The Local Nature of Δ-Coloring and Its Algorithmic Applications*†

Alessandro Panconesi
Aravind Srinivasan

TR 92-1303
September 1992

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*A preliminary version of this paper appeared as a part of the paper "Improved Distributed Algorithms for Coloring and Network Decomposition Problems", in the *Proceedings of the ACM Symposium on Theory of Computing*, pp. 581-592, 1992.
†This research was supported in part by NSF PYI award CCR-89-96272 with matching support from UPS and Sun Microsystems.

The Local Nature of Δ-coloring and Its Algorithmic Applications *†

Alessandro Panconesi & Aravind Srinivasan
Department of Computer Science
Cornell University, Ithaca NY 14853
E-mail: {ap, srin}@cs.cornell.edu

September 16, 1992

Abstract

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 when designing 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.

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

*A preliminary version of this paper appeared as a 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 supported in part by NSF PYI award CCR–89–96272 with matching support from UPS and Sun Microsystems.

own topology. This problem is interesting because a coloring is a partition of the vertices into independent sets, thus defining a schedule for the processors to compute in parallel, since an independent set defines a set of processors that can compute in parallel without interfering with their neighbors.

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 consequences.

Theorem *Let G be a connected graph such that $\Delta \geq 3$, G is not a clique, and all but one vertex v of G 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 uncoloreed 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.

There is a simple and beautiful randomized distributed algorithm to compute a $(\Delta + 1)$-coloring in $O(\log n)$ expected time, due to Luby [12]. 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. We also prove that for any $\Delta \geq 2$, the problem of Δ–edge coloring a bipartite graph G needs $\Omega(diam(G))$ time distributively, even given an unlimited amount of randomness (whereas this can be done in NC: see Lev, Pippenger & Valiant [10]). For paths and even cycles, edge coloring and vertex coloring are equivalent and hence, we can state the following distributed version of Brooks' theorem:

Theorem *A connected graph G is Δ-colorable in expected polylogarithmic time in the distributed model of computation if and only if G is neither a complete graph nor a degree–2 graph.*

When Δ is bounded by a polylogarithmic function of n, we can implement the reduction to

$(\Delta + 1)$–coloring deterministically in polylogarithmic time, distributively.

Theorem *A connected graph G is Δ–colorable in $O(\Delta$ polylog$(n))$ time in the distributed model of computation if and only if G is neither a complete graph nor a degree–2 graph.*

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]).

Theorem *A connected graph G can be Δ–colored in the* PRAM *model of computation with linearly many processors and in polylogarithmic time if and only is G is neither a clique nor an odd cycle.*

The reduction to $(\Delta + 1)$-coloring can be implemented sequentially with a depth first search (DFS), which yields a linear time sequential algorithm for Δ-coloring. The details of this construction do not appear in this paper.

By making use of the notion of *network decomposition* (Awerbuch, Goldberg, Luby & Plotkin [1], Panconesi & Srinivasan [13]) we obtain one final theorem.

Theorem *A connected graph G is Δ–colorable in $O(n^{\epsilon(n)})$ time in the distributed model of computation where $\epsilon(n) = O(1/\sqrt{\log n})$, if and only if it is neither a complete graph nor a degree–2 graph.*

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

We first introduce our distributed model of computation and then give some graph–theoretic 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 one 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 $G - \{v\}$.

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*, and represent this situation by letting $\chi(v) = \perp$. If S is a set of pebbled vertices and $G - S$ is legally Δ–colored we say that G is *partially Δ-colored.* We denote the set of neighbors of a vertex v by $N(v)$, and its degree by $deg(v)$. If v is pebbled and $|\chi(N(v)) - \{\perp\}| < \Delta$, 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 v is pebbled and $|\chi(N(v)) - \{\perp\}| = \Delta$; let u be any non–pebbled neighbor of v with color α, say. A *step* is the following recoloring operation: $\chi'(v) = \alpha$, $\chi'(u) = \perp$, and $\chi'(w) = \chi(w)$, for all $w \in V - \{u, v\}$. 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 $G - ((P - \{v\}) \cup \{u\})$. A *walk* is a sequence of steps (see Figure 1). Clearly, a walk transforms a partial Δ–coloring into another partial Δ–coloring.

