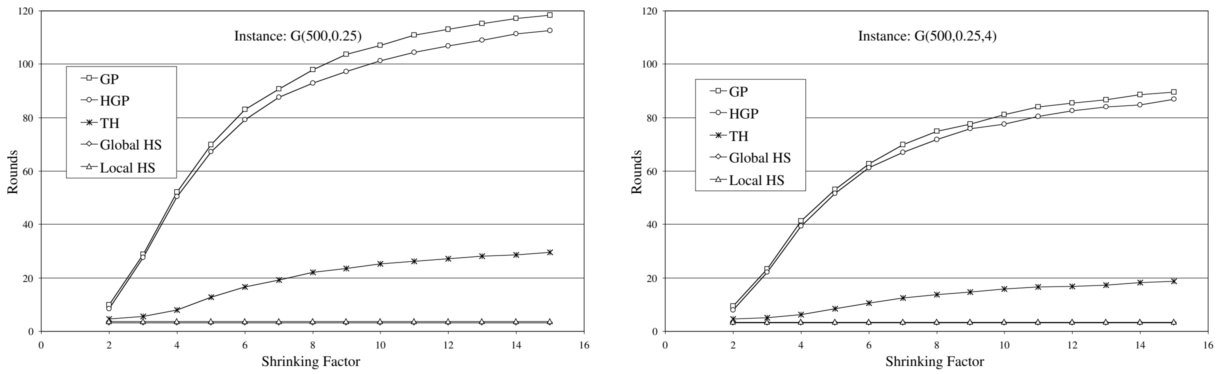


Figure 11: Brooks-Vizing colorings. Running time of algorithms GP, HGP and TH as the shrinking factor increases, i.e., as the initial palette size decreases. Algorithms Global HS and Local HS are $(\Delta + 1)$-coloring algorithms with the Hungarian step heuristic.

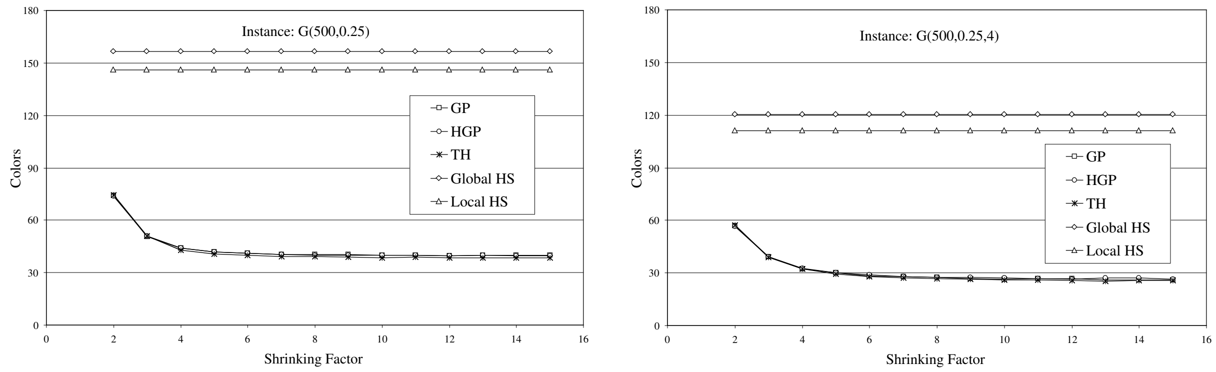


Figure 12: Brooks-Vizing colorings. Final number of colors used by algorithms GP, HGP and TH as the shrinking factor increases, i.e. as the initial palette size decreases. For a comparison, the number of colors used by $(\Delta + 1)$-algorithms with global and local palette setting is also displayed (straight lines).

values of the shrinking factor.

