



ELSEVIER

Information Processing Letters 70 (1999) 229–232

Information
Processing
Letters

www.elsevier.com/locate/ipl

Simple distributed $\Delta + 1$ -coloring of graphs

Öjvind Johansson¹

Department of Numerical Analysis and Computing Science, Royal Institute of Technology, SE-100 44 Stockholm, Sweden

Received 9 December 1998; received in revised form 19 April 1999

Communicated by L.A. Hemaspaandra

Abstract

A very natural randomized algorithm for distributed vertex coloring of graphs is analyzed. Under the assumption that the random choices of processors are mutually independent, the execution time will be $O(\log n)$ rounds almost always. A small modification of the algorithm is also proposed. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Distributed computing; Randomized algorithms; Vertex coloring of graphs

1. Introduction

To *vertex color* a graph means to give each vertex a color in such a way that no two adjacent vertices get the same color. If at most k colors are used, the result is called a k -*coloring*. Vertex coloring is useful when edges between vertices signify interference between the objects of a system. A coloring then partitions the system into sets which are internally free from interference. This partition may be the desired end result, but it could also be of intermediate use to an algorithm constructing the solution to another problem, as described in [5].

In this paper we discuss vertex coloring in a *distributed network*. Thus, we have a set V of processors and a graph $G = (V, E)$. Two processors are joined by a bidirectional communication link if they are adjacent in G . There is no shared memory. Each processor knows its own links, and it has a unique identification number which it knows too, but apart from that, it has no prior knowledge of the structure of G . We

now want these processors to agree upon a coloring of G . We will assume that the system is synchronized in phased *cycles*: In a first phase, all processors compute; in a second, they send up to one message per neighbor. In a third and final phase, processors receive these messages. Thereafter, another cycle begins. The number of such cycles needed will be our measure of efficiency.

More precisely, we shall consider the computation of $\Delta + 1$ -colorings, where Δ is the highest vertex degree in the graph. That such colorings must exist is evident from the following greedy sequential algorithm: For each vertex v , assign to v the first color not already assigned to any neighbor. (We assume that colors are ordered in a sequence.) But sequential solutions are not fast enough in the distributed world. We would like processors to work in parallel. And such an algorithm lies not so far away. The following randomized one is mentioned in [1] as “*the trivial algorithm*”.

The vertex coloring algorithm. As long as it is uncolored, each vertex u in the graph keeps a *palette*: a list (set) of colors not already assigned to any neigh-

¹ Email: ojvind@nada.kth.se.

bor. This list, $P(u)$, is initialized to the first $d(u) + 1$ colors in S , where $d(u)$ is the degree of u , and where S is a predefined color sequence. The algorithm then proceeds in *rounds*. In a round (which can be implemented with two cycles as above), each uncolored vertex u randomly (uniformly) chooses a tentative color from its palette. It informs its neighbors and checks if any of them chose the same color. If not, u keeps this color — we say u is *colored* — and again it informs its neighbors. If, on the other hand, some neighbor of u did choose the same color, u does not keep it. (Nor does the neighbor.) At the beginning of the next round, u instead updates its palette by removing colors that were taken permanently by neighbors.

A similar algorithm has been given by Luby [4]. His algorithm contains an extra feature though; in each round, each uncolored vertex first “tosses a coin” to see if it should at all try to take a color during that round. Processors need then only be pairwise independent in their random choices in order to guarantee that, at each round, any uncolored vertex will get colored with probability at least $1/4$. By relying on pairwise independence only, Luby was able to derandomize his algorithm into an efficient deterministic algorithm for the PRAM model of computation. In the next section, we shall see that under the assumption of mutual independence, the “trivial algorithm” also will, in each round, color any uncolored vertex with probability $1/4$ or more. As a result, the expected running time will be $O(\log n)$ in the distributed model.

Interestingly, if we want to find $\Delta + 1$ -colorings with a *deterministic* distributed algorithm, we may have to spend some more time. The best result so far is stated in [5]: With the technique called network decomposition, a $\Delta + 1$ -coloring can be found in $n^{O(\varepsilon(n))}$ time, where $\varepsilon(n) = 1/\sqrt{\log n}$.

2. Analysis

We shall now analyze the vertex coloring algorithm defined in Section 1. As already stated, we will assume that random choices are mutually independent.

Theorem 1. *In each round of the algorithm, any uncolored vertex will be colored with probability at least $1/4$.*

Proof. We study the algorithm at the beginning of some arbitrary round. (When we refer to palettes, we imply that these have been updated with respect to the previous round.) Let u be a vertex which is still uncolored, and let C_u be the event that u gets colored in the present round. (In the following, we will simply say “ u gets colored”.) The probability for this is a function only of the palettes of u and its uncolored neighbors, the latter vertices which we denote by $N(u)$. To find a lower bound for $\Pr[C_u]$, we shall alter, if needed, the palettes of vertices in $N(u)$, such that $\Pr[C_u]$ will be easier to compute, but without increasing it.