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

The next definition introduces 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 $\Delta \geq 3$.*

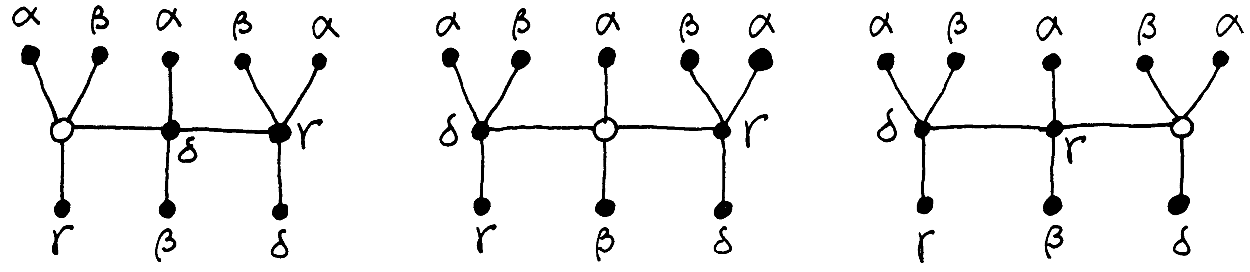

Figure 1: Steps and walks

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 be a nice graph with one pebbled vertex v_0. 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 $P = v_0, v_1, \ldots, v_\ell \equiv v$ 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 we remove the pebble, otherwise we make a step to v_1. Once v_1 is pebbled, if there is a spare color at v_1 we remove the pebble, otherwise we make a step to v_2, and so on. This procedure is correct because each step maintains a partial 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 $O(\ell)$ time both in the distributed model of computation and in the PRAM model using linearly many processors.

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

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 be a partially Δ-colored nice graph with one pebbled vertex v_0. A* T-path *is a path $P = v_0, v_1, \ldots, v_p$, where v_p has two neighbors x and y such that:* (i) $\chi(x) = \chi(y)$, *and* (ii) $x, y \notin P$.

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. The idea is that the pebble can walk from v_0 to v_p along the T-path; once at v_p the pebble can be removed because neither x nor y has its color changed by the walk.

Lemma 2 *A partially Δ-colored graph with one pebbled vertex v_0 and a T-path P, 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.

Definition 4 *A* stem *is a simple path $P = v_0 \bullet P_1 \bullet P_2$ such that: (i) v_0 is pebbled, and (ii) P_2 has at least four vertices and is bichromatic.*

Definition 5 *Let $P = v_0, v_1, \ldots, v_p = v_0 \bullet P_1 \bullet P_2$ be a stem, where $P_2 = v_i, v_{i+1}, \ldots, v_p$. A* P-detour *is any simple path P' such that:* (i) *P' has end-points $v_j \in P$ and at $v_{j+k} \in P_2$, where $i < j + k < p$ and $k \geq 2$, and* (ii) $P' \cap P = \{v_j, v_{j+k}\}$.

If P' is a P-detour then $v_0, .., v_j \bullet P'$ is a T-path (see Figure 2). In the sequel, when there is no danger of confusion, we shall refer to a P-detour simply as a detour.

The next lemma is the basic tool used to prove our main theorem. It states that if G is Δ-regular within radius 3ℓ, then given a stem P of length at most ℓ we can either spawn off a bichromatic path P' of length ℓ, or we can find a T-path of length at most 3ℓ. By repeated applications of this spawning operation we can grow what roughly is a Δ-regular tree. If we show that by growing the tree we always include only additional vertices then the tree rapidly covers G and we can find a T-path of length $O(\log_\Delta n)$.

Suppose we have a path $P = v_0, \ldots, v_p$, and a path P' originating from some vertex $v \in P$. If $P \cap P' \subseteq \{v, v_p\}$ we say that P and P' are *divergent.* From now on, let α, β and γ be three distinct colors.

Definition 6 *Let $P = v_0 \bullet P_1 \bullet P_2$ be a stem. An ℓ-*branch *is any bichromatic path originating from $v \in P_2$ which is simple, divergent from P and is of length ℓ.*

