Locality in Distributed Computing

Alessandro Panconesi
Ph.D Thesis

93-1356
June 1993

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

LOCALITY IN DISTRIBUTED COMPUTING

A Dissertation
Presented to the Faculty of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by

Alessandro Panconesi

August 1993

LOCALITY IN DISTRIBUTED COMPUTING

Alessandro Panconesi, Ph.D.

Cornell University 1993

The topic of this thesis is the issue of locality in distributed computing.

A first set of results in this thesis concerns the Δ–vertex coloring problem. They are all based on the following result. Let G be a graph such that $\Delta \geq 3$, G is not a complete graph, and such that all of G except one vertex v is Δ–colored. Then, it is possible to extend this Δ–coloring to all of G by recoloring a path originating from v of length at most $O(\log_\Delta n)$.

This property allows us to develop several efficient algorithms for Δ–coloring. In particular, but not exclusively, in the distributed and **PRAM** models of computation. It also implies a well-known result of Brooks as a corollary.

Another set of results concerns network decomposition– a basic notion in distributed graph algorithms. We improve the bounds for computing a network decomposition distributively and deterministically. Our algorithm computes an $(n^{\epsilon(n)}, n^{\epsilon(n)})$–decomposition in $O(n^{\epsilon(n)})$ time, where $\epsilon(n) = O(1/\sqrt{\log n})$.

We also show that the class of graphs $\mathcal{G}$ whose maximum degree is $O(n^{\delta(n)})$, where $\delta(n) = O(1/\log\log n)$, is complete for the task of computing a $(\log n, \log n)$-decomposition, in polylogarithmic in n time. Completeness is to be intended in

the following sense: if we have an algorithm $\mathcal{A}$ that computes a $(\log n, \log n)$-decomposition in polylogarithmic in n time for graphs in $\mathcal{G}$, then we can compute a $(\log n, \log n)$-decomposition in polylogarithmic in n time for all graphs.

A last set of results concerns the edge coloring problem. We give a randomized, distributed algorithm to compute an edge coloring of a given network G, that uses at most $1.6\Delta + \log^{2+\delta} n$ colors, for any $\delta > 0$. The running time is $O(\log n)$ and the failure probability is at most ϵ, for any fixed $\epsilon > 0$. The algorithm is quite simple but requires an interesting probabilistic analysis. At the core of the analysis is an extension of the Chernoff-Hoeffding bounds, which are fundamental tools used in estimating the tail probabilities of the sum of Bernoulli-like trials.

Biographical Sketch

Alessandro Panconesi was born in Roma, Italy, on May 20, 1960. *" Godel's Proof"*, a delightful introductory booklet by Ernest Nagel & James R. Newman, had a great impact on his life; fascinated by the beauty and the grand architectural sweep of those magnificent ideas, he decided to study Mathematics at the University of Rome *La Sapienza*, where he got his Laurea degree in the Fall of 1984. Mathematics proved too esoteric and difficult and he thought that Theoretical Computer Science was a good source of more tractable and yet meaningful mathematical problems.

He remained out of school for a couple of years before going back to the academic world. Thanks to a fellowship of the Italian Ministry of Education he had the wonderful opportunity to visit the renowned Computer Science Department of Cornell University for a year.

He was quite impressed by the department and decided to go for a PhD. The department was graceful enough to accept him in the program.

As for student... life in Ithaca, the following poem of Charles Baudelaire accurately expresses his feelings.

Spleen IV[1]

Quand le ciel bas et lourd pèse comme un couvercle
Sur l'esprit gémissant en proie aux longs ennuis,
Et que de l'horizon embrassant tout le cercle
Il nous verse un jour noir plus triste que les nuits;

Quand la terre est changée en un chacot humide,
Oú l'Espérance, comme une chauve-souris,
S'en va battant les murs de son aile timide,
Et se cognant la tête à des plafonds pourris;

Quand la pluie étalant ses immenses traînées
D'une vaste prison imite les barreaux,
Et qu'un peuple muet d'infâmes araignées
Vient tendre ses filets au fond de nos cerveaux,

Des cloches tout á coup sautent avec furie

[1]When the low, leaden sky presses down like a lid on the suffering spirit in endless travail, and when from around the horizon's whole rim it pours on us dark daylight more somber than night;

When the earth is turned into a dank dungeon cell wherein Hope, like a fluttering bat in the gloom, in vain beats the walls with a timorous wing and bruises her head on ceiling's foul slime;

When rain, hanging out its immense sweeping trains, resembles a vast, murky prison's thick bars, and a host of vile spiders soundlessly come to spin their damp webs in the depths of our brains,

Then bells, of a sudden, explode in a rage and fling toward the heavens a dolorous cry, like wandering spirits in search of a home, beginning an obstinate whimpering plaint.

–And in silent cortège, without music or drums, long black hearses dead-slowly defile in my soul; Hope, vanquished, weeps; and fierce Anguish, despotic, upon my bowed skull firmly plants her black flag.

Et lancent vers le ciel un affreux hurlement,
Ainsi que des esprits errants et sans patrie
Qui se mettent à geindre opiniâtrement.

