COMBINATORICA
Akadémiai Kiadó – Springer-Verlag

COMBINATORICA **15** (2) (1995) 255–280

THE LOCAL NATURE OF Δ-COLORING AND ITS ALGORITHMIC APPLICATIONS

ALESSANDRO PANCONESI and ARAVIND SRINIVASAN

Received October 6, 1992

Given a connected graph $G=(V,E)$ with $|V|=n$ and maximum degree Δ such that G is neither a complete graph nor an odd cycle, Brooks' theorem states that G can be colored with Δ colors. We generalize this as follows: let $G-v$ be Δ-colored; then, v can be colored by considering the vertices in an $O(\log_\Delta n)$ radius around v and by recoloring an $O(\log_\Delta n)$ length "augmenting path" inside it. Using this, we show that Δ-coloring G is reducible in $O(\log^3 n/\log\Delta)$ time to $(\Delta+1)$-vertex coloring G in a distributed model of computation. This leads to fast distributed algorithms and a linear-processor NC algorithm for Δ-coloring.

1. Introduction

A main concern in the design of efficient algorithms for distributed networks is locality. A message-passing distributed network can be thought of as a graph where vertices are processors communicating via the edges of the graph; the absence of shared memory disallows the fast dissemination of information and hence, computation must be based on local data. The question of locality can be stated as follows: can each processor compute its part of the output by searching only a small neighborhood of itself?

In a distributed network the following trivial strategy is always possible: the network elects a leader (say, the processor with maximum ID) which then collects all of the information, computes and sends the answers to the rest of the network. This takes time proportional to the diameter of the network (the diameter of a network is the maximum length of a shortest path between any pair of vertices) which can be $\Theta(n)$, where n is the number of nodes in the network. We are interested in "subdiametric" time protocols, in general, ones that run in time polylogarithmic in n.

Mathematics Subject Classification (1991): 68 Q 22, 05 C 15, 68 R 10

A preliminary version of this paper appeared as part of the paper "Improved Distributed Algorithms for Coloring and Network Decomposition Problems", in the *Proceedings of the ACM Symposium on Theory of Computing* pages 581–592, 1992. This research was done when the authors were at the Computer Science Department of Cornell University. The research was supported in part by NSF PYI award CCR-89-96272 with matching funds from UPS and Sun Microsystems.

In this paper we are concerned with the vertex coloring problem in a distributed model of computation, where a synchronous network G wants to compute a vertex coloring of its own topology. The relevance of this problem to distributed computing stems from the fact that an independent set defines a set of processors that can compute in parallel without interfering with their neighbors. Hence, a vertex coloring defines a schedule for the processors to compute in parallel.

Given a graph $G = (V, E)$ with $|V| = n$, Δ will denote its maximum degree, *i.e.*, the maximum number of neighbors of any vertex. A Δ-coloring of a graph is a vertex coloring that uses at most Δ colors. In this paper we prove a surprising result about the "local" nature of Δ-colorings, which has several interesting algorithmic applications.

Theorem. *Let G be a connected graph such that $\Delta \geq 3$, G is not a clique. Suppose $G - v$ is Δ-colored. Then, we can extend the Δ-coloring to the whole of G by recoloring a path originating from v, which is of length at most $O(\log_\Delta n)$.*

This theorem can be used to compute a Δ-coloring of G inductively by adding vertices one by one and each time applying a "small radius search". Hence, this result is a generalization of a well-known theorem of Brooks [3] (see also the discussion in Bollobás [2]), which states that every connected graph of maximum degree Δ which is neither an odd cycle nor a complete graph, can be colored with Δ colors. Brooks' proof does not appear to have this locality property. The $O(\log_\Delta n)$ bound is tight up to a constant factor, in the sense that there exists a family of graphs and partial Δ-colorings of them, for which a search of radius $\Omega(\log_\Delta n)$ is required.

The small radius search can be carried out effectively in our distributed model of computation and in NC, allowing us to derive several algorithmic results. The intuition behind our algorithms is the following. Suppose a graph G is Δ-colored except for a set of uncolored vertices P. If the vertices in P are sufficiently far apart, we can extend the coloring to the whole of G by a simultaneous application of the small radius search to all vertices of P. The problem is to construct a set P with the desired property. We now give an overview of the various algorithmic consequences of the small radius search that we establish in this paper.

We call *nice* graphs the class of connected graphs which are neither paths, nor cycles, nor cliques. We focus on connected graphs because we are interested in studying algorithmic properties of reliable (*i.e.*, no faults) distributed networks.

Our "small radius search" theorem leads to a reduction from Δ-coloring to $(\Delta + 1)$-coloring. This allows us to derive fast randomized distributed and NC algorithms for Δ-coloring because $(\Delta+1)$-colorings can be computed fast. Our first result states that nice graphs are precisely those graphs that can be computed by fast randomized algorithms in our distributed model of computation.

Theorem. *Nice graphs can be Δ-colored in expected polylogarithmic time in the distributed model of computation; the running time is polylogarithmic with high probability. Moreover, there is no $o(n)$ time randomized protocol to Δ-color paths, cycles and cliques.*

Notice the analogy with Brooks' theorem, which characterizes Δ-colorable graphs. This theorem can be viewed as a distributed analog of Brooks' theorem. The $\Omega(n)$ lower-bound is derived as a corollary of a more general result proven in

Section 6 which states that for any $\Delta \geq 2$, the problem of Δ-*edge coloring* bipartite graphs needs $\Omega(\text{diameter}(G))$ time distributively, even given an unlimited amount of randomness (whereas this can be done in NC: see Lev, Pippenger & Valiant [10]). *For vertex coloring,* this result implies that paths and even cycles cannot be 2-vertex colored in $o(n)$ time distributively, even given unlimited randomness. Clearly, cliques and odd cycles cannot be Δ-colored at all.

Randomness can be removed from the previous theorem at a cost of an extra Δ-factor.

Theorem. *Nice graphs are precisely those graphs that can be Δ-vertex colored deterministically in $O(\Delta \log^3 n / \log \Delta)$ time in the distributed model of computation.*

Notice that when Δ is bounded by a polylogarithmic function of n, the running time becomes polylogarithmic in n. (Similarly, when Δ is bounded by a suitable sub-linear function such as, say, n^γ, for $\gamma < 1$, the running time is sublinear).

By using ideas from [12], the randomized reduction can be implemented and derandomized in NC with $O(|V| + |E|)$ processors, yielding the first known linear processor NC algorithm for Δ-coloring. The existing NC algorithms for Δ-coloring all seem to need superlinear processors (Hajnal & Szemerédi [6], Karchmer & Naor [7], and Karloff [8]). With a PRAM paths and even cycles can be 2-colored quickly. We thus have this "PRAM" version of Brooks' theorem.

Theorem. *Nice graphs, paths and even cycles can be Δ-colored in the* CREW PRAM *model of computation in $O(\log^5 n \log\log n / \log \Delta)$ time, with linearly many processors.*

By making use of the notion of *network decomposition* (Awerbuch, Goldberg, Luby & Plotkin [1], Panconesi & Srinivasan [13]) we obtain one final theorem which is another deterministic, distributed version of Brooks' theorem whose running time is independent of Δ.

Theorem. *Nice graphs are precisely those graphs that can be Δ-colored deterministically in $O(n^{O(\epsilon(n))})$ time in the distributed model of computation, where $\epsilon(n) = 1/\sqrt{\log n}$.*

It is an important open problem whether a Δ-coloring or a $(\Delta+1)$-coloring can be computed deterministically in polylogarithmic time in the distributed model of computation.

2. Definitions

A distributed network is a graph G where each vertex is a processor with a distinct ID, and each edge is a bidirectional communication link. There is no shared memory. The network is synchronous and computation takes place in a sequence of *rounds*; during each round a processor sends messages to its neighbors, then collects all data sent to it by its neighbors, and then performs some local computation. The complexity of a protocol is given by the number of rounds. Hence, if we want a protocol to terminate within t rounds, every vertex can communicate with only the vertices which are at a distance of at most t from it. We do not charge for local computation; in other words, we want to study the complexity of a problem when communication is the bottleneck, imposed by this locality (*i.e.*, the absence of a global shared memory).

Given a graph $G=(V,E)$ and a set $S\subseteq V$, $G[S]$ denotes the subgraph induced by S, and $G-S$ denotes $G[V-S]$. When $S=\{v\}$ for some $v\in V$, we write $G-v$ instead of $G-\{v\}$.

With $\mathrm{DIAM}(G)$ we denote the diameter of G. A function $t(\cdot)$ is *subdiametric* (with respect to G) if $t(|V|)=o(\mathrm{DIAM}(G))$.

A vertex coloring will be denoted by $\chi(\cdot)$; if S is a set of vertices, then $\chi(S)$ is the set of colors used by the vertices of S. The maximum degree of G is denoted by Δ. When a vertex v is uncolored, we say that v is *pebbled.* We denote the set of neighbors of a vertex v by $N(v)$, and its degree by $deg(v)$. If v is pebbled and $|\chi(N(v))|<\Delta$, then there is a spare color for v; if we color v with a spare color, the pebble at v is said to be *removed.*

The following operations will be used often. Suppose u is pebbled and $|\chi(N(u))|=\Delta$; let v be any non-pebbled neighbor of u with, say, $\chi(v)=\alpha$. A *step* is the following recoloring operation: *i)* v becomes pebbled, *ii)* u is colored with α, and *iii)* all other vertices stay the same (either colored as before or pebbled). A very important property of the step operation that will be used throughout the paper is that if P is the set of pebbled vertices and a pebble makes a step from u to v, then this step operation transforms a legal Δ-coloring of $G-P$ into a legal Δ-coloring of $G-((P-\{u\})\cup\{v\})$. A *walk* is an arbitrary sequence of steps made by the same pebble (see *Figure 1*). Clearly, a walk transforms "legal" partial Δ-coloring into "legal" partial Δ-coloring (*i.e.*, if P and Q are the sets of pebbles before and after the walk, and if $G-P$ is Δ-colored, then $G-Q$ is Δ-colored).