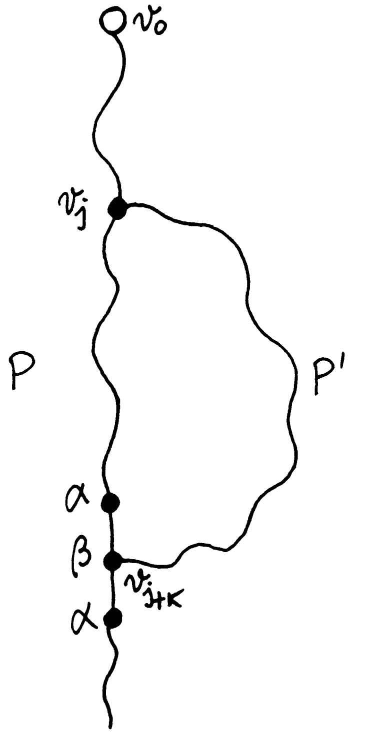

Figure 2: A detour yields a T-path

Lemma 3 *Let G be a partially Δ-colored nice graph with one pebbled vertex v_0, and let G be Δ-regular within radius 3ℓ, for any $\ell \geq 5$. Let $P = v_0 \bullet P_1 \bullet P_2$ be a stem of length at most ℓ, such that $P_2 = v_i, \ldots, v_p$, $\chi(P_2) = \{\alpha, \beta\}$, and $\chi(v_i) = \alpha$. Then, for any γ different from α and β, either there is an ℓ-branch P_{i+1} originating from v_{i+1} such that $\chi(P_{i+1}) = \{\beta, \gamma\}$, or there is an ℓ-branch P_{i+2} originating from v_{i+2} such that $\chi(P_{i+2}) = \{\alpha, \gamma\}$, or there is a T-path of length at most 3ℓ.*

Proof. Let P_{i+1} be the path obtained by following a (β, γ)-path Q_1 starting from v_{i+1} for ℓ edges or till Q_1 ends, whichever occurs earlier; let P_{i+2} be the path obtained by following an (α, γ)-path Q_2 starting from v_{i+2} for ℓ edges or till Q_2 ends, whichever occurs earlier. Let z' and z be the last vertices of P_{i+1} and P_{i+2} respectively. We first show that either P_{i+2} is simple and divergent from the stem P, or there is a T-path of length at most 2ℓ. If P and P_{i+2} are not divergent, there is a detour to or from v_{i+2} and hence a T-path of length at most 2ℓ. On the other hand, if P_{i+2} is not simple then $P' = v_0, \ldots, v_{i+2} \bullet P_{i+2}$ contains a T-path (see Figure 3).

If the length of P_{i+2} is ℓ, then we are done. Otherwise, consider P_{i+1}. First, we show that if P_{i+1} does not go through z, the last vertex of P_{i+2}, we are done. Since z is the last vertex of the bichromatic path P_{i+2} and since G is Δ-regular within radius 3ℓ, z must have two neighbors x and y with the same color (if the pebbled vertex v_0 is a neighbor of z then there is a detour from v_0 to v_{i+2} of length at most $\ell + 1$). If $P' = v_0, .., v_{i+2} \bullet P_{i+2}$ is not a T-path, at least one of x and y, say x, must be in P'. But $x \notin P_{i+2}$ because P_{i+2} is simple, and $x \notin \{v_0, .., v_i\}$ or there is a detour from x to v_{i+2} of length at most $\ell + 1$. The only possibility then is that $x = v_{i+1}$. If $\chi(z) = \alpha$, then we are done since $v_0, v_1, \ldots, v_{i+1}$ is a T-path, otherwise consider the (β, γ)-path $P_{i+1} = v_{i+1}, z, \ldots, z'$. The last vertex z' of P_{i+1} again must have two neighbors x' and y' with the same color and one of them, say x', must be v_i, or else we have a short T-path. But this