–Et de longs corbillards, sans tambours ni musique,
Défilent lentement dans mon âme; l'Espoir,
Vaincu, pleure, et l'Angoisse atroce, despotique,
Sur mon crâne incliné plante son drapeau noir.

In September 1989 he was awarded the title of Dottore di Ricerca by the Italian Ministry of Education.

In August 1990 he was awarded a Master of Science by Cornell University.

On May 25, 1993, he successfully defended his Ph.D. thesis at Cornell University.

To my Family: Mother, Father, and Sister Silvia.

To my friend Corrado,

and to my friend Julia,

whose smile makes the Sun jealous.

Acknowledgements

My advisor is like an Eagle: steady and majestic, he soars the sky with elegance covering great distances with minimum effort, and constantly patrolling the earth with his sharp eyes.

His students are sometimes birds; one day they will fly with poise, able to control their speed. Sometimes, as in my case, they are explorers, clumsy bipeds bound to the ground. They will never fly. The Eagle is the companion of the explorer and escorts him on a voyage through an unknown land.

The sight of the Eagle is comforting: his motion is at the same time precise and elegant, his speed paired with remarkable acceleration, his vision allows him to see from afar both the general patterns and the minute details. Sometimes the Eagle sees that what seemed from the ground a promising path is in fact leading to a featureless desert. Sometimes he sees that a few scattered bushes soon become an inextricable, impenetrable forest. Some other times, thanks to his sharp sight, he sees a trail comfortably lending to the other side of an otherwise awesome and inaccessible mountain range. At times, he gives a beautiful gift to the explorer; gradually, step by step, he brings him to the top of a small mountain from where he can appreciate for a fleeting moment what, for the bird, is the common sight of the earth seen from the sky.

It is the explorer, however, who finally draws the detailed map of the voyage.

I wish to express my sincere thanks to my advisor, Professor David Shmoys. Besides his wide mathematical knowledge, and a taste for mathematics which is at the same time meaningful and elegant, I appreciated and benefitted from his rare combination of scholarly integrity and academic pragmatism. I also thank him for his remarkable generosity and for his silent but deep understanding of my needs, especially those outside of academia. During my thesis research I was supported by his NSF PYI award CCR-89-96272 with matching funds from UPS and Sun Microsystems.

My sincere thanks also go to Professor Juris Hartmanis for serving on my committee. I came to Cornell attracted by his fame and I was not in the slightest disappointed. Besides being an extremely kind and cheerful man, and a great and enjoyable teacher, he is also an excellent example of what a scientist should, or at least could, be. I always admired the aesthetic quality of his contributions: simple and meaningful–the trademark of good science.

Thanks to Professor Frank Keil, of the Department of Psychology, for serving on my committee and guiding me through my minor studies.

Thanks to Eva Tardos for generously supporting me during my last semester with funds from her Packard Foundation Fellowship.

Thanks to Sam Toueg for his kind help and encouragement.

My thanks and respect go to Gianfranco Bilardi who has been, and will continue to be, a great model of integrity and academic excellence. By talking to him I started to appreciate the role and responsibility of the intellectual, and the difficulties that they entail. I gratefully acknowledge his guidance and help.

Perhaps the most important and certainly the most enjoyable part of my graduate studies was my collaboration with other students. My heartfelt thanks go to Aravind Srinivasan; the results in this thesis are joint work with him. Aravind is a bird. It has always been a pleasure and a source of amazement to watch his acrobatic flight. Soon he will master his technique and build strength for his wings, to make his flight not only acrobatic but seamless.

My collaboration with Desh Ranjan was also important and enjoyable. My only regret is that we did not have more opportunities to work together.

Other students in the department were important for my scientific experience or were just nice and made my life as a graduate student more bearable: Devdatt Dubhashi, James Allan, Jim Caldwell, Lorenzo Alvisi, Prasad Jayanti, Captain Sofoklis Efremidis, and Suresh Chari.

The support staff of the Computer Science Department of Cornell University is absolutely outstanding. They remind me of cheerful giants, moving heavy stumbling blocks with ease. Their genuine smiles were always refreshing and their readiness to help students, truly incredible. My warm thanks go to all of them, but especially to Jan Batzer and Carol Ayer.

My warm thanks go to Sridhar Vembu for his wonderful hospitality when I was in Princeton, and to Sharon Rodgers of the Computer Science Department of Princeton University for her great help, remarkable generosity, and very enjoyable company during my semester at Princeton.

Finally, my respectful thanks go to my Family.

Table of Contents

List of Figures

Chapter 1

Locality in Distributed Computing

The topic of this thesis is the issue of locality in distributed computing, which is, roughly speaking, as follows. In a distributed network the bottleneck of any computation is due to communication, *i.e.*, sending messages from one processor to another. To minimize cost it is necessary that each processor not send messages to processors that are far away in the network. Computation should take place based on local information alone, *i.e.*, a processor should communicate only with nearby processors. The question is how efficiently a given function can be computed given this locality constraint.