An (α,β)*-ruling forest* with respect to G and a subset $V'\subseteq V$ is a forest of rooted trees $F=\{T_i\}$, where each tree is a subgraph of G, with the following properties:

- For each i, the root of T_i, called the *leader* of T_i and denoted by $l(T_i)$, is in V';
- each vertex in V' belongs to some tree;
- the trees are disjoint *i.e.*, each vertex in the forest belongs to a unique tree;
- inter-root distance is at least α, and
- tree depth is at most β (the depth of a rooted tree is the maximum distance between the root and any leaf).

Notice that trees of an (α,β)-ruling forest can contain vertices not in V'. This notion was introduced by Cole & Vishkin [4], and generalized by Awerbuch, Goldberg, Luby & Plotkin [1]. A $(k,k\log n)$-ruling forest can be computed in $O(k\log n)$ time distributively [1].

An important notion in distributed graph algorithms is that of a *network decomposition* (also called cluster decomposition) introduced in [1]. Given a graph $G=(V,E)$ and a partition of V into a set of *clusters* C, define the *cluster graph* $G_C=(C,E_C)$, where $E_C=\{(C_i,C_j)\mid i\neq j\wedge \exists u\in C_i,v\in C_j : (u,v)\in E\}$. A $(d(n),c(n))$-network decomposition of G is a set of clusters C and a vertex-coloring of G_C with $O(c(n))$ colors, such that each cluster has diameter $O(d(n))$.

The best-known deterministic result for computing a network decomposition in a distributed system is contained in [13], where it is shown how to compute

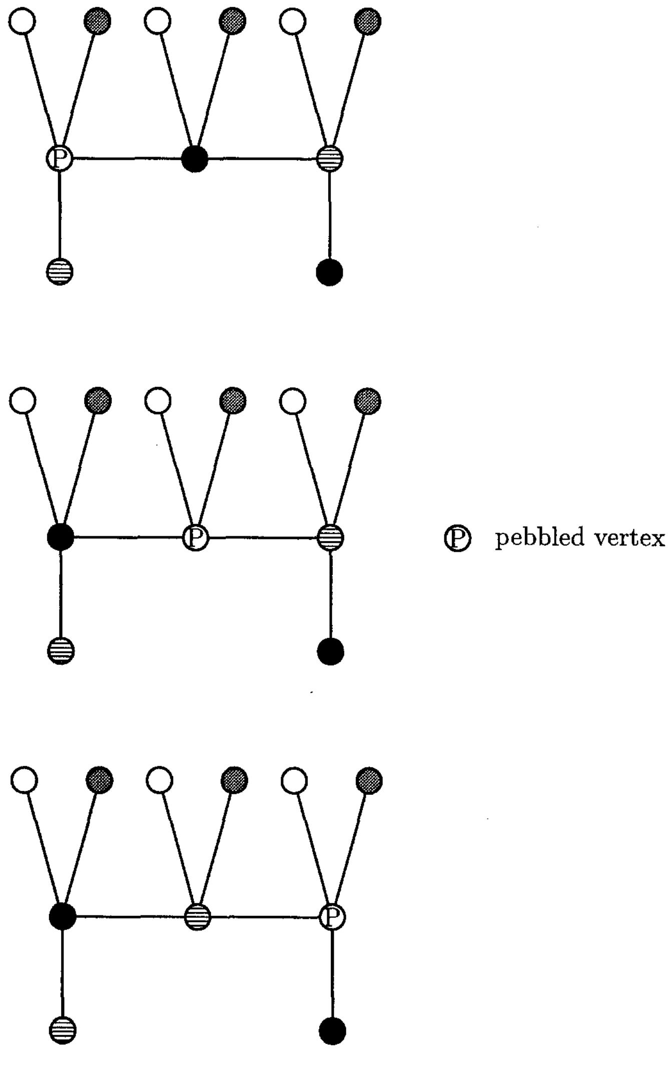

Fig. 1. Steps and walks

an $(O(n^{O(\epsilon(n))}), O(n^{O(\epsilon(n))}))$-decomposition in $O(n^{O(\epsilon(n))})$ time, where $\epsilon(n) = 1/\sqrt{\log n}$.

The next definition introduces formally the class of graphs that we will consider for Δ-coloring:

Definition 1. A *nice* graph is a connected graph G which is not a complete graph, with $\Delta \geq 3$.

3. Distributed Brooks' Theorem

In this section, we show that given a partially Δ-colored nice graph G with one pebbled vertex v_0, we can extend the coloring to G by recoloring an "augmenting path" of length $O(\log_\Delta n)$. Brooks' theorem follows as a corollary. For the sake of clarity, we first give a weaker result, an $O(\sqrt{n})$ bound, and then give the stronger result.

We first establish our result in the easy case of when a vertex of degree less than Δ is "near" the pebbled vertex. Let $G=(V,E)$ be a graph of maximum degree Δ; a vertex $v \in V$ is called a *sanctuary* if $\deg(v) < \Delta$.

Lemma 1. *Let $G-v_0$ be Δ-colored and v_0 be pebbled. If $v \in V$ is a sanctuary at distance ℓ from v_0, then the pebble can be removed by walking it for at most ℓ steps.*

Proof. Let $P=v_0,v_1,\dots,v_\ell \equiv v$ be any simple path between v_0 and v, and consider the following procedure. Initially v_0 is pebbled; if there is a spare color at v_0 remove the pebble, otherwise make a step to v_1. Once v_1 is pebbled, if there is a spare color at v_1 remove the pebble, otherwise make a step to v_2, and so on. This procedure never creates an illegal coloring. Eventually, unless a spare color is found along the way, we reach $v_\ell \equiv v$ which has a spare color because its degree is less than Δ. ∎

Note that the search for a sanctuary within a distance of ℓ can be easily implemented in $O(\ell \log n)$ time both in the distributed model of computation and in the PRAM model using linearly many processors, provided that the vertex degrees have been precomputed. Precomputing degrees takes $O(\log n)$ time.

The rest of this section is devoted to establishing the existence of a short augmenting path in the case when there is no sanctuary near the pebbled vertex. A graph with no sanctuary near the pebble must be "locally Δ-regular". The next definition makes this notion precise.

Definition 2. Let G be a graph with one pebbled vertex v_0. G is *Δ-regular around v_0 within radius ℓ* if there is no sanctuary at a distance of at most ℓ from v_0.

For sake of brevity, we will say that a graph G is Δ-regular within radius ℓ instead of "Δ-regular around v_0 within radius ℓ". The following definition introduces the basic structure that allows us to extend the coloring from $G-v_0$ to G, when G is locally Δ-regular within a radius which will be specified later.

Definition 3. Let $G-v_0$ be Δ-colored and v_0 be pebbled. A *T-path* is a path $P= v_0,v_1,\dots,v_p$, where v_p has two neighbors x and y such that: *(i)* $\chi(x)=\chi(y)$, and *(ii)* $x,y \notin P$.

The notion of T-path is analogous to that of augmenting path which plays a vital role in the theory of matching and network flows. Our aim is to prove that if there is no sanctuary within $O(\log_\Delta n)$ distance from the pebbled vertex then there is a T-path of length $O(\log_\Delta n)$, and to show how to find it. First, we show that a T-path allows us to extend the coloring to G.

Lemma 2. *Let $G-v_0$ be Δ-colored and v_0 be pebbled. If there is a T-path P then G can be Δ-colored by walking the pebble along P.*

Proof. As in the proof of Lemma 1 we walk the pebble along P starting from v_0. Eventually, unless we find a spare color along the way, we reach v_p, which has two neighbors x and y with the same color, and whose colors are not changed by the walk of the pebble. ∎

Given two paths $P_1 = v_0, v_1, \ldots, v_k$ and $P_2 = v_k, v_{k+1}, \ldots, v_l$, their concatenation is the path $P_1 \bullet P_2 = v_0, v_1, \ldots, v_k, v_{k+1}, \ldots, v_l$. The set of colors of the vertices in a path P is denoted by $\chi(P)$. When $|\chi(P)| = 2$ we call P *bichromatic.* If P is bichromatic with colors α and β, we say that P is an (α,β)-path. In what follows, the set of vertices of a path P will be denoted by the same letter P. The next definition is crucial.

Definition 4. A *stem* is a simple path $P = v_0, v_1, \ldots, v_p$ such that:

i) v_0 is pebbled, and

ii) there exists $i > 0$ such that $v_i, \ldots, v_p$ is bichromatic and has at least four vertices (i.e., $p - i \geq 3$).

The basic tool to prove our main theorem is a forthcoming lemma called the Spawning Lemma. Roughly speaking, the lemma states that if we start a walk along a bichromatic path Q originating from (the bichromatic part of) a stem S we cannot intersect S again or, if we do, we have found a T-path. This property will be used to start a tree growing process that, if no T-path is found, visits new vertices at an exponential rate. Hence, pretty soon we must encounter already visited vertices and find a T-path.

The following convention will be handy. Given a path $P = v_0, \ldots, v_p$ and a vertex $v_i \in P$, v_i^+ denotes v_{i+1} and v_i^- denotes v_{i-1} (clearly these are not defined for, respectively, v_p and v_0).

Definition 5. Let $P = v_0, \ldots, v_p$ be a stem with bichromatic part $v_i, \ldots, v_p$. An *ℓ-branch* is a path $Q = x_0, \ldots, x_\ell$ of length ℓ such that:

i) $x_0 \in \{v_{i+1}, \ldots, v_{p-1}\}$,

ii) Q is bichromatic and simple,

iii) $Q \cap P = \{x_0\}$.

Notice that the origin of Q, the vertex x_0, is "internal" to the bichromatic part, namely it belongs to $\{v_{i+1}, \ldots, v_{p-1}\}$. The following lemma states that a stem S can either be used to generate a path visiting "brand new" vertices or to find a T-path.