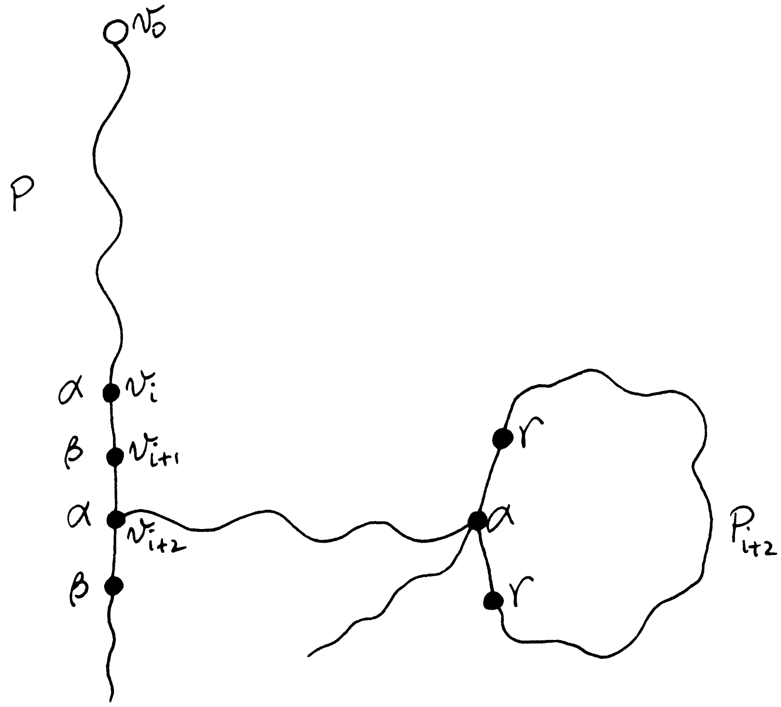

Figure 3: If P_{i+2} is not simple we find a T-path

is also ruled out, or else we could reach v_{i+2} with a detour starting from v_i that uses P_{i+1} and P_{i+2} "backwards", resulting in a T-path of length at most 3ℓ (see Figure 4). $\square$

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

Corollary 1 *Let G be a partially colored nice graph with one pebbled vertex, and let G be Δ-regular within radius 3ℓ, for any $\ell \geq 5$. Then, given a stem P of length at most ℓ, we can spawn off $\Delta - 2$ ℓ-branches from it, or else there is a T-path of length at most 3ℓ.*

Lemma 3 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 4 *Let G be a partially Δ-colored nice graph with one pebbled vertex v_0, and let G be Δ-regular within radius 3ℓ, for any $\ell \geq 5$. Then G either has a stem of length at least four or 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

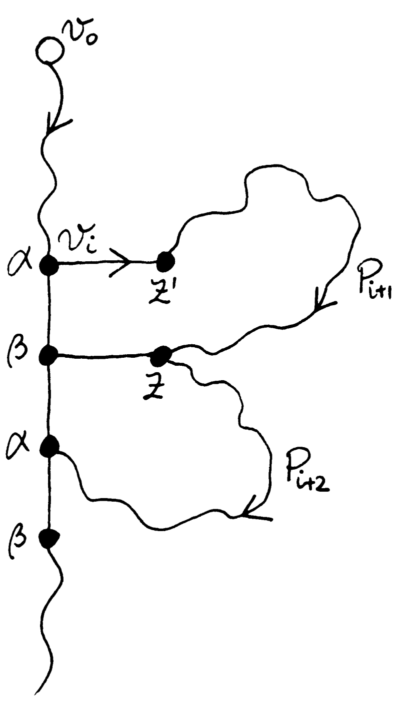

Figure 4: If P_{i+1} and P_{i+2} fail we find a T-path

have at least three edges. □

By combining Lemmas 3 and 4 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 4 to get an initial stem, then we invoke Lemma 3 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 partially Δ-colored graph with one pebbled vertex v_0, then 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 4 ensures that we can find a first stem P of length four. Given P, with one application of Lemma 3, we can 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 of three edges each (adjacent blocks share a vertex), and apply Lemma 3 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, then there is an S-detour of length at most 2ℓ, and hence a T-path of