Let us now turn to the second question: Does it pay off, in terms of number of colors, to start with a small palette size (large s)? The conclusion that can be drawn from our tests is rather robust. It is exemplified in Figure 12, showing tests for instances drawn from $G(500, 0.25)$ and $G(500, 0.25, 4)$. It does not pay off to start with values of s greater than 6, for the final number of colours stabilizes after that value (while the running time increases). Contrary to what we observed for the running time, in terms of colors the performance of algorithms GP, HGP and TH is the same. In particular, Figure 12 shows that great savings are attained by these algorithms with respect to algorithms for $(\Delta+1)$-colorings. The best colorings computed used between $\Delta/4$ and $\Delta/5$, a significant saving with respect to $(\Delta + 1)$-colorings. Our tests show that such colorings can be computed very quickly, especially by algorithm TH.

The same figure suggests that the color performance tends to an asymptotic value (as s increases). We will see later if this is really the case.

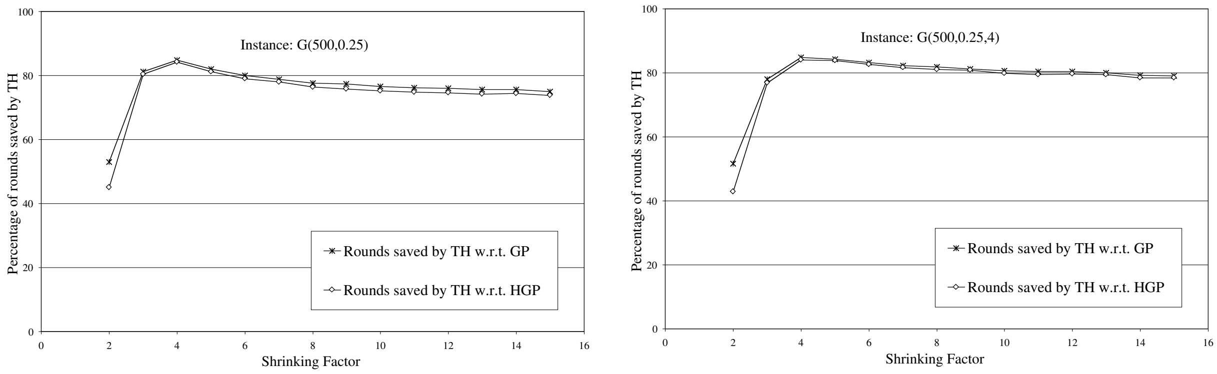


Figure 13: Brooks-Vizing colorings. The figure plots the ratios (*#rounds of GP-#rounds of TH*)/(*#rounds of GP*) and (*#rounds of HGP-#rounds of TH*)/(*#rounds of HGP*) against the shrinking factor. The figure shows that significant savings are attained by algorithm TH.

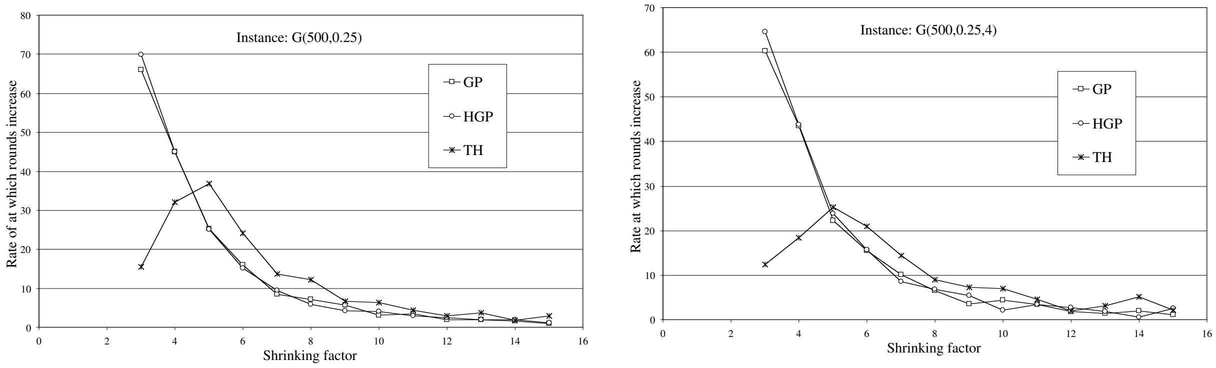


Figure 14: Brooks-Vizing colorings. Rate at which the running time of algorithms GP, HGP, and TH increases as the shrinking factor grows: $r(s) = k$ means that, when the shrinking factor s is used instead of $s-1$, the number of rounds increases by $k\%$. Instances were drawn from $G(500, 0.25)$ and $G(500, 0.25, 4)$.