Lemma 3. *Let G be a nice graph and let $P = v_0, \ldots, v_p$ be a stem with bichromatic part $v_i, \ldots, v_p$. Let $Q = x_0, \ldots, x_\ell$ be a bichromatic path of length ℓ such that $x_0 \in \{v_{i+1}, \ldots, v_{p-1}\}$. Suppose that G is Δ-regular within radius $|P| + |Q|$. Then, either Q is an ℓ-branch or there is a T-path of length at most $|P| + |Q|$.*

Proof. Without loss of generality suppose that $\chi(\{v_i, \ldots, v_p\}) = \{\alpha, \beta\}$, $\chi(Q) = \{\alpha, \gamma\}$, and $\chi(x_0) = \alpha$. Notice first that if Q is not simple then there is a T-path of the desired length (see *Figure 2*).

It remains to prove that either $Q \cap P = \{x_0\}$ or there is T-path. Suppose that $Q \cap P \neq \{x_0\}$, and let w be the first vertex of P that is met when walking along Q coming from x_0. It is important to realize that $w \neq x_0^-, x_0^+$ because $\chi(w) \in \{\alpha, \gamma\}$ and $\chi(x_0^-) = \chi(x_0^+) = \beta$. We distinguish two cases.

Suppose first that $w \in \{v_0, \ldots, v_{p-1}\}$. If $w \in \{v_1, \ldots, x_0^{--}\}$ (w, a colored vertex, cannot be v_0) then there is a path from v_0 to x_0 that does not include x_0^- and x_0^+,

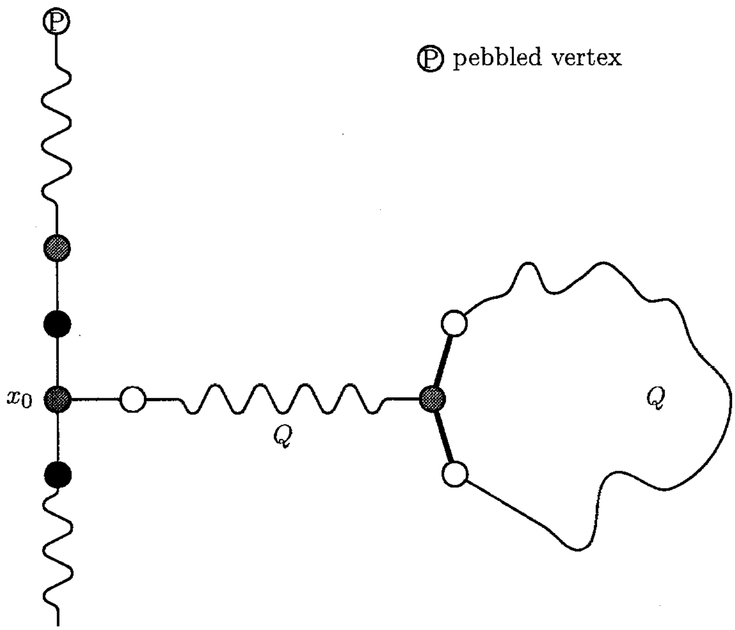

Fig. 2. If Q is not simple a T-path is found

hence a T-path. Similarly, if $w \in \{x_0^{++}, \dots, x_{p-1}\}$ there is a path from v_0 to w that does not include w^- or w^+, again a T-path because $\chi(w^-) = \chi(w^+) = \beta$. (Recall that $v_i, \dots, v_p$ is bichromatic.)

Suppose then that $w = v_p$. Let $A = v_0, \dots, x_0$, and $B = x_0, \dots, v_p \equiv w$. Recall that B is an (α,β)-path and that Q is an (α,γ)-path. Then, it must be that $\chi(v_p) = \alpha$. Again, we distinguish two cases (please refer to *Figure 3*).

- If $N(v_p) \cap A = \emptyset$ then $A \bullet B$ or $A \bullet Q$ is a T-path of the desired length. To see this, suppose that v_p has another (*i.e.*, besides v_{p-1}) neighbor u colored β; then $A \bullet Q$ is a T-path because $u \notin A \cup Q$. Similarly, if v_p has another neighbor colored γ then $A \bullet B$ is a T-path. If v_p has neither, then it has two neighbors with the same color not belonging to $A \bullet B$ or $A \bullet Q$, because $|N(v_p) - (B \cup Q)| = \Delta - 2$, and $|\chi(N(v_p) - (B \cup Q))| \le \Delta - 3$. Hence there is a T-path of the desired length.
- If $N(v_p) \cap A \neq \emptyset$ then a T-path of the desired length can be found as follows. Let $z \in N(v_p) \cap A$. First, $z = x_0$ is impossible because $\chi(x_0) = \chi(v_p) = \alpha$. If $z = x_0^-$ then $v_0, \dots, x^-$ is a T-path, because $\chi(x_0) = \chi(v_p) = \alpha$. Finally, if $z \in \{v_0, \dots, x_0^{--}\}$, there is a T-path from v_0 to x_0, namely the path $v_0, \dots, z, v_p \bullet Q^R$ (this path takes the edge (z, v_p) and then traverses Q "backwards"). This path does not include x_0^- or x_0^+ and hence is a T-path of the desired length. ∎

Given a stem S with bichromatic part $v_i, \dots, v_p$ a *block* is any subpath of four vertices $B = b_0, b_1, b_2, b_3$ such that $B \subseteq \{v_i, \dots, v_p\}$. A path Q originating from either b_1 or b_2 is said to originate from the block B. We can now prove the Spawning

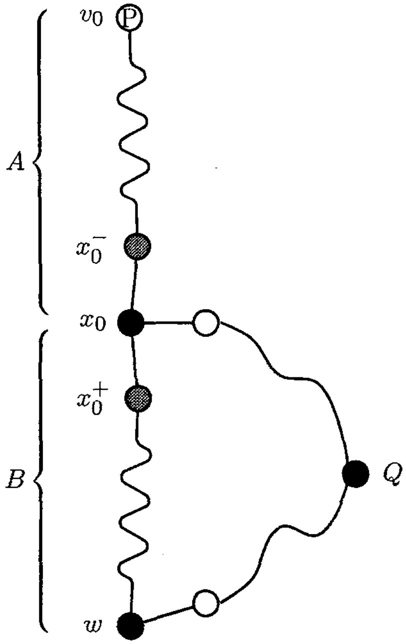

Fig. 3. Q meets the stem S at the last point $w = v_p$

Lemma which states that given a stem S we can either "spawn off" an ℓ-branch from any of its blocks or find a short T-path. From now on, let α, β and γ be *any* three distinct colors.

Lemma 4. [Spawning lemma] *Let G be a nice graph with a pebbled vertex v_0 such that $G - v_0$ is Δ-colored. Let $S = v_0, \ldots, v_p$ be a stem of length at most ℓ, whose bichromatic part $v_i, \ldots, v_p$ is colored with colors α and β, and let $B = b_0, b_1, b_2, b_3$ be a block of S. Suppose that G is Δ-regular within radius 3ℓ, for some $\ell \geq 5$. Then, for any color $\gamma \notin \{\alpha, \beta\}$, either there is an ℓ-branch Q_γ originating from B or there is a T-path of length at most 3ℓ.*

Proof. Without loss of generality suppose $\chi(b_1) = \beta$ and $\chi(b_2) = \alpha$. The structure of the proof is first to try a walk along an (α, γ)-path starting from b_2; if this turns out to be successful (an ℓ-branch or a T-path is found), then we are done. Otherwise, we start a walk along a (β, γ)-path originating from $b_1 = b_2^-$. We will show that if the first search fails the second one must be successful. Let $Q = b_2 \equiv x_0, \ldots, x_q$ be the path obtained by following an (α, γ)-path starting from b_2 for ℓ edges or till it ends, whichever occurs earlier. Q must have non-zero length or otherwise $v_0, \ldots, b_2$ is a T-path (b_2 has no γ-neighbors). By Lemma 3, Q must be a $|Q|$-branch, or there is a T-path of length at most 2ℓ. If Q has length ℓ we are done. Otherwise, we show that either $v_0, \ldots, b_2 \bullet Q$ is a T-path or there is an edge between $b_1 = b_2^-$ and x_q, the last vertex of Q. First, if there is an edge from v_0, the pebbled vertex, to x_q we have a T-path $v_0, x_q, x_{q-1}, \ldots, x_0$ of the desired length. Hence, $N(x_q) - Q$ must contain two vertices a and b with the same color. If they do not belong to the path

$v_0,\ldots,b_1,b_2 \bullet Q$ then we have found a T-path of the desired length. However, only b_1 can be a neighbor of x_q. Indeed, a and b cannot lie on Q because Q is simple, and if, say, $a \in \{v_0,\ldots,b_0\}$ there is a path from v_0 to b_2 which does not include $b_2^- = b_1$ and $b_2^+ = b_3$, hence a T-path is found (the path is $v_0,\ldots,a,x_q,\ldots,x_0 \equiv b_2$).