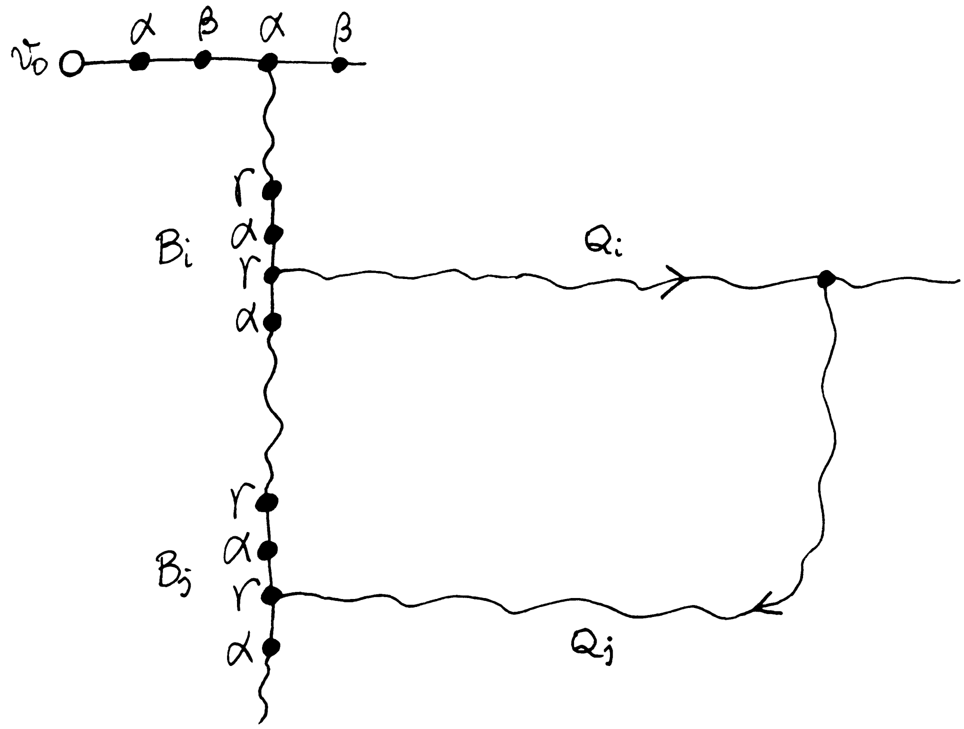

Figure 5: Q_i and Q_j cannot intersect

length at most $3\ell + 4$ (see Figure 5). Any Q_i which is not divergent from the stem S yields a T-path of length at most $2\ell + 4$. On the other hand, if all Q_i's are divergent from S, then there must be a pair of ℓ-branches Q_j and Q_k that intersect each other, since we have generated at least $\ell^2/3 + 4$ vertices.

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 3. When a new ℓ-branch Q_i is spawned off, we either get a T-path if Q_i intersects the existing tree (this happens if Q_i intersects other ℓ-branches or is not divergent from the stem) or we generate ℓ *brand new* vertices. Clearly, after $O(\sqrt{n})$ spawning operations, no new vertices can be included, and the ℓ-branch must intersect the existing tree.

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, and hence show the existence of an $O(\log_\Delta n)$ length T-path.

Theorem 1 *Let G be a partially Δ-colored nice graph with one pebbled vertex, and let G be Δ-regular within radius 3ℓ, where $\ell = 7\log_{2\Delta-4} n + 11$. Then, G has a T-path of length at most 3ℓ.*

PROOF. We show how to generate a tree of depth $O(\log_\Delta n)$ which covers all the vertices of G, unless a T-path of length $O(\log_\Delta n)$ was found. We want to 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. The idea is that if $P = v_0, v_1, \ldots, v_p$ is a stem and if P' is a bichromatic path spawning off from v_i in P, then $P'' = v_0, \ldots, v_i \bullet P'$ is a new stem. The new stem can be used to generate more stems, and so on.

A first stem P of length 4 can be generated by Lemma 4. From P we generate a bichromatic path $P' = w_0, w_1 \ldots w_7$ of length 7 where $w_0 \in P$, and subdivide it into two blocks B_1 and B_2, where $B_1 = w_1, w_2, w_3, w_4$ and $B_2 = w_4, w_5, w_6, w_7$. Call B_1 and B_2 *adjacent.* Note that