In order to formulate this question in a mathematically precise way we introduce the following simple model of distributed computation, which has been extensively used in the literature. A *message-passing distributed network* is an undirected graph $G = (V, E)$ where vertices, or nodes, correspond to processors and edges correspond to bi-directional communication links. Each node has its unique ID. The network is *synchronous*, *i.e.*, computation takes place in a sequence of *rounds*; in each round, each node reads messages sent to it by its neighbors in the graph, does any amount of local computation, and sends messages back to all of its neighbors. The time complexity of a distributed algorithm, or *protocol*, is given by the number of rounds needed to compute a given function.

In spite of the totally unrealistic assumption that a node is allowed to perform any amount of internal computation in one round, this model captures some essential aspects of a distributed computation. In particular, it captures nicely the issue of locality. A more detailed discussion of the advantages and disadvantages of this model is postponed to later in this chapter.

Notice that in this model the cost of sending a message from one vertex to another is proportional to the shortest path between the two vertices, that is, to the minimum number of edges of any path between them. There is no shared memory and processors can communicate only by sending messages through the network. Hence, if we want a protocol to run for t rounds, then each vertex can communicate only with vertices at distance at most t from it. This is not so in the PRAM model, where the shared memory allows any two processors to communicate in one unit of time.

For example, suppose that a network with n processors has diameter $\Omega(n)$ and that we want the time complexity of our protocol to be $O(\log n)$; then every vertex can communicate with only a small neighborhood of vertices of $O(\log n)$ radius. Yet the network is computing a global function of its data, initially spread across the whole network.

In this thesis, we study the complexity of several basic graph theoretic problems in this model. The general problem is of the following type: a network G is to compute some function of its own topology such as, for example, a maximal independent set or a vertex coloring. In particular, we are interested in the vertex coloring, edge coloring, maximal independent set (MIS), and network decomposition problems.

An *independent set* in a graph $G = (V, E)$ is a set of vertices $I \subseteq V$ satisfying the following independence property: if u and v are vertices in I then there is no edge between them. An independent set is *maximal* if no vertex can be added to it without violating the independence property. A *vertex coloring* of G is a partition of V into independent sets, called *colors*. It is well known that coloring a graph with the minimum possible number of colors is an NP-hard function [GJ79]. A good coloring is often expressed in terms of the maximum degree of G, *i.e.*, the maximum number of edges incident on any node, which is denoted by Δ. Colorings using Δ or $(\Delta + 1)$ colors are considered good, especially in a parallel or distributed setting. A *matching* in a network G is a set of edges such that any two of them share no endpoints. An *edge coloring* is a partition of the edges of G into a collection of matchings, called colors. Network decomposition is a particular kind of graph decomposition that was introduced to study basic graph problems in the distributed model of computation [AGLP89]. It will be defined later in this chapter.

The reason for studying these problems is twofold. On one hand, they seem to be useful in the context of synchronization and resource allocation problems. For example, vertex and edge colorings are used to solve classical problems like the *Dining Philosophers Problem* of Dijkstra [CS92,Lyn81,SP88], and computing a maximal independent set is an important primitive in many parallel algorithms [KB87,KN88,Kar89,Kar86,KS87,KW85,Lub86,Nao91a]. More examples are given in the introductory sections of forthcoming chapters. On the other hand, studying these problems in the PRAM model has proven extremely fruitful because it initiated the development of theoretical tools of wide applicability—*i.e.*, the derandomization technique of conditional probabilities and the theory of small probability spaces [AGHP92,BR91,Lub86,Lub88,MNN89,NN90].

A useful computational resource in distributed and parallel computing is randomization. Randomization seems useful not only as a resource but also, perhaps surprisingly, as a methodology.

The methodological aspects of probabilistic techniques applied to parallel computation are illustrated by the derandomization technique of conditional probabilities and the theory of small probability spaces. The method of conditional probabilities, implicitly introduced by Erdős & Selfridge and systematized by Spencer [ES73,Spe87], is a powerful mathematical technique to give constructive proofs of the existence of certain combinatorial objects. A probabilistic analysis establishes the existence of the desired object, followed by a deterministic process that actually constructs it. In many instances this process can be carried out computationally in a cost-effective way, for example by means of a polynomial-time or an NC algorithm [Rag90,BR91,Lub86,Lub88,MNN89].

The theory of small probability spaces is another computational offspring of the probabilistic method [ASE92]. In the probabilistic method one shows the existence of a combinatorial object by means of a probabilistic analysis. Typically, one shows that the probability of existence of the object is non zero. It has been noted that in some cases the probability space used in the analysis can be replaced by a smaller one. The new space either maintains all the properties that are relevant for the analysis or approximates the original one. The existence of the combinatorial object is still ensured under the new conditions. If the new space is small enough, *e.g.*, of polynomial size, then it can be searched exhaustively in parallel, that is, all possible random outcomes are tried in parallel. The randomized process is thereby replaced with a deterministic one [KW85,Lub86,Lub88,AGHP92,NN90].