Hence, if Q is unsuccessful, there must be an edge (b_1,x_p). We then start a (β,γ)-path $Q' = y_0,\ldots,y_r$ starting with $y_0 = b_1$ and $y_1 = x_q$ and continuing for ℓ steps or until it ends, whichever occurs earlier. Notice that Q' has the same properties of Q. If Q' has length ℓ then we are done, otherwise either we find a T-path or there is an edge between $b_0 = b_1^-$ and y_r. But now this edge allows to walk from v_0 to b_2 via the path $v_0,\ldots,b_0,y_r,\ldots,y_1 \bullet x_q,\ldots,x_0 = b_2$, which does not include $b_2^- = b_1$ or $b_2^+ = b_3$ and hence is a T-path of length at most 3ℓ (see *Figure 4*). ∎

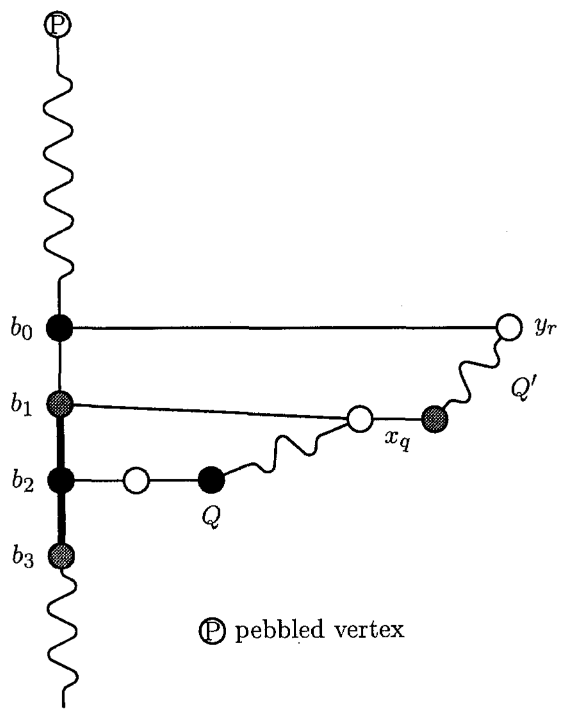

Fig. 4. If both Q and Q' are not ℓ-branches there is a T-path from v_0 to b_2

The above lemma is independent of the particular γ chosen as long as $\gamma \notin \{\alpha,\beta\}$, so that a total of $\Delta-2$ new bichromatic paths can be generated (or "spawned off"), some from v_{i+1} and some from v_{i+2}.

Corollary 1. *Let $G-v_0$ be Δ-colored, v_0 be pebbled, and let G be Δ-regular within radius 3ℓ, for some $\ell \geq 5$. Then, given a stem S of length at most ℓ and a block $B \subseteq S$, we can spawn off $\Delta-2$ ℓ-branches from B, or else there is a T-path of length at most 3ℓ.*

Lemma 4 shows how to inductively generate new stems from old ones. The next lemma shows how to find an initial stem; it is the basis of an induction proof showing the existence of a short T-path.

Lemma 5. *Let G be a nice graph such that $G-v_0$ is Δ-colored and v_0 is pebbled. Suppose that G is Δ-regular within radius 3ℓ, for some $\ell \geq 5$. Then, either G has a stem of length at least four or it has a T-path of length at most four.*

Proof. Since G is not a clique, v_0 has two neighbors x and y such that $(x,y) \notin E$; let $\chi(x)=\alpha$ and $\chi(y)=\beta$. Starting from v_0 we perform a walk along an (α,β)-path P according to the following procedure: let v_i be the current pebbled vertex; if there is a free color at v_i then remove the pebble, otherwise make a step to a vertex v_{i+1} not previously visited, such that $\chi(v_{i+1}) \in \{\alpha,\beta\}$. If no T-path is found this procedure must perform at least four steps, because no sanctuary can be found within 4 steps and the shortest (α,β)-path between x and y must have at least three edges. ∎

By combining Lemmas 4 and 5 we can obtain Brooks' Theorem as a corollary.

Corollary 2. *Every nice graph G can be Δ-colored.*

Proof. The proof is by induction on the number of vertices. The basis is trivial. The induction step is to assume, for some vertex v, that $G-v$ is partially Δ-colored and that v is pebbled. If G is not Δ-regular then there exists a sanctuary at distance at most $n-1$ from v and hence, by Lemma 1, we can extend the coloring to v. Suppose then that G is Δ-regular. First we invoke Lemma 5 to get an initial stem, then we invoke Lemma 4 by setting $\ell = n$. Since branches of such length cannot exist, we must find a T-path of length at most 3ℓ and can remove the pebble. ∎

We now prove that if G is a nice graph such that $G-v_0$ is Δ-colored and v_0 is pebbled, the pebble can be removed by a walk of length at most $O(\sqrt{n})$. Let $\ell = 3\sqrt{n}$. If there is a sanctuary at a distance of at most 3ℓ from v_0, then we are done by Lemma 1; otherwise G is locally Δ-regular within radius 3ℓ and we show that a T-path of at most $O(\sqrt{n})$ length must exist. Lemma 5 ensures that we can find a first stem P of length four. Given P, with one application of Lemma 4, we can spawn off an ℓ-branch P'. This gives a new stem S of length at most $|P'|+|P| = \ell+4$. Then, we subdivide P' into contiguous blocks (adjacent blocks share a vertex), and apply Lemma 4 in each block, thus generating a sequence of bichromatic paths $Q_1, Q_2, \ldots, Q_{\sqrt{n}}$, each of length ℓ. Notice that if any two distinct Q_i and Q_j intersect there is a T-path of length at most $3\ell+4$ (see *Figure 5*). Also, if $|Q_i \cap S| > 1$, for any i, we have found a T-path of length at most $2\ell+4$. However, since $\ell/3$ paths of length ℓ each are generated, some intersection must occur, thus yielding a T-path of length at most $3\ell+4$.

The basic idea of this proof is to generate a tree of diameter $O(\sqrt{n})$ starting from an initial stem and to spawn off ℓ-branches by repeatedly applying Lemma 4. When a new ℓ-branch Q_i is spawned off, we either get a T-path if Q_i intersects the existing tree or *we visit ℓ brand new vertices*. Clearly, after $O(\sqrt{n})$ spawning operations, no new vertices can be visited, and the ℓ-branch must intersect the existing tree.

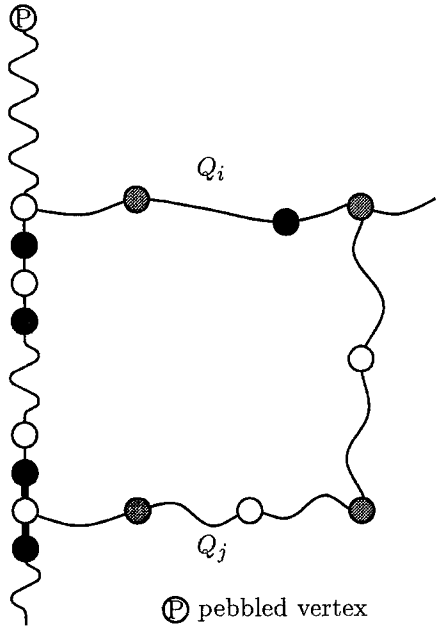

Fig. 5. If Q_i and Q_j intersect a T-path is found

A more intricate use of the same technique shows that we can generate a tree of depth $O(\log_\Delta n)$ with the same properties, hence showing the existence of an $O(\log_\Delta n)$ length T-path.

Theorem 1. *Let G be a nice graph such that $G-v_0$ is Δ-colored and v_0 is pebbled, and suppose that G is Δ-regular within radius $\ell+14$, where $\ell=7\lceil\log_{2\Delta-4} n\rceil+11$. Then, G has a T-path of length at most $\ell+14$.*

Proof. We generate a sequence of trees $\{T_k : k=0,1,2,\ldots\}$ all rooted at v_0; T_{k+1} is generated from T_k by simultaneous applications of Corollary 1.

First, we invoke Lemma 5 and produce a stem S of length four. Then, we invoke Lemma 4 to generate a 7-branch $P=w_0,\ldots,w_7$. There is a path from v_0, the pebbled vertex, to w_7 made of vertices in $S\cup P$; this is the first tree T_0. Notice that T_0 is a stem with bichromatic part $w_0,\ldots,w_7$.

We subdivide $w_1,\ldots,w_7$ into two contiguous blocks $B_1=w_1,\ldots,w_4$ and $B_2=w_4,\ldots,w_7$. From Corollary 1, each B_i can be used to generate $\Delta-2$ new bichromatic paths of length 7. Each of these is subdivided again into two blocks and each block is used to spawn off $\Delta-2$ new paths of length 7, and so on. In this way, we generate a sequence of trees T_k rooted at v_0. T_{k+1} is obtained from T_k by simultaneously spawning off the new length 7 paths from all the unused blocks B in T_k. This process is based on the following invariant: any path from the root v_0 to any leaf w is a stem with bichromatic part of length at least 7. Hence two adjacent blocks can be individuated and Corollary 1 applied. We want to show that when T_{k+1}

is generated from T_k either a short T-path is found or T_{k+1} is indeed a tree (*i.e.*, "brand new" vertices are visited).

Suppose, by induction, that at stage k no T-path has been found (*i.e.*, T_k is a tree) and that T_k has depth at most ℓ. Let B_1 and B_2 be any two unused blocks of T_k. If there is any path from $x \in B_1$ to $y \in B_2$ then we have found a T-path. Hence, any two intersecting 7-branches originating from B_1 and B_2 yield a T-path of length at most $\ell + 14$. Similarly, any 7-branch originating from an unused block B cannot intersect the old tree T_k. To see this, recall that an ℓ-branch cannot intersect its own stem by Lemma 3. If the 7-branch intersects a different stem of T_k a T-path of length at most $\ell+7$ is found (this situation is depicted in *Figure 6*). It