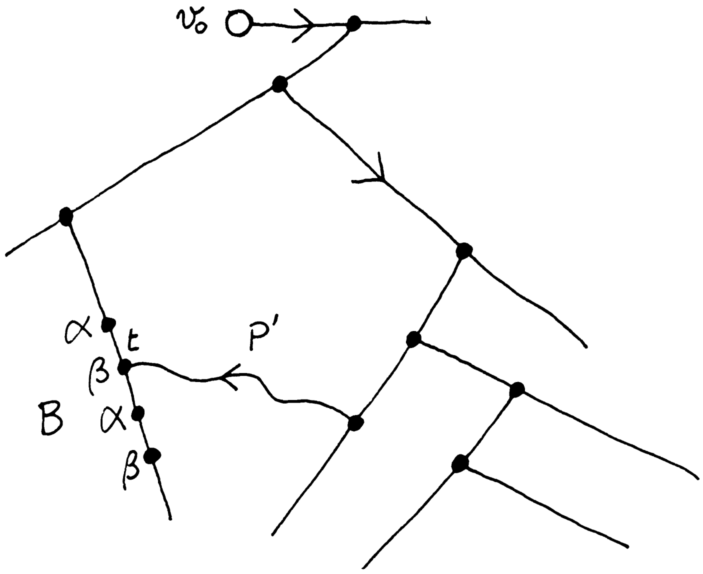

Figure 6: New bichromatic paths are divergent from the old tree

$P' = w_0, w_1 \bullet B_1 \bullet B_2$. The stem P with the 7-branch P' attached to it, forms the first tree T_0. Each block B_i has two *internal points* from which new paths can be spawned off. These points are w_2 and w_3 for B_1 and w_5 and w_6 for B_2. From Corollary 1, we can use each B_i 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. Assuming that the depth of T_k is at most $\ell = 7\log_{2\Delta-4} n + 11$, we will show that the depth of T_{k+1} is at most ℓ. We want to argue that T_{k+1} is a tree, that is that the new paths do not intersect each other and do not intersect the branches of the old tree T_k, or else we can find a T-path of length at most 3ℓ.

Let B be an unused block of T_k, x and y be its internal points with $\chi(x) = \alpha \neq \beta = \chi(y)$, and P be the stem B belongs to. Some of the new $\Delta - 2$ paths might originate from x and some from y; an internal point which is the origin of a new path is called a *turning point*. Let P' be one of the new paths, and let t be its turning point; by Lemma 3, P' and the stem P are divergent, or there is a T-path of length at most 3ℓ. Also, P' cannot intersect other branches of T_k or there is a detour to t and hence a T-path of length at most $\ell + 7$ (see Figure 6).

Now, consider an (α, γ_i)-path P_{α,γ_i} and an (α, γ_j)-path P_{α,γ_j} originating from, say, vertex x. If they meet, then they must meet at some vertex colored α, which cannot be the last vertex of P_{α,γ_i} or P_{α,γ_j}, since both of these last vertices are at a distance of 7, an odd number, from x along their respective paths, and hence will be colored γ_i and γ_j respectively. Hence, if P_{α,γ_i} and P_{α,γ_j} meet, they do so at an internal vertex of each of these paths, and hence we find a T-path of length at most $\ell+7$ (see Figure 7). An (α, γ_i)-path originating at x and a (β, γ_j)-path originating at y cannot intersect because $\{\alpha, \gamma_i\} \cap \{\beta, \gamma_j\} = \emptyset$. New paths from different blocks cannot intersect, or else there is a path between two turning points t_1 and t_2. In turn, this gives a detour between the least common ancestor of t_1 and t_2 (which is a vertex of T_k) and either t_1 and t_2. So, T_{k+1} is indeed a tree. If we collapse each pair of adjacent used blocks in T_{k+1} 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, the depth of T_{k+1} is at most ℓ.

The process we have described has the property that when T_{k+1} is generated from T_k, either a T-path is found or the branches that we spawn off only include new vertices. Clearly, after at most ℓ iterations (*i.e.*, when we generate $T_{\ell+1}$ from T_ℓ) the new branches must intersect the old tree because the whole graph is covered. □

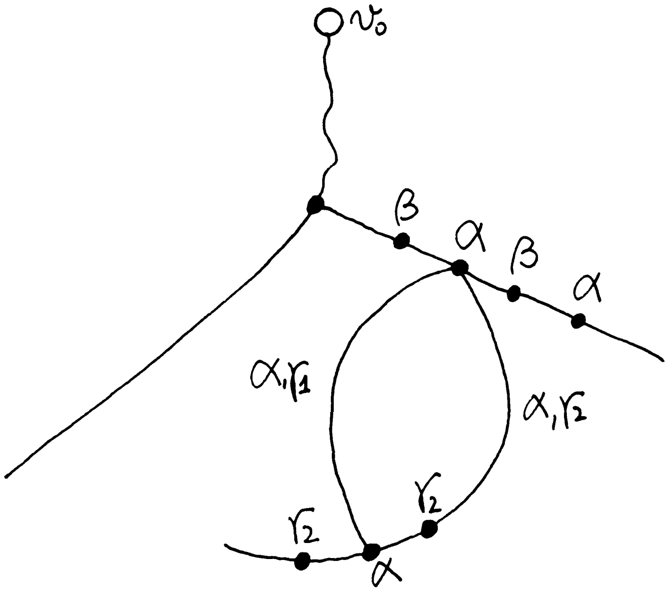

Figure 7: Bichromatic paths cannot meet

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 $3\ell = 21\log_{2\Delta-4} n + 33$ 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, and a partial Δ–coloring of T such that there is a pebble at the root, and 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 $(\Delta + 1)$–coloring which can be implemented distributively by an $O(\log^3 n/\log\Delta)$ expected time randomized algorithm or by an $O(\Delta\log^3 n/\log\Delta)$ time determin-