First, let us look at the situation prior to such alterations. The following trivial fact is of course crucial for the algorithm to work.

Proposition 2. *For any uncolored vertex v , $|P(v)| \geq |N(v)| + 1$.*

Proof. By definition, the inequality holds at the outset. From then on, a color may be removed from the palette of v only if it is taken by a neighbor. \square

To further estimate palette sizes, we look at neighborhoods. If $N(u)$ is empty (this can only occur at the first round), then surely u will be colored immediately. We can assume therefore that u has at least one uncolored neighbor. As a consequence, we have:

Proposition 3. *Vertex u and its uncolored neighbors all have palettes of size at least 2.*

Next, it is time to formulate the probability that u will get colored. For a color c and a vertex v , let $W_{c,v}$ be the event that v chooses c . And for a color c and a vertex set V , let $\overline{W}_{c,V}$ be the event that none of the vertices in V chooses c . We get:

$$\begin{aligned} \Pr[C_u] &= \sum_{c \in P(u)} \Pr[W_{c,u} \wedge \overline{W}_{c,N(u)}] \\ &= \sum_{c \in P(u)} \Pr[W_{c,u}] \cdot \Pr[\overline{W}_{c,N(u)}] \\ &= \frac{1}{|P(u)|} \sum_{c \in P(u)} \Pr[\overline{W}_{c,N(u)}]. \end{aligned} \quad (1)$$

For an uncolored neighbor v of u , we can then describe the impact of the palette of v on $\Pr[C_u]$. For each color $c \in P(u)$, we have

$$\begin{aligned} \Pr[\overline{W}_{c,N(u)}] &= \Pr[\overline{W}_{c,N(u)\setminus v} \wedge \overline{W}_{c,v}] \\ &= \Pr[\overline{W}_{c,N(u)\setminus v}] \cdot \Pr[\overline{W}_{c,v}] \\ &= \Pr[\overline{W}_{c,N(u)\setminus v}] \cdot (1 - \Pr[W_{c,v}]), \end{aligned}$$

so Eq. (1) gives

$$\Pr[C_u] = \frac{1}{|P(u)|} \sum_{c \in P(u)} \Pr[\overline{W}_{c,N(u)\setminus v}] \cdot (1 - \Pr[W_{c,v}]). \quad (2)$$

We are now ready to show how palettes can be altered. For the following three propositions, we will refer to Eq. (2) when we modify the palette of an uncolored neighbor v of u . It should be noticed then that $\Pr[\overline{W}_{c,N(u)\setminus v}]$ does not depend on this palette, whereas $\Pr[W_{c,v}]$ equals $1/|P(v)|$ if $c \in P(v)$, and zero otherwise.

Proposition 4. *Let $v \in N(u)$. To find a lower bound for $\Pr[C_u]$, we may assume that $P(v) \subseteq P(u)$.*

Proof. Let us call colors in $P(u)$ “valid” and the remaining colors “invalid”. Suppose we have a situation in which the palette of v contains some invalid colors. Let us then remove these successively. At each removal, add a valid color instead, unless $P(v)$ already contains all valid colors. During this process, $\Pr[W_{c,v}]$ will not decrease for any $c \in P(u)$, so $\Pr[C_u]$ will not increase, as Eq. (2) shows. Because of the addition of valid colors, Proposition 3 will remain true. \square

Proposition 5. *Let $v \in N(u)$, and let $P(v) \subseteq P(u)$. Then we may also assume that $|P(v)| = 2$.*

Proof. By Proposition 3, $|P(v)| \geq 2$. Suppose that $|P(v)| > 2$. Then let c' be any color $c \in P(v)$ which minimizes $\Pr[\overline{W}_{c,N(u)\setminus v}]$, and remove c' from $P(v)$. This will make

$$\sum_{c \in P(u)} \Pr[\overline{W}_{c,N(u)\setminus v}] \cdot \Pr[W_{c,v}]$$

at least as large as before. (We have simply transferred probability from $W_{c',v}$ to events $W_{c,v}$ that yield no less in this sum.) By Eq. (2), $\Pr[C_u]$ will not increase. Repeat the process until $|P(v)| = 2$. \square

vertex	palette				
w	α		γ		
x			γ	δ	
y		β	γ		
z	α	β			
u	α	β	γ	δ	ε

Fig. 1. An example illustrating the proof of Proposition 6. Vertices w , x , y , and z are the uncolored neighbors of u . Three of their palettes contain γ . We can avoid this without increasing $\Pr[C_u]$. For example, let $P(w)$ be $\{\alpha, \varepsilon\}$ instead.