More on speed and shrinking factor. We can now quantify more precisely the time savings that can be obtained by algorithm TH with respect to GP and HGP. TH was chosen rather than CH because it systematically performed better than CH for different values of the wake-up probability.

Figure 13 shows that algorithm `TH` saves around $70\% - 80\%$ of the number of rounds of `GP` (the percentage is smaller only for $s = 2$). Since the same relative performance of `GP` and `TH` has been observed in all our experiments, we can definitely conclude that algorithm `TH` is the algorithm of choice for computing Brooks-Vizing colorings in a distributed setting.

In order to find heuristic recurrences that could explain the splendid performance of algorithm `TH` we plotted the percentage of vertices that succesfully colour themselves at each round (see Figure 8). sAs iIt is apparent the behaviour of algorithm `TH` is very irregular. We have not been able to find a satisfactory explanation for this or for the difference in performance, whether heuristic or rigorous. We leave this as an interesting and challenging open set of questions.

A very rough explanation to account for this marked difference in speed for the very first rounds is the following. In the initial rounds of algorithm GP a vertex wakes up with probability $\sim 1/s$, where s is the shrinking factor. A vertex that wakes up has on average a number of neighbors vying for coloring equal to the size of its list of available colors. Therefore a vertex colors itself with probability $\sim 1/(se)$. In contrast, algorithm TH first partitions the vertex set into a certain number of classes, say k. The average degree of a conflict graph G_c– the graph induced by all vertices that picked tentative color c– is s. Recall that an application of the HS step in a graph with x vertices and average degree d yields an independent set of size $\sim x/d$. Therefore on average n/s vertices will get a final color after one step of algorithm TH, as opposed to $\sim n/(se)$ with algorithm GP. As remarked, this rough analysis is plausible only for the initial rounds.

Finally, Figure 14 plots $r(s)$, the rate at which the number of rounds increases against the shrinking factor s. The chart should be interpreted as follows: $r(s) = k$ means that when shrinking factor s is used instead of $s-1$ the number of rounds increases by $k\%$. All the Brooks-Vizing algorithms exhibit a good behaviour w.r.t. this measure.

What is the best shrinking factor? Figure 12 seems to suggest that the shrinking factor tends to an asymptotic value, i.e., values of $s > 6$ do not yield better colorings. This is not always the case, however. We investigated this issue on random k-partite instances and observed a somewhat surprising phenomenon. We fixed a distribution $G(n,p)$ (i.e., we fixed n and p) and chose distributions of k-partite random graphs $G(n, p_k, k)$ to satisfy the following. Graphs from $G(n, p_k, k)$ all had the same expected density, and the same minimum, average and maximum degree of graphs from $G(n,p)$. This can be done by setting $p_k = k/(k-1)p$. We then ran algorithm TH on these instances with different shrinking factors. The outcome is reported in Figure 15. The figure shows that for graphs from $G(n, p_k, k)$ it does pays off to consider bigger values of s.

At first this behavior looks surprising, since it seem to depend on the chromatic number of the graphs– a global property. The chromatic number of graphs from $G(n, p_k, k)$ is almost surely k. Note that TH finds better colorings for smaller values of k. However, by considering more carefully the local properties of these instances, we are able to put forward a plausible explanation based on local properties only. Consider for each vertex v the subgraph induced by its neighbors and v itself. Call it the *local subgraph* of v. Clearly when $k = 2$ local subgraphs have no triangles. But as soon as $k \geq 3$ they contain many. The number of triangles grows with k up to an asymptotic maximum and its progress is similar to that reported in Figure 15 for the number of colors when $s = 10, 20$. Now, when the shrinking factor is quite large and the local subgraphs have few triangles then the conflict subgraphs are big but sparse. This allows the selection of big independent sets, and thus many vertices get the same color.