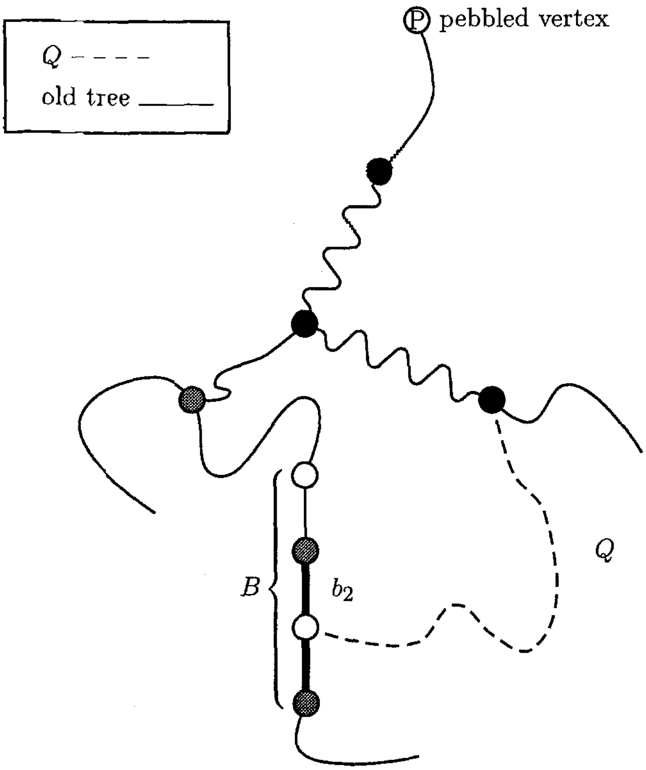

Fig. 6. If a newly spawned off 7-branch Q meets the old tree T_k a T-path is found

remains to show that when 7-branches originating from the *same* block meet a T-path is found. Let $B = b_0, \ldots, b_3$ be an unused block and let $\chi(b_1) = \alpha$ and $\chi(b_2) = \beta$. Recall that there are exactly $\Delta - 2$ 7-branches Q_γ that are spawn off from B, one for each $\gamma \notin \{\alpha, \beta\}$. Each Q_γ is either an (α,γ)-path or a (β,γ)-path. Hence, any Q_{γ_i} originating from b_1 cannot intersect any Q_{γ_j} originating from b_2 because $\chi(Q_{\gamma_i}) \cap \chi(Q_{\gamma_j}) = \emptyset$. If Q_{γ_i} and Q_{γ_j} originate from the same vertex, say b_1, they can meet only in a vertex x colored α. Since Q_{γ_i} and Q_{γ_j} have length seven (an

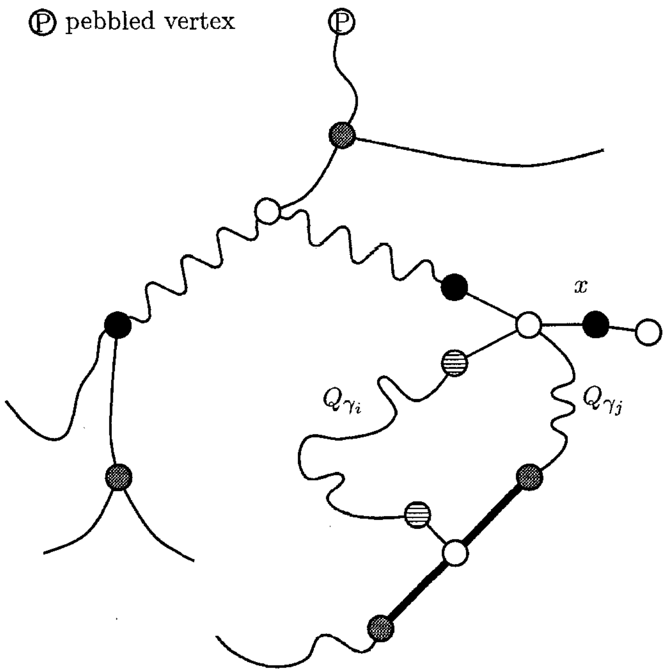

Fig. 7. If Q_{γ_i} and Q_{γ_j} intersect a T-path is found

odd number) x must have two neighbors with the same color on one path, say Q_{γ_i}, thus yielding a T-path of length at most $\ell+14$ (please refer to *Figure 7*).

We now analyze the growth rate of the trees T_k. If we collapse each pair of adjacent *used* blocks in T_k into one vertex, we obtain a tree where every collapsed vertex has degree at least $2\Delta-4$ but for the unused blocks and the initial stem, and whose depth is reduced by at most a factor of 7. Hence, by stage $\ell+1$ we must have found a T-path of length at most $\ell+14$. ∎

Note that for $\Delta \geq 3$, $7\log_{2\Delta-4} n + 11 = \Theta(\log_\Delta n)$. Theorem 1 and Lemma 1 ensure that a short augmenting path can always be found. There is either a sanctuary at a distance of at most $\ell+14 = 7\log_{2\Delta-4} n+25$ from the pebble, or a T-path of at most that length. It can be verified that both Lemma 1 and Theorem 1 can be implemented in $O(\log_\Delta n)$ time in the distributed model of computation, and with linearly many processors in the PRAM model.

Finally, an $\Omega(\log_\Delta n)$ radius search is necessary in general, to remove a pebble. Consider a rooted tree T in which every non-leaf node has degree Δ, with a partial Δ-coloring of T such that there is a pebble at the root, and such that for any non-leaf node v, the colors of v's children are all distinct. The color of at least one leaf of T must be changed to give the root a legal color, since at least one child of v must be recolored, to recolor any non-leaf node v.

4. Algorithms for Δ-coloring

In this section, we show how the small radius search can be applied to the design of efficient distributed and parallel algorithms for computing Δ-colorings. The algorithms are based on a reduction from Δ-coloring to (Δ+1)-coloring which can be implemented distributively by an algorithm running in $O(\log^3 n/\log\Delta)$ time with high probability or by an $O(\Delta\log^3 n/\log\Delta)$ time deterministic algorithm. These yield, respectively, randomized and deterministic distributed algorithms for Δ-coloring with the same complexity bounds.

4.1. The Randomized Reduction

We now describe a randomized distributed algorithm for Δ-coloring that runs in $O(\log^3 n/\log\Delta)$ expected time. The idea is to first compute a $(\Delta+1)$-coloring, which can be thought of as a partial Δ-coloring, and then to remove a color class.

An outline of the algorithm is as follows. Let $G=(V,E)$ be the network. First, compute a $(\Delta+1)$-coloring with colors $1,2,\ldots,\Delta,(\Delta+1)$ and pebble all vertices with color $(\Delta+1)$. In what follows it is convenient to think of the pebbled vertices as colored with the "empty" color, a situation denoted by $\chi(v)=\perp$[1]. Then, compute a $(k,k\log n)$-ruling forest $\mathcal{F}$ with respect to G and the set P of pebbled vertices, where $k=c\log n/\log\Delta$ is more than twice the search radius required by Theorem 1; this can be achieved by an appropriate choice of c. Recall that the root of each tree in the forest is pebbled, and that each pebbled vertex belongs to a unique tree in the forest. If we are able to remove all non-root pebbles, then we can apply the small radius search of Theorem 1 in parallel on the roots, and by our choice of c, each root will either find its own T-path or its own sanctuary, without interfering with the other roots, and will remove the pebble.

The problem, then, is to remove all non-root pebbles. This can be achieved by making use of a randomized process described below, which uses a slight modification of an idea of Luby [12]. The idea behind the reduction is to make all pebbles walk to the root along the path specified by the tree; the pebbles are either removed along the way if a spare color is found, or are eventually "absorbed" by the root, which is itself a pebble. Each walk, however, is a recoloring operation and we must ensure that in doing several of them in parallel, we always have legal partial colorings of the graph. A symmetry-breaking problem arises when we have adjacent pebbles; moving pebbles in parallel might result in an inconsistent coloring (see *Figure 8*).

We now describe the randomized process which allows us to remove all non-root pebbles. Each vertex in G has a list A_v of available colors: $A_v=\{1,\ldots,\Delta\}-\chi(N(v))$, for all v. We denote the current color of v by $\chi(v)$ and the new color after one step by $\chi'(v)$. In parallel, each pebbled vertex v does the following:

[1] That is, $\chi(v)=\perp$ if and only if v is pebbled. Notice that $\perp$ is a special coloring in that neighboring vertices can both be colored with $\perp$.

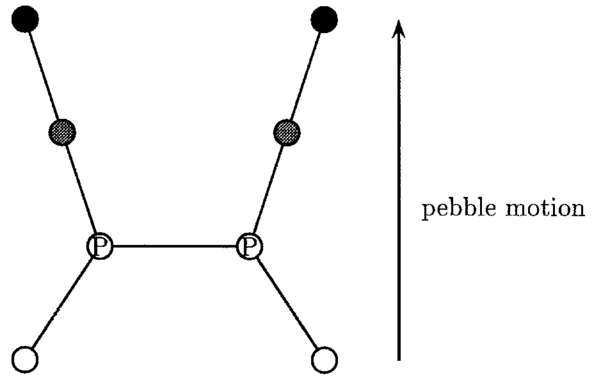

Fig. 8. Moving both pebbles in parallel yields an inconsistent coloring

Randomized Reduction.
If no neighbor of v is pebbled, then the pebble is removed if A_v is nonempty, and the pebble makes a step to v's parent if A_v is empty. If v has some pebbled neighbor, we say that v is *asleep*. With probability 1/2, v remains asleep and does nothing, *i.e.*, $\chi'(v) = \chi(v) = \perp$. With probability 1/2 it *wakes up*, in which case v chooses a tentative color α uniformly at random from A_v; if α is also chosen by some neighbor of v then $\chi'(v) = \chi(v) = \perp$, else $\chi'(v) = \alpha$ and the pebble is removed.

First, we show that by executing the randomized reduction we never produce an inconsistent partial coloring; second, we prove that the expected running time to remove all non-root pebbles is polylogarithmic and in fact, that it is polylogarithmic with high probability.

Lemma 6. *Let G be a nice graph such that $G - P$ is Δ-colored and P is a set of pebbled vertices. Let P' be the set of pebbled vertices after one step of the randomized reduction. Then, $G - P'$ is Δ-colored.*

Proof. The claim follows from inspecting the randomized reduction. If a pebble has no neighboring pebbles then it is removed if there is a spare color, or it makes a step, if there is no spare color. In both cases the new partial coloring is legal. For the case when there are neighboring pebbles let v denote the pebbled vertex. A tentative color is assigned as the new color to v only if the same tentative color was not chosen by any neighboring pebble. The correctness then follows from the observation that non-pebbled neighbors can be pebbled but cannot change their color. ∎

The next lemma shows that all non-root pebbles are removed within $O(\log^3 n / \log \Delta)$ time with high probability: the failure probability is inverse polynomial in n. With essentially the same proof it is possible to show that the running time is $O(f(n) \log^2 n / \log \Delta)$ with probability at least $1 - 2^{-\Omega(f(n))}$ ($f(n)$ is any arbitrary function which goes to infinity as n grows).