Proposition 6. *Let all uncolored neighbors of u have palettes of size 2. We may then assume that for each color $c \in P(u)$, there are at most two uncolored neighbors of u whose palettes contain c . This condition is equivalent to $\Pr[\overline{W}_{c,N(u)}] \geq 1/4$.*

Proof. We will use the term $N(u)$ -palette for the palette of a vertex in $N(u)$. Since all of these are of size 2, it follows from Proposition 2 applied to u that there are at most $2|P(u)| - 2$ color occurrences in all the $N(u)$ -palettes. Suppose now that a color $c' \in P(u)$ is contained in more than two $N(u)$ -palettes. Then take one of these and let v be its “owner”. By the pigeon-hole principle, there are at least two colors in $P(u)$ which are contained in at most one $N(u)$ -palette each. Among these two or more colors, we can find $c'' \notin P(v)$. (See Fig. 1 for an example assisting the following argument. In the example, $c' = \gamma$. With $v = w$, c'' can be either δ or ε .) Note that $\Pr[\overline{W}_{c'',N(u)\setminus v}] \geq 1/2$, since either c'' is in no $N(u)$ -palette, or it is in one, which is of size 2. Note also that $\Pr[\overline{W}_{c',N(u)\setminus v}] \leq (1/2)^2$, since c' is contained in at least two $N(u)$ -palettes other than that of v , each of size 2. Therefore, by removing c' from $P(v)$ and including c'' instead (which means that we transfer probability from $W_{c',v}$ to $W_{c'',v}$), $\Pr[C_u]$ will be reduced, as Eq. (2) shows. Repeat the above process until c' is contained in at most two $N(u)$ -palettes. Since these are both of size 2, we have $\Pr[\overline{W}_{c',N(u)}] \geq (1/2)^2$. \square

We now conclude the proof of Theorem 1. We have seen that to find a lower bound for $\Pr[C_u]$, we only have to consider cases satisfying the inequality in Proposition 6. Using Eq. (1), we get $\Pr[C_u] \geq 1/4$. \square

Corollary 7. *The algorithm runs in $O(\log n)$ rounds almost always. More precisely, for the running time $T(n)$ on a graph with n vertices, we have, for any positive integer t :*

$$\Pr[T(n) \geq \lfloor \log_{4/3} n \rfloor + t + 1] \leq \left(\frac{3}{4}\right)^{t-1}.$$

Proof. This follows from Theorem 1 above with the help of Theorem 1 in [2] (Theorem 1.1 in [3]). \square

3. Suggestion

It is clear from the proof that Theorem 1 would hold even if the palette of a vertex u were initialized to any $d(u) + 1$ or more colors among the $\Delta + 1$ first colors in S (see the definition of the algorithm). Using larger palettes could mean a shorter running time, since many vertex pairs would then have a smaller probability of a choice clash (although for some palette pairs, an additional color in their intersection would increase this probability). However, if u does not know Δ , but only its own degree, it cannot start with a larger palette than defined. To try to exploit some more colors anyway, we can extend the algorithm as follows:

Modification. In the first round, u initializes a new variable $\hat{d}(u)$ to $d(u)$ and sends this value to its neighbors. From then on, let $\hat{d}(u)$ be the highest

degree that u has heard of in previous rounds. Each round that $\hat{d}(u)$ increases, u passes the new value on, and before u makes its random choice of color, it extends its palette, as if $\hat{d}(u) + 1$ had been the initial palette size.

Acknowledgements

I thank Alessandro Panconesi, who introduced me to the problem, and who has inspired and assisted me in writing this paper. I also thank Stefan Arnborg and the two referees for their comments.

References

- [1] D.A. Grable, A. Panconesi, Fast distributed algorithms for Brooks–Vizing colourings, in: Proceedings of the Ninth Annual ACM–SIAM Symposium on Discrete Algorithms, 1998, pp. 473–480.
- [2] R.M. Karp, Probabilistic recurrence relations, in: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, 1991, pp. 190–197.
- [3] R.M. Karp, Probabilistic recurrence relations, J. ACM 41 (1994) 1136–1150.
- [4] M. Luby, Removing randomness in parallel computation without a processor penalty, J. Computer System Sci. 47 (1993) 250–286.
- [5] A. Panconesi, A. Srinivasan, On the complexity of distributed network decomposition, J. Algorithms 20 (1996) 356–374.