In both cases, the final algorithm is deterministic. Randomization is not used as a computational resource but as an artifact in the analysis. The tremendous success of these techniques is underscored by the puzzling fact that the only deterministic algorithms known to date for certain problems are obtained through derandomization techniques [KS87,Lub88]. For the maximal independent set problem such algorithms were the only available ones for a long time [GS,KW85,Lub86]

It is an important open problem, beyond the scope of this thesis, to determine whether derandomization techniques are possible in a distributed environment.

As a resource, randomization is very powerful in dealing with *symmetry–breaking*, which is one of the essential problems encountered in the design of distributed and parallel algorithms. Generally speaking, the problem is that of choosing one among several indistinguishable alternatives during a parallel computation. To clarify what symmetry–breaking is let us take an extreme example. Consider n processors connected together via an *anonymous ring*, *i.e.*, processors are connected in a ring topology but have no ID's. Suppose also that the ring is to compute a maximal independent set S of itself. Straightforward symmetry arguments show that no deterministic algorithm is possible; either all processors decide to be in S or they all decide to be outside of S.

But if we are allowed to use randomization then the problem becomes easy. For example, each processor can choose a number in the range $[1, n^3]$ uniformly at random and independently of other processors. Simple calculations show that with probability at least $(1 - 1/n^4)$ all the ID's will be different. Then, for example, we can run one of the existing deterministic algorithms based on the assumption that processors have different ID's and compute the desired maximal independent set [CV86,GPS89,Lub86].

Intuitively, the additional constraint imposed by locality makes the problem of symmetry-breaking only worse. The central open problem concerning locality is whether randomization helps. The current state of affairs is that there are very efficient randomized algorithms for many basic combinatorial problems (usually related to scheduling, synchronization, and resource allocation), whereas it is not known if deterministic solutions with comparable running time exist. For example, for the problems maximal independent set, maximal matching, $(\Delta + 1)$–vertex coloring, approximate edge coloring, and network decomposition there are (usually very simple) randomized, distributed algorithms whose expected running time is poly–logarithmic in the size of the network [ABI86,II86,LS91,Lub86,Lub88,PS92b, PS92a]. On the other hand, the best deterministic upper–bound for the same problems in the distributed model is $O(n^{\epsilon(n)})$, where $\epsilon(n)$ tends to zero as n goes to infinity [AGLP89,PS92b]. This is better than any root of n but asymptotically much worse than any poly-logarithmic function of n. Attempts thus far at finding deterministic algorithms have failed but no logarithmic lower bound proof is available either.

One could hope to develop symmetry-breaking techniques based on the fact that nodes have different ID's. Cole & Vishkin developed a technique, known as deterministic coin-tossing, which has several interesting and non-trivial applications [AGLP89,CV86,GPS89], but in general, randomization is the only way to deal with symmetry–breaking known so far.

Whether randomization helps in circumventing the locality bottleneck is one instance of a general and fundamental problem in computer science: to what extent, if at all, the use of randomization helps. The most general theoretical questions, such as $P \stackrel{?}{=} BPP$ or $P \stackrel{?}{=} RNC$, are still unanswered, but in more limited domains some answers are available. In some cases randomized algorithms can be proven to be better than any possible deterministic algorithm (*e.g.* oblivious routing in parallel architectures [Lei92,Rag90]), in other cases randomized algorithms provide a solution when a deterministic solution is not even possible (*e.g.* consensus in asynchronous networks [BO83,FLP85]).

In view of the limited power of the distributed model, finding an answer to this problem with regard to locality, though a difficult task, does not appear impossible. Some very limited partial answers are already available.

Linial has shown that if n processors are connected in a ring topology then $\Omega(\log^* n)$ is a lower–bound for computing a maximal independent set and a 3–vertex coloring [Lin92]. Interestingly, this matches the upper–bound given by a beautiful algorithm of Cole & Vishkin [CV86]. Naor has shown that in this case randomization does not help [Nao91b].

In some cases, it is quite easy to obtain $\Omega(n)$ lower–bounds; for example, in this thesis we show that if a network is a bipartite graph with n processors then $\Omega(n)$ is a lower–bound for computing a Δ–edge coloring, whereas such a coloring can be computed in NC [LPV81]. We also show that in this case randomization does not help [PS92b,PS92c].

Naor & Stockmeyer have investigated the question of what can be computed in constant time, with and without randomization, and give a series of upper and lower–bounds for some simple kinds of colorings [NS92]. Again, they were not able to find examples of functions for which randomization helps. Randomization also does not help for special topologies like rooted trees or constant degree graphs [GPS89], and planar graphs [Pan].

In order to understand whether randomization helps with regard to locality, network decomposition, a notion introduced by Awerbuch, Goldberg, Luby & Plotkin [AGLP89], plays a central role. Given a graph $G = (V, E)$, a partition of V into a collection of *clusters* is a set $\mathcal{C} = \{C_i\}$ such that $V = \cup C_i$ and $C_i \cap C_j = \emptyset$ if $i \neq j$. The *cluster graph* $G_\mathcal{C}$ induced by $\mathcal{C}$ is the graph with vertex set $V(G_C) = \mathcal{C}$ and edge set

$$E(G_\mathcal{C}) = \{(C_i, C_j) : \ i, j \text{ distinct}, \exists u \in C_i, \exists v \in C_j \ (u, v) \in E\}.$$

A $(d(n), c(n))$–network decomposition of G is a set of clusters $\mathcal{C}$ with a vertex coloring of $G_\mathcal{C}$ with $O(c(n))$ colors, such that every $G[C_i]$, the subgraph induced by vertices in cluster C_i, is connected and has diameter $O(d(n))$. The requirement that $G[C_i]$ is connected can be relaxed, see for example [LS91].