Experiments with different graph densities. Figure 16 is concerned with reduced palette algorithms on random instances $G(1000, p)$ and $G(1000, p, 3)$ for $0.05 \leq p \leq 0.45$ and is the companion to Figure 5. The first two charts show the number of rounds of GP for different shrinking factors on $G(1000, p)$ and on $G(1000, p, 3)$. The next two charts do the same for TH. Since GP and TH use the same number of colors, the last two charts report only on the coloring of GP. By comparing the number of rounds performed by the two algorithms on $G(1000, p)$, it is evident that in both cases the number of rounds increases linearly with the density and curves related to bigger shrinking factors are steeper. The range of the values on the y-axes in the two charts is the same, making it easier to grasp the advantage of TH over GP: though not reported in the chart, we point out that `GP s=8` requires 293 rounds for $p = 0.45$, instead of 56 rounds used by `TH s=8`. Although the number of rounds for $G(1000, p, 3)$ of the two algorithms differ markedly, both curves increase up to a maximum and then decrease. This is more noticeable

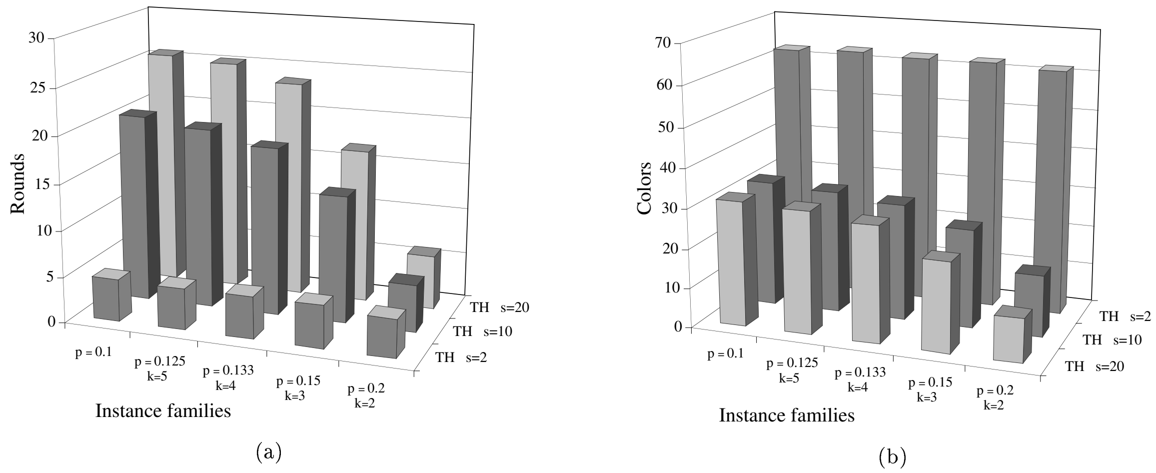


Figure 15: Brooks-Vizing colorings. Algorithm TH for different shrinking factors on k-partite instances with the same density and degrees: the smaller k, the easier the instance.