Lemma 7. *Let G be a nice graph with n vertices, P be a set of pebbled vertices and $G - P$ be Δ-colored. Suppose $\mathcal{F}$ is a $(k, k \log n)$-ruling forest with respect to G*

and P, where k is any positive integer. Then, if we run the randomized reduction, every non-root pebble is removed within $O(k\log^2 n)$ time with probability at least $1-1/q(n)$, for any polynomial $q(\cdot)$ such that, for all n, $q(n)\geq 1$.

(Comment: the constant implicit in the $O(k\log^2 n)$ term depends on $q(\cdot)$.)
Proof. We want to set up the necessary machinery to invoke a theorem by Karp that will give us the claim [9]. First, we argue intuitively that if $v\in P$ has some pebbled neighbor, then it is removed with probability at least $1/4$; a formal proof can be easily derived and can be found in [12]. Let $B=N(v)\cap P$ be the set of pebbled neighbors of v, and let $W\subseteq B$ be the set of pebbled neighbors of v in B that wake up. Since every pebbled vertex wakes up with probability $1/2$, the expected size of W is $E[|W|]=|B|/2$. Every $u\in W$ chooses a tentative color uniformly at random from its list A_u. In the worst possible scenario, all vertices of W will choose a color which is also in A_v. But since $|A_v|\geq|B|=2E[|W|]$, the probability that v chooses a tentative color not chosen by any $u\in W$ is at least $1/2$. The claim follows from the fact that v wakes up with probability $1/2$.

We can summarize the algorithm by saying that when v is pebbled, the pebble makes a step to the parent of v if there are no neighboring pebbles and there is no spare color, it is removed if there are no neighboring pebbles and there is a spare color, and it is removed with probability at least $1/4$ if there are neighboring pebbles. For the sake of the analysis, we think of the algorithm as follows: if v has no neighboring pebbles and there is no spare color, the pebble makes a step with probability $p=1/4$, otherwise it is removed with probability $p=1/4$. Given ℓ pebbles, we want to study the random variable $T(\ell)$, which denotes the time by which every pebble has either made a step or been removed. Clearly, an upper bound for $T(\cdot)$ with the modified algorithm is also an upper bound for $T(\cdot)$ with the old one.

Let $h(\ell)$ be the random variable denoting the number of pebbles that after one step are neither removed nor have made a step. Then $E[h(\ell)]\leq(1-p)\ell$ (if two or more pebbles step on the same vertex all but one can be removed). Moreover, $T(1)=1$ and $T(\ell)$ satisfies the following recurrence

$$T(\ell) = 1 + T(h(\ell)).$$

Let $b=(1-p)^{-1}=4/3$ and $u(n)=\lfloor\log_b n\rfloor+1$; $u(n)$ is the minimal integer solution to the recurrence

$$U(n) = 1 + U(E[h(n)]) = 1 + U((1-p)n)$$

which intuitively governs the expectation of $T(n)$. By Theorem 3 of Karp [9], it follows that for any $d\geq 1$,

$$Pr(T(n) > u(n) + d) \leq p^{d-1}.$$

This probability is inverse polynomial when $d = O(\log n)$. Also note that this upper bound on the probability applies to any configuration of the pebbles. Hence, $T(n)k\log n$ is an upper bound on the time by which every pebble reaches the root, at which time is certainly removed. The total time taken is hence $O(k\log^2 n)$, with high probability. ∎

We summarize the whole algorithm now.

- Compute a $(\Delta+1)$-coloring of G with colors $1,2,\ldots,\Delta,(\Delta+1)$. This takes $O(\log n)$ with high probability using a randomized algorithm of Luby [9, 12].
- Pebble all the vertices with color $(\Delta+1)$, and let P be the set of pebbled vertices. Compute a $(k, k\log n)$-ruling forest $\mathcal{F}$ with respect to G and P, where $k = c\log_\Delta n$ for a suitable constant c (c is chosen so that $k > 2(7\lceil\log_{2\Delta-4} n\rceil+25)$ insuring that the small radius searches of Step 4 do not overlap). This takes $O(k\log n) = O(\log^2 n/\log\Delta)$ time using an algorithm of Awerbuch, Goldberg, Luby & Plotkin [1].
- Run the randomized reduction. At the end all non-root pebbles are removed. This takes $O(\log^3 n/\log\Delta)$ time with high probability.
- Apply the small radius search on the roots in parallel. Each pebble will either find its T-path or its sanctuary, and will be removed. This takes $O(\log n/\log\Delta)$ time.

The overall complexity is dominated by Step 3. The correctness of the algorithm follows from Lemma 6, which proves the correctness of Step 3, and by the correctness of the small radius search, which ensures that Step 4 is correct. This yields the following theorem.

Theorem 2. *Nice graphs can be Δ-colored in the distributed model in $O(\log^3 n/\log\Delta)$ expected time. Moreover, the running time is $O(\log^3 n/\log\Delta)$ with high probability.*

4.2. Deterministic Distributed Δ-coloring

In this section, we give a distributed, deterministic algorithm for Δ-coloring with complexity $O(\Delta\log^3 n/\log\Delta)$. We stress that when Δ is bounded by a polylogarithmic (sub-linear) function of n, the complexity is polylogarithmic (sub-linear).

In the previous algorithm, randomness was used in two places; to compute a $(\Delta+1)$-coloring and for the randomized reduction. The basic device used to remove randomness is an $O(\Delta\log n)$ time distributed algorithm for $(\Delta+1)$-coloring, due to Goldberg, Plotkin & Shannon [5] which, roughly speaking, substitutes the randomized procedure of Luby.

To remove all non-root pebbles we use the fact that the graph induced by P, the set of pebbled vertices at any given time, is itself $(\Delta+1)$-colorable in $O(\Delta\log n)$ time. The coloring is used to schedule the motion of the pebbles, using the fact that pebbles in a color class can safely take decisions simultaneously. (A color class is an independent set.) We first give the algorithm to remove all non-root pebbles — the deterministic reduction, and then prove its correctness. As in the previous section, prior to invoking the reduction we compute a $(k, k\log n)$-ruling forest, where $k = c\log n/\log\Delta$ for an appropriate value of c.

Deterministic Reduction.

Repeat $k\log n$ times (the maximum tree depth):

1. Let $G[P]$ be the subgraph induced by the set of pebbles P. Compute a $(\Delta+1)$-coloring of $G[P]$ and let $C_1,..,C_{\Delta+1}$ be the color classes.

2. Sequentially, for $i = 1, 2, \ldots, \Delta+1$ do the following: in parallel, each non-root pebbled vertex $v \in C_i$ checks if $|\chi(N(v))| < \Delta$. If so, a spare color is chosen and the pebble is removed.
3. Let Q be the set of remaining non-root pebbles; in parallel, for each pebbled vertex $v \in Q$, if $|\chi(N(v))| < \Delta$ then color v with a spare color and remove the pebble, else the pebble makes a step to v's parent.

In order to prove the correctness of this algorithm it is enough to show that each of the $k \log n$ many iterations transforms a legal partial coloring into a new legal partial coloring. Notice that the coloring of Step 1 is used only to schedule the operations of the pebbles and should not be confused with the partial coloring of G. Step 2 gives a legal partial coloring because each color class C_i is an independent set; if $v \in C_i$ none of its neighbors will change its color, and v can safely color itself if a spare color is available. To prove the correctness of Step 3 we first prove that Q is an independent set. Suppose not, and let u and v be two adjacent pebbles in Q. Without loss of generality suppose that $u \in C_i$ and $v \in C_{i+k}$, where $k > 0$. But since u had an uncolored neighbor when it was processed, namely v, it could have colored itself then. Hence Q is an independent set. A pebbled vertex v in Step 3 either performs a step or colors itself; since Q is an independent set, for all $u \in N(v)$, either u does not change its color or it becomes pebbled (*i.e.* some pebble made a step to u). In either case the color assigned to v is legal.

We now argue that all non-root pebbles are removed by the end of the algorithm. Consider any pebbled vertex v; for each of the $k \log n = O(\log^2 n / \log \Delta)$ iterations, either the pebble is removed or it makes a step towards the root, which decreases the distance of the pebble from the root by one. Each iteration takes $O(\Delta \log n)$ steps (which is the complexity of Step 1), which gives a total of $O(\Delta \log^3 n / \log \Delta)$ time to remove all non-root pebbles. The whole algorithm is summarized as follows.

1. Deterministically compute a $(\Delta+1)$-coloring with colors $1, 2, \ldots, \Delta, (\Delta+1)$ in $O(\Delta \log n)$ time by using an algorithm of Goldberg, Plotkin & Shannon [5]. Pebble all vertices with color $(\Delta+1)$.
2. Compute a $(k, k \log n)$-ruling forest with respect to G and the set of pebbles, where $k = O(\log n / \log \Delta)$. This takes $O(k \log n)$ time [1].
3. Compute the deterministic reduction to remove all non-root pebbles. This takes $O(\Delta k \log^2 n) = O(\Delta \log^3 n / \log \Delta)$ time.
4. Apply the small radius search to all pebbled roots in parallel. Every pebble will either find its own T-path or its own sanctuary, and will be removed. This takes $O(\log_\Delta n)$ time.

The complexity of this algorithm is dominated by that of the deterministic reduction. Hence,

Theorem 3. *Nice graphs can be Δ-colored deterministically in the distributed model of computation in* $O(\Delta \log^3 n / \log \Delta)$ *time.*

5. Further Applications of the Small Neighborhood Search

In this section, we discuss briefly some other applications of the small radius search. First, we show how the randomized algorithm of Section 4.1 can be derandomized in NC with linearly many processors. Second, we discuss a deterministic $O(n^{O(\epsilon(n))})$ time algorithm for Δ-coloring in the distributed model, where $\epsilon(n) = 1/\sqrt{\log n}$.

5.1. A linear processor NC algorithm

The randomized algorithm for Δ-coloring can be implemented and derandomized in the CREW PRAM model using linearly many processors by making use of the standard techniques discussed in [12]. To our knowledge, this is the first linear processor NC algorithm for Δ-coloring; the existing algorithms seem to require superlinear processors [6, 7, 8].