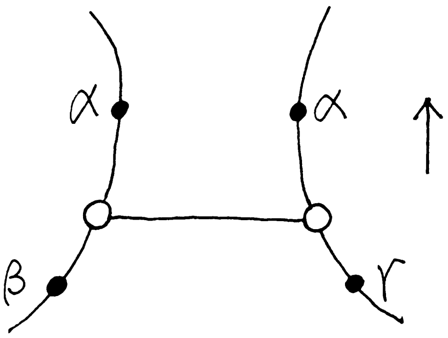

Figure 8: Steps must be scheduled

istic 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 $1,2,\ldots,\Delta,\perp$ and pebble all vertices with color $\perp$. 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 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:

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 5 *Let G be a partially colored nice graph, and let P denote the 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 legally.*

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. 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 $1 - 2^{-\Omega(f(n))}$.

Lemma 6 *Let G be a partially colored nice graph with n vertices, P be the set of pebbled vertices, and $\mathcal{F}$ be 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 $q(\cdot)$.*

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 modify 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 new 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 $E[h(\ell)] = (1-p)\ell$. Then, $T(1) = 1$ 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

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

Then by Theorem 3 of Karp [9], we see 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 is removed, because a pebble can walk at most $k \log n$ steps before being absorbed by the root pebble; so the total time taken is $O(k \log^2 n)$, with high probability. □

We summarize the whole algorithm now.

- Compute a $(\Delta + 1)$–coloring of G with colors $1, 2, \ldots, \Delta, \perp$. This can be done in $O(\log n)$ expected time distributively using a randomized algorithm of Luby [12].
- Pebble all the vertices with color $\perp$, 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. 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 5, 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 *If G is a nice graph, then it can be Δ-colored in the distributed model in $O(\log^3 n/\log\Delta)$ expected time.*

4.2 Deterministic Distributed Δ–coloring

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

In the previous algorithm, randomness was used in two places; to compute a $(\Delta+1)$–coloring and for the randomized reduction. The basic idea is that we can substitute the randomized procedure of Luby by an $O(\Delta\log n)$ time distributed algorithm for $(\Delta + 1)$-coloring, due to Goldberg, Plotkin & Shannon [5].

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. (Recall that 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.

- Deterministically compute a $(\Delta + 1)$–coloring with colors $1, 2, \ldots, \Delta, \perp$ in $O(\Delta \log n)$ time by using an algorithm of Goldberg, Plotkin & Shannon [5]. Pebble all vertices with color $\perp$.
- 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].
- 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.
- 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 *If G is a nice graph, it 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 consequences 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 $O(n^{\epsilon(n)})$ time algorithm for Δ–coloring in the distributed model, where $\epsilon(n) = O(1/\sqrt{\log n})$.

5.1 A linear processor NC algorithm

The randomized algorithm for Δ–coloring can be implemented and derandomized in the 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 operations performed at each node only require $O(1)$ time. Both these steps essentially require performing BFS searches for $O(\log^2 n/\log\Delta)$ and $O(\log n/\log\Delta)$ depth respectively.

To implement the randomized reduction we consider the following modification of Step 3. Let G be a partially Δ-colored graph with a set of pebbles P. 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 $1,\ldots,\Delta,\perp$. Let $C_\perp$ be the pebbles that got color $\perp$; all pebbles in $P - C_\perp$ have a legal color and $C_\perp$ is an independent set and hence, all pebbles in $C_\perp$ 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 $C_\perp$ 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 *A graph G can be Δ-colored in the* PRAM *model of computation with linearly many processors in $O(\log^5 n\log\log n/\log\Delta)$ time if and only if G is neither a clique nor an odd cycle.*

5.2 A Sublinear Time 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 $(n^{\epsilon(n)},n^{\epsilon(n)})$-decomposition distributively in $O(n^{\epsilon(n)})$ time where $\epsilon(n)=O(1/\sqrt{\log n})$ [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 $O(n^{\epsilon(n)})$, the total cost of this implementation is $O(n^{\epsilon(n)})$.

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, $d(n)=O(n^{\epsilon(n)})$ be an upper bound on the diameter of each cluster of the network decomposition, and let $c(n)=O(n^{\epsilon(n)})$ 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 $O(t(n)c(n)d(n)) = O(n^{\epsilon(n)})$ time, where $\epsilon(n) = O(1/\sqrt{\log n})$.