A decomposition with $O(c(n)) = O(d(n)) = O(\log n)$ is called *near-optimal*, since Linial & Saks have exhibited families of graphs for which $c(n) + d(n) \geq \Omega(\frac{\log n}{\log \log n})$ and $c(n)d(n) \geq \Omega(\log n)$ for any $(d(n), c(n))$–decomposition [LS91]. A priori, it is not clear that a near-optimal decomposition should exist at all. Using a linear time algorithm of Awerbuch & Peleg, Linial & Saks observe that a near-optimal decomposition exists for any graph G, and give an $O(\log^2 n)$ expected time randomized distributed algorithm for this [AP92,LS91].

The importance of network decomposition lies in the fact that given a (good) decomposition it is possible to compute other structures efficiently, *e.g.* matchings, independent sets, vertex and edge colorings. More precisely, given a $(d(n), c(n))$–network decomposition we can compute other graph structures, say a maximal independent set, in $O(d(n)c(n))$ time. The generic algorithm for such problems, given a cluster 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 trivial algorithm can be used: the leader of the cluster, say the processor with highest ID, collects complete information on the topology of the cluster, internally computes a solution, and sends the answer back to all vertices in the cluster. (This solution is mainly of theoretical interest because it might involve large message size and significant internal computation. In some cases, however, it can be useful.) The bounds on the cluster diameter and the number of colors used, yield the bound on the time complexity of this generic algorithm.

Hence, given a near-optimal decomposition, other graph structures like maximal independent sets, maximal matchings, and $(\Delta + 1)$–vertex colorings can be computed deterministically in poly-logarithmic time. In other words, near-optimal decomposition is complete for a reasonably large class of problems. This class includes maximal independent set, Δ–vertex coloring, maximal matching, minimal edge cover, and many others. So, if a near-optimal decomposition can be computed in polylogarithmic time deterministically, then randomization does not "help" in computing all these various graph structures. For this reason, it is a major open problem in this area to determine whether a near-optimal decomposition can be computed deterministically in the distributed model of computation in $O(polylog(n))$ time, *i.e.*, in polylogarithmic in n time.

Let us return to the issue of locality. Some graph structures trivially have a locality property. Consider for example the $(\Delta+1)$-vertex coloring problem: given a graph $G = (V, E)$ we want to color the vertices of G using $(\Delta + 1)$ colors, in such a way that adjacent vertices always have different colors. Δ, we recall, is the maximum degree of the network, *i.e.*, the maximum number of edges incident on any node. In a sequential setting, this problem is trivial: pick one vertex v, color it with a color not chosen by any of its neighbors, and repeat. We can decide the color of a vertex u just by looking at its immediate neighbors, *i.e.,* a local search of radius 1. But other problems do not seem, a priori, to satisfy this locality requirement. For example, if the network is to compute a Δ–vertex coloring then we might face a situation where an uncolored vertex of degree Δ does not have a free color readily available, since all Δ colors are being used by its Δ neighbors. In order to assign a valid color we then might be forced to recolor a large part of the network.

A first set of results in this thesis concerns the Δ–vertex coloring problem. They are all based on the following result on the "local" nature of Δ–vertex colorings, that we call the Small Radius Search Theorem. Let G be a graph such that $\Delta \geq 3$, G is not a complete graph, and such that all of G except one vertex v is Δ–colored. Then, it is possible to extend this Δ–vertex coloring to all of G by recoloring a path originating from v of length at most $O(\log_\Delta n)$ [PS92b,PS92c].

This non-trivial "locality" property of Δ–vertex colorings allows us to develop several efficient algorithms for Δ–vertex coloring [PS92b,PS92c]. In particular, but not exclusively, in the distributed and parallel (PRAM) models of computation. It also implies the following well-known results of Brooks as a corollary [Bol79, Bro41]: a graph G can be Δ colored if and only if it is neither an odd cycle nor a complete graph. A more detailed list of all the results stemming from the Small Radius Search Theorem are given in the introduction to Chapter 3.

Another set of results concerns network decomposition, whose importance we have already discussed. In this thesis, we give a deterministic, distributed algorithm for computing a $(n^{\epsilon(n)}, n^{\epsilon(n)})$–decomposition in $O(n^{\epsilon(n)})$ time, where $\epsilon(n) =$

$1/\sqrt{\log n}$. This is a slight improvement over the previously known best upper–bound, for which $\epsilon(n) = \sqrt{\log\log n}/\sqrt{\log n}$ [AGLP89,PS92b].