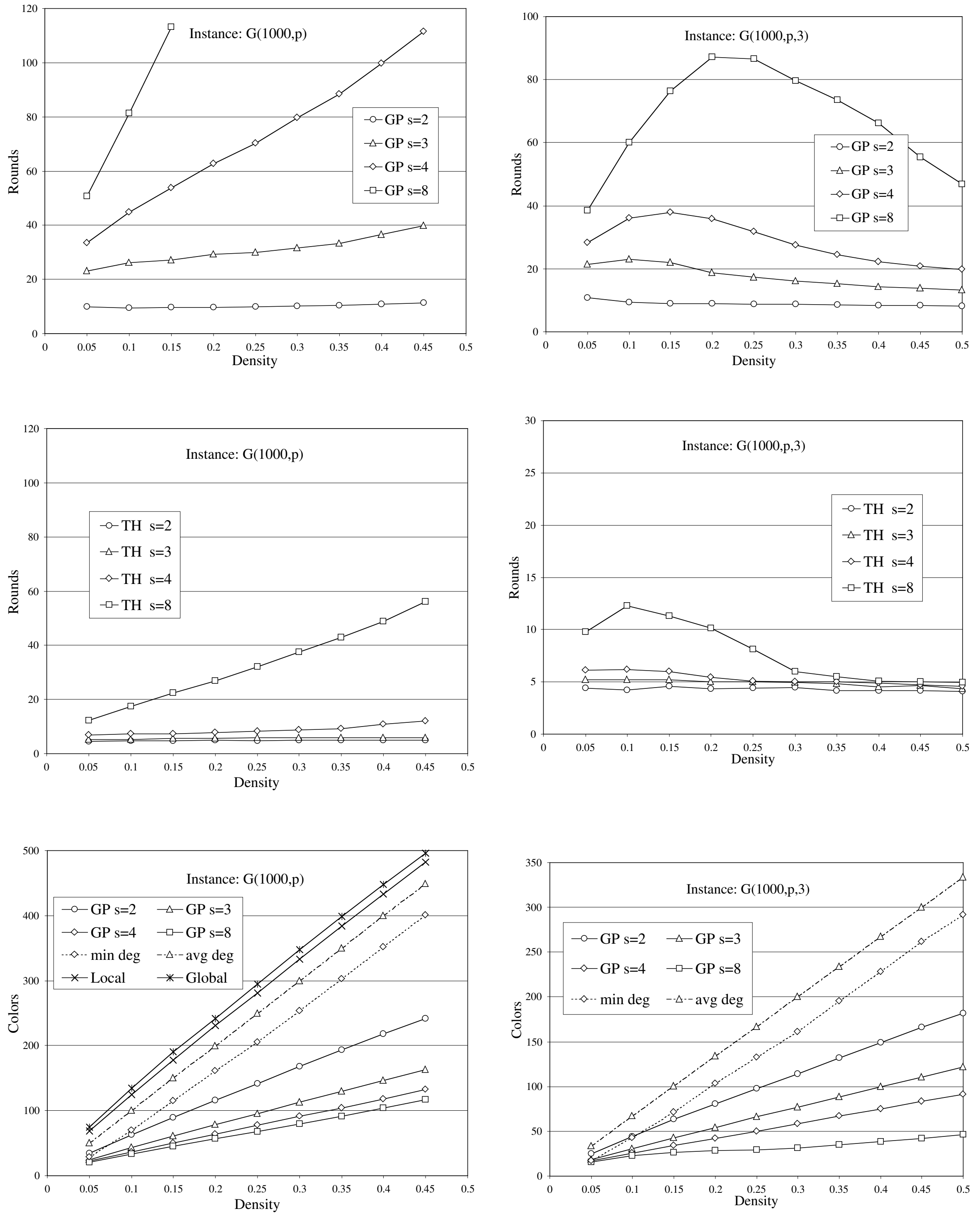
for bigger shrinking factors. Again, a possible explanation for this phenomenon involves the structure of the local subgraphs. When compared with instances from $G(1000, p)$ instances with the same average degree, the local subgraphs here contain much less triangles. This might explain the trends exhibited by the number of colors. As the density (i.e., the average degree) grows the size of the initial palette grows too and so does the number of colors used. But in the case of $G(1000, p, 3)$ the ratio of the palette size over the number of colors grows much faster than the same ratio for $G(1000, p)$ instances. For example when $s = 8$ the ratio goes from about 0.2 for $p = 0.05$ to 0.56 for $p = 0.2$ and to 0.83 for $p = 0.45$. While for $G(1000, p)$ instances the ratio is always less than 0.5. The relevance of this ratio for the number of rounds should be evident: it is directed connected to the percentage of vertices that run out of colors which, in turn, is the most important factor to determine the increase in the number of rounds.

5 Results with DIMACS instances

In this section we briefly discuss the performance of algorithms HGP and TH on a subset of the instances used in the second DIMACS implementation challenge [13], on other test sets available from the DIMACS FTP site, and on graphs with decreasing degree variance generated using Culberson's clique-driven generator. We consider both colors and rounds and, with respect to colors, in the case of DIMACS benchmarks we compare HGP and TH against two well known algorithms: Iterated Greedy and Hybrid [13]. These algorithms are known to produce very good colorings, though can require a considerable amount of time (minutes or even hours). Our distributed algorithms compare rather well.