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 *A nice graph G is Δ-colorable in $O(n^{\epsilon(n)})$ time in the distributed model of computation, where $\epsilon(n) = O(1/\sqrt{\log n})$.*

6 Lower Bounds for some Distributed Coloring Problems

In this section, we prove an $\Omega(diameter(G))$ 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 $\Delta = 2$, coloring the edges is the same as coloring the vertices. 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. In this case, two-coloring the edges is the same as two-coloring the vertices. We will describe our lower-bounds in terms of vertex coloring.

Theorem 6 *Let $t(n) = o(n)$, and G be a connected graph with $\Delta = 2$. Then, there is no distributed protocol that computes a two-coloring of the vertices of G 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 of vertex i (i is the ID of the vertex) at time t. From the definition of our computation model, it follows that for any path-coloring protocol,

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

where i_L and i_R are the ID's of the neighbors of i. Also, $s(i,0)$ is the same for all vertices i. 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 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 P is used. But from the state transition function it follows that

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

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 P. □

6.2 Optimal Edge Coloring of Bipartite Graphs

In this subsection, we prove the same lower bound of $\Omega(diameter(G))$ 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 not outside this neighborhood.

Theorem 7 *For any $\Delta \ge 2$, there is no subdiametric time deterministic protocol for edge coloring an arbitrary graph of maximum degree Δ with Δ colors, 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.

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

If we connect Δ-widgets in a chain-like manner so that the e vertex of one widget coincides with the a vertex of the next widget, we are going to have a degree Δ bipartite graph.

We are now ready to prove our claim. 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. Let W_i be the ith-widget starting from the left, and let a_i, b_{ij}, c_{ij}, d_i be its vertices. Without loss of generality, assume

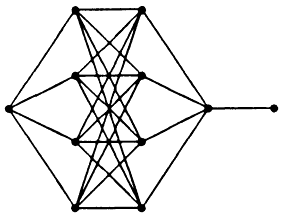

Figure 9: A 5-widget

that all edges (d_i, a_{i+1}) are colored with color Δ. We now modify the chain A by removing any vertex v from the last widget W_{3t+3} from the left and inserting v between d_t and a_{t+1}; we have 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 $\Omega(diam(G))$ 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 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 $o(n)$.*

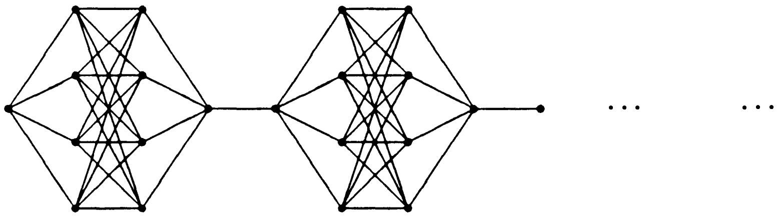

Figure 10: A chain of widgets

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 such that $n/4 > 2t$ (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 the desired probability. □

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 distributed protocol that finds a Δ-edge coloring of an arbitrary degree Δ bipartite graph in subdiametric time, nor is there a Las Vegas distributed protocol that finds a Δ-edge coloring of an arbitrary degree Δ bipartite graph in subdiametric expected 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 and locality in distributed computation. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 364–369, 1989.

[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.*, 37:194–197, 1941.

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

[5] A. V. Goldberg, S. A. Plotkin, and G. E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM J. Disc. Math.*, 1:434–446, 1989.

[6] P. Hajnal and E. Szemerédi. Brooks coloring in parallel. *SIAM J. Disc. Math*, 3(1):74–80, 1990.

[7] M. Karchmer and J. Naor. A fast parallel algorithm to color a graph with Δ colors. *Journal of Algorithms*, 9:83–91, 1988.

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

[9] R. M. Karp. Probabilistic recurrence relations. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 190–197, 1991.

[10] G. F. Lev, N. Pippenger, and L. G. Valiant. A fast parallel algorithm for routing in permutation networks. *IEEE Transactions on Computers*, 30:93–100, 1981.

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

[12] M. Luby. Removing randomness in parallel computation without a processor penalty. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 162–173, 1988. To appear in a special issue of *Journal of Computer and System Sciences*, devoted to FOCS 1988.

[13] A. Panconesi and A. Srinivasan. Improved distributed algorithms for coloring and network decomposition problems. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 581–592, 1992.