We also show that the class of graphs $\mathcal{G}$ whose maximum degree is $\Delta = O(n^{\delta(n)})$, where $\delta(n) = O(1/\log\log n)$, is complete for the task of computing a near-optimal decomposition in $O(polylog(n))$ time. What we mean is that if we have an algorithm $\mathcal{A}$ that computes a near-optimal decomposition in $O(polylog(n))$ time for graphs in $\mathcal{G}$ then we can compute a near-optimal decomposition in $O(polylog(n))$ time for all graphs. This is actually a corollary of a more general characterization, see Chapter 2.

A last set of results concerns the edge coloring problem. We give a randomized, distributed algorithm to compute an edge coloring of a given network G, that uses at most $1.6\Delta + \log^{2+\delta} n$ colors, for any $\delta > 0$. The running time is $O(\log n)$ and the failure probability is at most ϵ, for any fixed $\epsilon > 0$. The algorithm is quite simple but requires an interesting probabilistic analysis. At the core of the analysis is an extension of the Chernoff-Hoeffding bounds, which are fundamental tools used in estimating the tail probabilities of the sum of Bernoulli-like trials [Che52,Hoe63,Sri93,Rag90]. Let $X = \sum_{i\in[n]} X_i$ be the sum of n *independent* 0-1 random variables, such that $E[X] = \mu$. Roughly speaking, these bounds state that the probability of X being $(1+\delta)$-away from μ goes to zero exponentially fast, as n goes to infinity. More precisely, the probability

$$Pr(X > (1+\delta)\mu) \leq \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mu} \doteq F(\mu,\delta).$$

The hypothesis that the X_i's are independent is used crucially in the proof of these bounds. Our extension concerns a certain case of dependence among the X_i's that we call λ-correlation. The intuition behind this extension is as follows. Suppose we have a set of points S, and a family of n subsets $\mathcal{A} = \{A_i : A_i \subseteq S,\ i \in [n]\}$. Each point of S flips a coin independently of the other points. Let X_i be the indicator random variable of the event "All points in A_i come up head". Let $X = \sum_{i\in[n]} X_i$, and $E[X] = \mu$. Notice that X_i and X_j are independent if and only if A_i and A_j are disjoint. Moreover, if $A_i \cap A_j \neq \emptyset$, the conditional probability

$$Pr(X_i = 1 \mid X_j = 1) > Pr(X_i = 1)$$

which in turn implies that, for all nonempty $I \subseteq [n]$,

$$Pr\left(\bigwedge_{i\in I} X_i = 1\right) \geq \prod_{i\in I} Pr\left(X_i = 1\right)$$

It also implies that given that $X_1, \ldots, X_i$ are all 1 then it is more likely that X_{i+1} is also 1. The extent to which this is more likely depends on how the A_i's overlap. If the overlap is very significant, large deviations of X from its mean are quite likely. Suppose, on the other hand, that the X_i's are λ-*correlated*, that is, there exists λ such that, for all nonempty $I \subseteq [n]$,

$$Pr\left(\bigwedge_{i\in I} X_i = 1\right) \leq \lambda \prod_{i\in I} Pr\left(X_i = 1\right)$$

In this case, given that $X_1, \ldots, X_i$ are all 1 then the probability that X_{i+1} is also 1 is not too different from the unconditional probability, and large deviations from the mean are not likely. What we prove is that if the X_i's are λ-correlated random variables and X is the sum of the X_i's then

$$Pr(X > (1+\delta)\mu) \leq \lambda\ F(\mu, \delta).$$

This extension of the Chernoff-Hoeffding bounds has also other interesting applications, some of which are related to classical problems in combinatorics [Sri].

Let us return to the model of computation to analize the advantages and disadvantages of several simplifying assumptions. As we shall see, the model has the advantage of simplicity without loosing much in accuracy.

In a distributed network the cost (in terms of time) of sending one message to a neighbor is many orders of magnitude bigger than that of executing one elementary operation internally in one node. This justifies the approximation of our model that allows for any amount of internal computation to be executed in one round.

In the analysis of our algorithms we then have to make sure that only "reasonable" amounts of internal computation are executed. This is usually the case for the algorithms in this thesis, with the only possible exception, depending on the definition of "reasonable", of some simulations involving network decomposition. In general, it is quite easy to give an accurate estimate of the total running time of our algorithms, including internal computation, but such estimates are omitted in the thesis.

Another unrealistic assumption that is made in the model is that every processor is allowed to send messages to all of its neighbors in unit time; it would seem more reasonable to charge a cost proportional to the degree of the node. Again this simplifying assumption can be justified on the grounds of hardware considerations. Also, it allows us to avoid cumbersome details in the combinatorics. If needed, the appropriate costs incurred by our algorithms can be computed quite easily.

Another relevant quantity that it is not explicitly taken into account is message size. Given that we are only concerned with problems where a network $G = (V, E)$ is computing a function of its own topology, an obvious upper bound on the size of any message is $O(|V| + |E|)$. In general, again with the exception of simulations involving network decomposition, message size required by our algorithms is constant or $O(\log n)$. As before, the precise estimate of such cost is omitted but can be quite easily derived.