We ran algorithms HGP and TH at least 10 times per instance and we averaged the values of colors and rounds over the runs. The value of the shrinking factor used on each instance is reported between parentheses (results for different shrinking factors are available with the experimental package).

The table in Figure 17 confirms that TH is considerably faster than HGP and shows that TH usually obtains slightly better colorings. It also reports the standard deviation of the number of rounds. Note that the standard deviation of TH is always much smaller than that of HGP.



(a) (b)

Figure 16: Brooks-Vizing colorings. Behavior of algorithms GP and TH on instances with increasing edge probability drawn from $G(1000, p)$ and $G(1000, p, 3)$.

Instance	HGP Rounds (Std. Dev.)	TH Rounds (Std. Dev.)	HGP Colors (s)	TH Colors (s)	IG Colors	HY Colors
C2000.5.col	661.2 (33.55)	122.2 (1.47)	229.6 (10)	226.0 (10)	190	DNR
flat300_20_0.col	119.2 (6.03)	32.6 (1.02)	49.1 (10)	47.7 (10)	20.2	20
flat300_26_0.col	123.6 (10.2)	32.8 (1.4)	50 (10)	48.3 (10)	37.1	32.4
flat300_28_0.col	125.6 (8.55)	33.1 (0.94)	49.8 (10)	48.2 (10)	37	33
flat1000_50_0.col	355.3 (13.91)	76.7 (1)	131 (10)	128.8 (10)	65.6	97
flat1000_60_0.col	365.4 (12.06)	76.9 (1.22)	130.5 (10)	128.9 (10)	102.5	97.8
flat1000_76_0.col	362.4 (3.77)	76.6 (1.02)	130.6 (10)	128.6 (10)	103.6	99
latin_square_10.col	514.7 (25.52)	91.9 (2.11)	159.6 (10)	159.9 (10)	106.7	109.25
le450_15a.col	43.8 (4.19)	13.9 (0.7)	23.5 (10)	21.8 (10)	17.9	15
le450_15b.col	45.1 (2.98)	14.8 (0.98)	23.4 (10)	22.3 (10)	17.9	15
le450_15c.col	69.7 (3.52)	18.8 (0.98)	33 (10)	30.9 (10)	25.6	16.6
le450_15d.col	70.8 (5.81)	19.0 (0.77)	33.1 (10)	30.9 (10)	25.8	16.8
mulsol.i.1.col	72.5 (5.64)	38.7 (1.55)	49.3 (10)	49.1 (10)	49	49
school1.col	104.1 (6.02)	26.8 (1.07)	46 (15)	43.4 (15)	14	14
school1_nsh.col	85.1 (5.41)	20.9 (2.3)	40.4 (10)	39.8 (10)	14.1	14
inithx.i.1.col	50.2 (6.55)	17.4 (3.92)	62.7 (10)	56.2 (10)		
inithx.i.2.col	28 (2.0)	4.9 (0.94)	50.9 (10)	47.2 (10)		
inithx.i.3.col	30.8 (5.74)	4.9 (0.7)	52 (10)	46.6 (10)		
fpsol2.i.1.col	89.5 (7.48)	42.8 (0.98)	65.5 (10)	65.1 (10)		
fpsol2.i.2.col	34.6 (3.26)	6.2 (0.87)	41.9 (10)	35.2 (10)		
fpsol2.i.3.col	33.6 (5.42)	6.2 (0.74)	40.3 (10)	34.2 (10)		
degree_variance.1	70.4 (7.49)	26.6 (4.22)	46.8 (20)	43.4 (20)		
degree_variance.2	66.2 (7.44)	21.6 (3.13)	43.6 (20)	41.0(20)		
degree_variance.3	59.6 (6.46)	20.6 (3.2)	41.8 (20)	40.0 (20)		
degree_variance.4	61.8 (9.51)	19.8 (2.31)	42.2 (20)	40.6 (20)		
degree_variance.5	58.8 (6.4)	25.4 (3.26)	40.6 (20)	40.0 (20)		
degree_variance.6	56.6 (8.4)	27.0 (2.6)	40.6 (20)	40.0 (20)		
degree_variance.7	55.6 (7.73)	22.6 (5.38)	40.0 (15)	40.0 (15)		
degree_variance.8	44.2 (9.8)	17.2 (5.52)	39.0 (15)	39.0 (15)		

Figure 17: Results on DIMACS instances.