The distributed randomized algorithm of Section 4.1 has four steps. We now describe how each of them can be implemented in the PRAM model.

Step 1 is the $(\Delta+1)$-coloring algorithm of Luby and can be derandomized with linearly many processors [12].

To implement Step 2, computing a ruling forest, and Step 4, performing the small radius search, it is sufficient to simulate the message passing distributed model in NC. This can be easily done by introducing a processor for each edge and by noticing that the *computations* performed at each node only require $O(1)$ time (per step of the distributed algorithm). Simulation of the message passing mechanism requires $O(\log n)$ time because $O(\log n)$ is an upper bound on the message size (recall that we have introduced a processor for each edge). Hence, each round of the distributed algorithm can be simulated in the PRAM model in $O(\log n)$ time. Essentially, both Steps 2 and 4 are BFS searches of $O(\log^2 n/\log\Delta)$ and $O(\log n/\log\Delta)$ depth respectively. Combining these with the time needed for simulating a round gives a complexity of $O(\log^3 n/\log\Delta)$ for both Steps 2 and 4.

To implement the randomized reduction we consider the following modification of Step 3. Let $G-P$ be Δ-colored and P be a set of pebbles. We run (the derandomized NC version of) Luby's $(\Delta+1)$-coloring algorithm on G, which induces a $(\Delta+1)$-coloring of the pebbles with colors $1,\ldots,\Delta,(\Delta+1)$. Let $C_{(\Delta+1)}$ be the pebbles that got color $(\Delta+1)$; all pebbles in $P-C_{(\Delta+1)}$ have a legal color and $C_{(\Delta+1)}$ is an independent set and hence, all pebbles in $C_{(\Delta+1)}$ can make a step to the root.

Notice that here, unlike the distributed implementation, we first run the coloring algorithm and then all pebbles in $C_{(\Delta+1)}$ make a step. This requires a kind of synchronization and global knowledge that is easily available in NC but not in the distributed model.

Each run of the $(\Delta+1)$-coloring algorithm requires $O(\log^3 n \log\log n)$ time [12] and we can have at most $O(\log^2 n/\log\Delta)$ runs before all non-root pebbles in the ruling forest (whose trees have depth $O(\log^2 n/\log\Delta)$) are removed. Paths and even cycles can be easily colored in NC in $O(\log n)$ time; hence, we can state the following theorem.

Theorem 4. *Nice graphs can be Δ-colored in the* CREW PRAM *model of computation with linearly many processors in* $O(\log^5 n \log\log n/\log\Delta)$ *time.*

5.2. A Sublinear Time Deterministic Distributed Algorithm

Problems like MIS and $(\Delta+1)$-coloring can be solved in $O(d(n)\cdot c(n))$ time distributively, given a $(d(n),c(n))$-decomposition of G. The generic algorithm for such problems, given a network decomposition, will iterate through the color classes, clusters of color 1 being "processed" first in parallel, clusters of color 2 being processed next, and so on. Inside each cluster the following trivial algorithm can be used: the maximum ID vertex within the cluster is elected as leader, which then collects information about all vertices in the cluster, solves the problem by itself, and then distributes the solution to all vertices in the cluster. The bounds on the cluster diameter and the number of colors used, yield the bound on the time complexity of this generic algorithm.

It is known how to compute an $(O(n^{\epsilon(n)}),O(n^{\epsilon(n)}))$-decomposition distributively in $O(n^{O(\epsilon(n))})$ time where $\epsilon(n)=1/\sqrt{\log n}$ [13]. Such a decomposition can be used to give a deterministic and distributed implementation of Step 1 and Step 3 of our Δ-coloring algorithm.

Step 1, computing a $(\Delta+1)$-coloring, can be implemented with the generic algorithm outlined above: cycle through the color classes, and when processing color class c, extend the partial $(\Delta+1)$-coloring to all clusters of color c. The extension to the coloring in each cluster of color c can be computed by the leader of the cluster by means of global communication inside the cluster, with the time complexity being proportional to the diameter of the cluster. Since both the number of colors and cluster diameter are $O(n^{O(\epsilon(n))})$, the total cost of this implementation is $O(n^{O(\epsilon(n))})$.

A naive implementation of the reduction of Step 3 is as follows. Let $t(n)=O(\log^2 n/\log\Delta)$ be the maximum tree depth of the ruling forest, $d(n)=O(n^{O(\epsilon(n))})$ be an upper bound on the diameter of each cluster of the network decomposition, and let $c(n)=O(n^{O(\epsilon(n))})$ be the number of colors used in the network decomposition. Then, for $i=1,2,\ldots,t(n)$ and for $c=1,2,\ldots,c(n)$ do the following: each leader in clusters of color c schedules the motion of the pebbles inside the cluster until they are either removed or step outside the boundary of the cluster. Inside each cluster the trivial algorithm outlined above is used. This takes $O(t(n)c(n)d(n))=O(n^{O(\epsilon(n))})$ time, where $\epsilon(n)=1/\sqrt{\log n}$.

The main observation is that each time a cluster is activated, each pebble in the cluster is either removed or makes at least one step. So, it is sufficient to activate each cluster $t(n)$ times to remove all non-root pebbles.

The correctness of the implementations of Step 1 and Step 3 follows from the fact that a node in a cluster C cannot be adjacent to a node in a cluster C' whose color is the same as that of C. Step 4 of the algorithm can be implemented with a BFS of depth $O(\log n/\log \Delta)$ at most. The following theorem summarizes the whole discussion.

Theorem 5. *Nice graphs can be Δ-colored deterministically in $O(n^{O(\epsilon(n))})$ time in the distributed model of computation, where $\epsilon(n) = 1/\sqrt{\log n}$.*

6. Lower Bounds for some Distributed Coloring Problems

In this section, we prove an $\Omega(\mathrm{DIAM}(G))$ lower bound for *edge coloring* bipartite graphs optimally (*i.e.* with Δ colors) in the distributed model of computation, and then show that the same lower bound applies even when the processors are allowed randomness. When $\Delta = 2$, the proofs for vertex and edge coloring are the same. Linial has proved lower-bounds for computing various types of vertex colorings distributively, using different techniques [11].

6.1. Deterministic Coloring of Paths

We first analyze the simpler case of coloring paths and then we will deal with general bipartite graphs. For paths and even cycles a lower bound proof for vertex coloring readily translates into one for edge coloring, and viceversa. We will describe our lower-bounds in terms of vertex coloring.

Theorem 6. *Let $t(n) = o(n)$. There is no distributed protocol which 2-colors all connected graphs of maximum degree-2 in $O(t(n))$ time.*

Proof. We consider the case where G is a path; the proof is similar if G is an even cycle. The motivation for this result is that two-coloring a path amounts to computing the parity of a given vertex.

Let $s(i,t)$ be the state at time t of (the processor corresponding to) vertex with ID i. From the definition of our computation model, it follows that for any path-coloring protocol,

$$s(i,t) = f(t, i, i_L, i_R, s(i,t-1), s(i_L, t-1), s(i_R, t-1)),$$

for some function $f(\cdot)$, and where i_L and i_R are the ID's of the neighbors of i. Also, $s(i,0) = g(\mathrm{ID}(i))$ for some function $g(\cdot)$. The above equation implies that if $d(i,j)$ is the distance between two vertices i and j, any information starting from i needs $d(i,j)$ steps to reach j. This observation is the basis for the proof. Let $t = t(n)$ be the worst-case complexity of a protocol $\mathcal{P}$ for two-coloring paths, and assume that $t(n) = o(n)$. Consider a path $A: v_1, .., v_{2t}, .., v_{i-t}, .., v_i, ..v_{i+t}, ..v_{n-1}, v_n$; notice that v_i is surrounded by a neighborhood of radius t and it is at least $3t+1$ far away from v_1. Consider now the path where v_n is inserted somewhere between v_{2t} and v_{i-t}; this changes the parity of the path, *i.e.* the coloring of v_1 and v_i in A must be different from that in B, when the protocol $\mathcal{P}$ is used. But from the state transition function it follows that

$$s_A(i,k) = s_B(i,k) \quad \& \quad s_A(v_1,k) = s_B(v_1,k), \quad 0 \le k \le t,$$

where the subscripts A and B denote the paths A and B. In other words, for any sublinear $t(n)$, the states of v_i and v_1 in A and B will be exactly the same and hence they will receive the same colors in both cases, contradicting the presumed correctness of $\mathcal{P}$. ∎

6.2. Optimal Edge Coloring of Bipartite Graphs

In this subsection, we prove the same lower bound of $\Omega(\mathrm{DIAM}(G))$ for edge coloring general bipartite graphs optimally, *i.e.*, with Δ colors. The idea of the proof is the same as before; if a protocol is constrained to finish within t steps, then a vertex cannot "tell the difference" between two situations where the topology of the network is the same in a neighborhood of radius t, but different outside this neighborhood.

Theorem 7. *For any $\Delta \geq 2$, there is no subdiametric time deterministic protocol for Δ-edge coloring bipartite graphs with maximum degree Δ, in the distributed model.*

Proof. The proof is by contradiction. Our graph G will be made by linking together in a chain-like manner certain subgraphs. Each subgraph is defined as follows. Consider a complete bipartite graph $K_{\Delta-1,\Delta-1}$ and let $b_1, .., b_{\Delta-1}$ be the vertices on one side of the bipartition and $c_1, .., c_{\Delta-1}$ the other side. Connect all of the b_i's to a vertex a and all of the c_i's to another vertex d. Finally, connect d to another vertex e. Such a graph will be called a Δ-widget. A 5-widget is shown in *Figure 9*.