An important aspect of this model of distributed computation is that it can be simulated in the PRAM model by linearly many processors with a time overhead of $O(\text{MAXMESSAGESIZE})$, where MAXMESSAGESIZE is an upper bound on the size of any message. More precisely, if a network $G = (V, E)$ computes a function f in time $O(t(n))$, then the PRAM simulation takes $O(t(n)\text{MAXMESSAGESIZE})$ time, using $O(|V| + |E|)$ processors. So, if the message size is constant or poly-logarithmic, then a very efficient distributed algorithm translates into a very efficient PRAM algorithm. This is the case, for example, with the Δ–vertex coloring algorithm in Chapter 3.

We also made the assumption that each processor in the network has its own unique ID. As we briefly discussed, symmetry considerations show that in an anonymous network there is practically nothing that can be computed deterministically. One could also consider a more economical assignment of ID's, for example one where neighboring vertices have different ID's. But this would not change things in any essential way.

Finally, it should be noted that, as far as lower bounds are concerned, this model is very general; any lower bounds derived in this model directly holds in any model charging for other additional costs incurred during the computation.

Chapter 2

Improved Network Decomposition

Two important and inter–related problems in the distributed model are to compute a maximal independent set (MIS) and to compute a good vertex coloring of a network G, say a $(\Delta+1)$–coloring (Δ is the maximum degree of any vertex in G). An MIS defines a set of processors which can compute in parallel without interference, and a coloring is a partition of V into independent sets, thus defining a schedule for the processors to compute in parallel, without interfering with their neighbors. There are very simple distributed randomized algorithms of Alon, Babai & Itai and Luby that compute an MIS and a $(\Delta+1)$–coloring in $O(\log n)$ expected time [ABI86,Lub86,Lub88]. While these papers also show how to derandomize these algorithms in NC, it is an outstanding open question whether the derandomization can be carried out in the distributed model.

In order to solve the MIS and related problems in the distributed model, Awerbuch, Goldberg, Luby & Plotkin introduced the notion of *network decomposition* (sometimes also called cluster decomposition) [AGLP89]. Let us first review the definition. Given a network $G=(V,E)$ and a partition of V into a set of *clusters* $\mathcal{C}=\{C_i\}$, the *cluster graph* $G_{\mathcal{C}}$ induced by $\mathcal{C}$ is the graph with vertex set $V(G_{\mathcal{C}})$

and edge set

$$E(G_{\mathcal{C}}) = \{(C_i, C_j) : \ i \neq j, \ \exists u \in C_i \ \exists v \in C_j \ (u,v) \in E\}.$$

A $(d(n), c(n))$–decomposition of G is a partition of V into a set of clusters $\mathcal{C}$ such that

- every $G[C_i]$, the subgraph induced by vertices in C_i, is connected and of $O(d(n))$ diameter, and
- the cluster graph is vertex colored with $O(c(n))$ colors.

Problems like MIS and $(\Delta+1)$–coloring can be solved in $O(d(n) \cdot c(n))$ time, given a $(d(n), c(n))$-decomposition of G. The generic algorithm for such problems, given a cluster 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 trivial algorithm can be used: the leader of the cluster (say the processor with highest ID) collects complete information on the topology of the cluster, internally computes a solution, and sends the answer back to all vertices in the cluster[1]. The bounds on the cluster diameter and the number of colors used, yield the bound on the time complexity of this generic algorithm.

Network decomposition has also other interesting applications, mainly, but not exclusively, to distributed computing [Awe85,AGLP89,AP92,AP90,BC].

Linial & Saks showed how to compute a near-optimal decomposition (that is, one with $O(c(n)) = O(d(n)) = O(\log n)$) in $O(\log n)$ expected time by using randomization, although they relax the requirement that the $G[C_i]$'s be connected [LS91].

In this chapter, we (slightly) improve the bounds of Awerbuch, Goldberg, Luby & Plotkin for computing a network decomposition distributively and determinis-

[1]This solution is mainly of theoretical interest because it might involve large message size and significant internal computation.

tically. We show how to compute a $(n^{\epsilon(n)}, n^{\epsilon(n)})$–decomposition in $O(n^{\epsilon(n)})$ time, where $\epsilon(n) = O(1/\sqrt{\log n})$ (as opposed to their $\epsilon(n) = O(\sqrt{\log\log n}/\sqrt{\log n})$).

As a corollary of this improved network decomposition result we obtain improved deterministic bounds for computing an MIS, $(\Delta + 1)$–coloring, and other graph problems.

We also show that the class of graphs $\mathcal{G}$ whose maximum degree is $\Delta = O(n^{\delta(n)})$, where $\delta(n) = O(1/\log\log n)$, is complete for the task of computing a near-optimal decomposition in $O(polylog(n))$ time. This is actually a corollary of a more general characterization. Completeness is to be intended in the following sense: if we have an algorithm $\mathcal{A}$ that computes an near-optimal decomposition in $O(polylog(n))$ time for graphs in $\mathcal{G}$ then we can compute an near-optimal decomposition in $O(polylog(n))$ time for all graphs. This completeness result pinpoints precisely what graphs are the most difficult to handle and we feel that this would help future attempts at solving this problem.