References

[1] J.R. Allwright, R. Bordawekar, P.D. Coddington, K. Dincer, and C.L. Martin. A comparison of parallel graph coloring algorithms. Technical Report SCCS-666, Northeast Parallel Architecture Center, Syracuse University, 1995.

[2] S. Basagni, Finding a maximal weighted independent set in wireless networks, *Telecommunication Systems*, 18:1,2, 155-168, 2001.

[3] B. Bollobás. Chromatic number, girth, and maximal degree. *Discrete Mathematics* 24:311–314, 1978.

[4] D. Brélaz. New methods to color vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.

[5] S. Chaudhuri and D. Dubhashi. Probabilistic recurrence relations revisited. *Theoretical Computer Science*, to appear.

[6] T.F. Coleman and J.J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM Journal on Numerical Analysis*, 20:187–209, 1983.

[7] J.C. Culberson. `http://web.cs.ualberta.ca/~joe/Coloring/`.

[8] FTP site of DIMACS implementation challenges. `ftp://dimacs.rutgers.edu/pub/challenge/`.

[9] A.H. Gebremedhin and F. Manne. Scalable parallel graph coloring algorithms. *Concurrency: Practice and Experience*, 12:1131–1146, 2000.

[10] D.A. Grable and A. Panconesi. Fast distributed algorithms for Brooks-Vizing colourings. *Journal of Algorithms*, 37:85–120, 2000. Special issue for SODA 98, the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms.

[11] D.A. Grable and A. Panconesi. Nearly optimal distributed edge colouring in $O(\log\log n)$ rounds. *Random Structures and Algorithms*, 10(3):385–405, 1997. Preliminary version in Proceedings of the Eight Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 97).

[12] G.R. Grimmet and C.J.H. McDiarmid. On colouring random graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 77:313–324, 1975.

[13] D.S. Johnson and M.A. Trick. *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

[14] A.R. Johansson. Asymptotic choice number for triangle-free graphs. Preprint, DIMACS, September 30, 1996.

[15] Ö. Johansson. Simple distributed $\Delta+1$-coloring of graphs. *Information Processing Letters* 70:229–232, 1999.

[16] M.T. Jones and P.E. Plassmann. A parallel graph coloring heuristics. *SIAM Journal on Scientific Computing*, 14(3):654–669, 1993.

[17] R.M. Karp. Probabilistic recurrence relations. In proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC 91), 190–197, 1991.

[18] J.H.Kim, On Brooks' theorem for sparse graphs, *Combinatorics Probability and Computing*, 4 (1995) 97-132.

[19] J. Körner. Private communication.

[20] G. Lewandowski and A. Condon. Experiments with parallel graph coloring heuristics and applications of graph coloring. *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

[21] M. Luby. Removing randomness in parallel without processor penalty. *Journal of Computer and System Sciences*, 47(2):250–286, 1993.

[22] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986.

[23] M.V. Marathe, A. Panconesi, and L.D. Risinger jr. An experimental study of a simple, distributed edge coloring algorithm. In proceedings of the 12th ACM Symposium on Parallel Algorithms and Architectures (SPAA'00), 166–175, 2000.

[24] P. Sinha, R. Sivakumar, and V. Bharghavan, Enhancing ad hoc routing with dynamical virtual infrastructures, Proceedings of IEEE Infocom 2001, 1-10.

[25] P. Sinha, R. Sivakumar, and V. Bharghavan, CEDAR: a core-extraction distributed ad-hoc routing algorithms, *IEEE Journal on Selected Areas in Communication*, 17, 8, 1454-1458, August 1999.