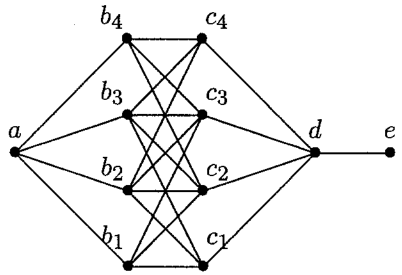

Fig. 9. A 5-widget

The widget is such that it forces a color on edge (d, e), as follows. A Δ-widget is a bipartite graph of maximum degree Δ and hence can be Δ-edge colored. Without loss of generality, suppose we use colors $1, .., \Delta - 1$ for the edges incident on vertex a. Hence, among the remaining edges incident on any b_i exactly one of them must use color Δ. This means that each c_i has exactly one edge (c_i, b_j) colored Δ. In turn, this implies that none of the edges incident on vertex d can use color Δ and this forces us to use color Δ on edge (d, e).

We build a bipartite graph by connecting Δ-widgets in a chain-like manner so that the e vertex of one widget coincides with the a vertex of the next (please refer to *Figure 10*).

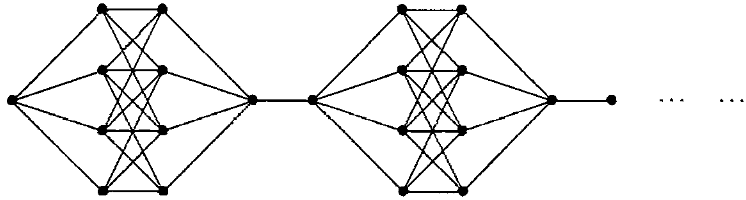

Fig. 10. A chain of widgets

To prove our claim, let W_i be the $i-th$ widget and let a_i, b_{ij}, c_{ij}, d_i be its vertices. The main observation is that if $\chi(d_i, a_{i+1}) = \Delta$ then $\chi(d_k, a_{k+1}) = \Delta$ for all k, and the same argument of Theorem 6 can be repeated. Suppose there is a protocol $\mathcal{P}$ to Δ-edge color bipartite networks with n vertices and with maximum degree Δ, with subdiametric worst-case time $t = t(n, \Delta)$. Consider a chain A with at least $3t+3$ Δ-widgets. Without loss of generality, assume that all edges (d_i, a_{i+1}) of the i-th widget are colored with color Δ. We now modify the chain A by removing any vertex v from the last widget W_{3t+3} and inserting it between d_t and a_{t+1}; this creates the two new edges (d_t, v) and (v, a_{t+1}). Let this be chain B. Clearly, the insertion of v implies that color Δ cannot be used on both edges incident on v; this implies that the coloring of the two subchains at the left and right side of v must be different. However, if we consider widgets W_1 and W_{2t+2}, they will behave exactly the same as they did in chain A since they are at distance greater than t from v and from W_{3t+3}, and this will cause a conflict of colors somewhere in the chain. ∎

6.3. Randomized Coloring

We now prove that the same $\Omega(\mathrm{DIAM}(G))$ lower bound applies when the processors are allowed to use random bits. At each step of the protocol, each processor can flip a fair coin independently any number of times; this is equivalent to assuming that each processor is given all of its random bits at the beginning of the protocol. There are two types of randomized protocols: `Monte Carlo` and `Las Vegas`. The definition of acceptance for a Monte Carlo protocol is that the protocol should find a Δ-edge coloring in any degree Δ bipartite graph with probability at least p, with $p > 1/2$. On the other hand, a Las Vegas protocol always computes the correct answer, but its running time is a random variable.

Theorem 8. *There is no Monte Carlo distributed protocol that finds a two-coloring of a path with n vertices in worst-case time $o(n)$.*

Proof. We use the same strategy as for the previous proof. Assume that there is a protocol $\mathcal{P}$ which violates the assumptions of the theorem; given a path A where $\mathcal{P}$ is supposed to work, construct a new path B by changing the parity of two vertices. If we take these two vertices far enough they will behave in the same way in both chains, and the resulting coloring in B will be invalid.

Let A be a path of length n such that $4|n$ and $n/4 > 2t$ (where $t = o(n)$ is the worst-case time complexity of $\mathcal{P}$), and let us subdivide it into 4 parts of equal length:

$$A = \overbrace{v_1 \dots v_{n/4}}^{A_1} \overbrace{v_{(n/4)+1} \dots v_{2n/4}}^{A_2} \overbrace{v_{(2n/4)+1} \dots v_{3n/4}}^{A_3} \overbrace{v_{(3n/4)+1} \dots v_n}^{A_4}.$$

Let the parts be A_1, A_2, A_3, and A_4. Let b be the number of random bits assigned to each processor. Given the random assignments to the processors (b bits to each of the n processors) we form a string of length bn by concatenating the assignments; we call this a *string assignment*. Since the protocol works on A with probability at least $1/2$, there must be at least 2^{bn-1} string assignments that find a right coloring; let S be this set of string assignments. Consider $H_1 = A_1 \cup A_3$ and $H_2 = A_2 \cup A_4$. Let $S_1 = \{a_1 \circ a_3 \in \{0,1\}^{bn/2} \mid \exists x, y \in \{0,1\}^{bn/4} \exists s \in S\, s = a_1 \circ x \circ a_3 \circ y\}$ and let $n_1 = |S_1|$. Similarly, let $S_2 = \{a_2 \circ a_4 \in \{0,1\}^{bn/2} \mid \exists x, y \in \{0,1\}^{bn/4} \exists s \in S\, s = x \circ a_2 \circ y \circ a_4\}$ and $n_2 = |S_2|$. Without loss of generality, suppose $n_1 \geq n_2$; since $|S| \leq |S_1 \times S_2|$, $n_1 n_2 \geq 2^{bn-1}$. It follows that

$$n_1 \geq 2^{\frac{bn-1}{2}}.$$

Let us now construct a new path B by removing v_n, the last vertex of A_4, and inserting it in the middle of A_2. We now claim that the probability that in B the vertices in H_1 compute exactly the same colors for themselves as they compute in A is at least $1/\sqrt{2}$. This would give us the claim. Notice that since v_n is at distance greater than t from the vertices in H_1, the vertices in H_1 will compute an invalid coloring when given any sequence of random strings from S_1, for *any* assignment of random strings to H_2. This happens with probability

$$n_1 \frac{2^{\frac{bn}{2}}}{2^{bn}} \geq \frac{2^{\frac{2bn-1}{2}}}{2^{bn}} = \frac{1}{\sqrt{2}},$$

which is greater than $1/2$. ∎

Theorem 9. *There is no Las Vegas distributed protocol that finds a two-coloring of a path with n vertices in expected time $o(n)$.*

Proof. Assume that there is such a protocol $\mathcal{P}$ with expected running time at most $T(n) = o(n)$. Given a path, run $\mathcal{P}$ on it for $2 \cdot T(n)$ steps; by Markov's inequality, a valid two-coloring would have been found with probability at least $1/2$ in time $2 \cdot T(n) = o(n)$, violating Theorem 8. ∎

Corollary 3. *There is neither a Monte Carlo nor a Las Vegas distributed protocol that computes Δ-edge colorings of bipartite graphs of maximum degree Δ in subdiametric time.*

Proof. The same arguments as in Theorems 8 and 9 go through if we use the chain of widgets of Theorem 7 instead of a path. ∎

Acknowledgments. Our sincere thanks go to Prof. David Shmoys, for his generous advice, support and suggestions. Many thanks to Radhakrishnan Jagadeesan and Suresh Chari who formalized the lower bound approach for us. We also thank Prof. Gianfranco Bilardi, Prasad Jayanti, David Pearson, Desh Ranjan and Pankaj Rohatgi for valuable discussions, and Prof. C. R. Muthukrishnan for his kind help during the summer of 1991.

References

[1] B. AWERBUCH, A. V. GOLDBERG, M. LUBY, and S. A. PLOTKIN: Network decomposition locality in distributed computation, in: *Proceedings of the IEEE Symposium on Foundations of Computer Science*, (1989), 364–369.

[2] B. BOLLOBÁS: *Graph Theory*, Springer Verlag, New York, 1979.

[3] R. L. BROOKS: On colouring the nodes of a network, *Proc. Cambridge Phil. Soc.* **37** (1941), 194–197.

[4] R. COLE, and U. VISHKIN: Deterministic coin tossing with applications to optimal parallel list ranking, *Information and Control* **70** (1986), 32–53.

[5] A. V. GOLDBERG, S. A. PLOTKIN, and G. E. SHANNON: Parallel symmetry-breaking in sparse graphs, *SIAM J. Disc. Math.* **1** (1989), 434–446.

[6] P. HAJNAL, and E. SZEMERÉDI: Brooks coloring in parallel, *SIAM J. Disc. Math.* **3** (1990), 74–80.

[7] M. KRACHMER, and J. NAOR: A faster parallel algorithm to color a graph with Δ colors, *Journal of Algorithms* **9** (1988), 83–91.

[8] H. J. KARLOFF: An NC-algorithm for Brooks' theorem, *Theoretical Computer Science* **68(1)** (1989), 89–103.

[9] R. M. KARP: Probabilistic recurrence relations, in: *Proceedings of the ACM Symposium on Theory of Computing* (1991), 190–197.

[10] G. F. LEV, N. PIPPENGER, and L. G. VALIANT: A fast parallel algorithm for routing in permutation networks, *IEEE Transactions on Computers* **30** (1981), 93–100.

[11] N. LINIAL: Locality in distributed graph algorithms, *SIAM J. Comput.* **21(1)** (1992), 193–201.

[12] M. LUBY: Removing randomness in parallel computation without a processor penalty, *Journal of Computer and System Sciences,* **47** (1993), 250–286.

[13] A. PANCONESI, and A. SRINIVASAN: Improved distributed algorithm for coloring and network decomposition problems, in: *Proceedings of the ACM Symposium on Theory of Computing* (1992), 581–592.

Alessandro Panconesi

Freie Universität Berlin
Fachbereich für Mathematik und Informatik
Institut für Informatik
Takustr. 9
D-14195 Berlin, Germany

Aravind Srinivasan

DIMACS Center
Rutgers University
PO Box 1179
Piscataway, NJ 08855-1179, USA