2.1 Definitions

A *message–passing distributed network* is an undirected graph $G = (V, E)$ where vertices, or nodes, correspond to processors and edges to bi–directional communication links. Each processor has its unique ID. The network is *synchronous*, *i.e.*, computation takes place in a sequence of *rounds*; in each round, each processor reads messages sent to it by its neighbors in the graph, does any amount of local computation, and sends messages back to all of its neighbors. The time complexity of a distributed algorithm, or *protocol*, is given by the number of rounds needed to compute a given function.

The absence of a shared memory imposes a *locality* constraint in the following sense: 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. This model is hence suited for studying the complexity of a problem when communication is the bottleneck. In this model we do not charge for local computation; in one round each processor is allowed to compute any function of its current data. In our algorithms however, each processor will perform very simple computations. In particular, each step can be simulated in $O(\Delta)$ by a single processor or in constant time with Δ processors, where Δ denotes the maximum degree of any vertex in the network.

Given a graph $G = (V, E)$ and a set $S \subseteq V$, $G[S]$ denotes the subgraph induced by S. By $V(G)$ and $E(G)$ we denote the set of vertices of G and the set of edges of G respectively. The degree of a vertex v in a graph G is denoted by $\deg_G(v)$. The distance between two vertices u and v in a graph G, *i.e.*, the length of a shortest path connecting them in G, is denoted by $d_G(u, v)$.

Given a graph $G = (V, E)$ and a set $S \subseteq V$, we denote by $G[k, S]$ the graph whose vertex set is $V(G[k, S]) = S$ and whose edge set is

$$E(G[k, S]) = \{(u, v) \mid \ u, v \in S,\ 1 \leq d_G(u, v) \leq k\}.$$

We denote by $[n]$ the set $\{1, 2, \ldots, n\}$. A vertex coloring of a graph $G = (V, E)$ with n vertices is a mapping $\chi : V \rightarrow [n]$ such that if $(u, v) \in E$ then $\chi(u) \neq \chi(v)$. A β–coloring of G (*i.e.*, a coloring that uses at most β colors) is trivially an (α, β)–decomposition of G, for any $\alpha \geq 0$. Given a graph G of maximum degree Δ, it is possible to compute a $\Delta + 1$ coloring of G in $O(\Delta \log n)$ time in the distributed model of computation [GPS89].

The following definition was introduced by Cole & Vishkin [CV86]: an (α, β)–*ruling set* S with respect to $G = (V, E)$ and $P \subseteq V$ is a set of vertices such that:

- $S \subseteq P$;
- any two vertices of S are at distance at least α from each other;
- every vertex $u \in P$ is at distance at most β from some vertex of S.

The notion of (α, β)–ruling set was generalized by Awerbuch, Goldberg, Luby & Plotkin [AGLP89] as follows: an *(α, β)–ruling forest* with respect to $G = (V, E)$ and $P \subseteq V$ is a forest of rooted trees $\mathcal{F} = \{T_i\}$, where each tree is a subgraph of G, with the following properties:

- For all i, the root of T_i, called the *leader* of T_i and denoted by $l(T_i)$, is in P,
- every vertex in P belongs to a unique tree,
- trees are vertex-disjoint,
- inter-root distance is at least α, and
- tree depth is at most β.

Notice that trees of an (α, β)–ruling forest can contain non-P vertices. In the distributed model of computation, a $(k, k \log n)$–ruling set can be computed in $O(k \log n)$ time deterministically [AGLP89]. Given an (α, β)–ruling set, an (α, β)–ruling forest can be computed in $O(\beta)$ time deterministically, so that a $(k, k \log n)$–ruling forest can be computed in $O(k \log n)$ time distributively [AGLP89].

Suppose we have a graph G with a partition of $V(G)$ into A and B where the degree of each vertex in B is at most $\beta - 1$, and are also given an (α, β)–decomposition of $G[A]$ and a β–coloring of $G[B]$. Then, we can compute an (α, β)–decomposition of G in $O(\beta)$ time. We call this the *merging* of the two decompositions (recall that a β-coloring is an $(1, \beta)$-decomposition). The merging can be computed as follows. Let $\mathcal{C}$ be the cluster set of the decomposition of $G[A]$; the cluster set of the new decomposition is $\mathcal{C} \cup B$. Colors of clusters in $\mathcal{C}$ remain the same, while colors of vertices in B are updated according to the following procedure: for $c = 1, 2, \ldots, \beta$, in parallel each vertex with color c chooses a color in $[\beta]$ not chosen by any of its neighbors. This procedure is correct because the set of vertices with color c is an independent set, and neighboring vertices will choose different colors.

We now introduce a notation in the spirit of the $\tilde{O}(\cdot)$ notation used in Computational Geometry. We say that $g(n) = \hat{O}(f(n))$ if there exists $c > 0$ such that $g(n) = O(f(n)^c)$. This convention is introduced to simplify notation.

2.2 Improved Network Decomposition

In this section we present our improved network decomposition algorithm. From now on, $G = (V, E)$ will denote the original input graph with n vertices, while H will denote a generic graph with ℓ vertices. Also, $p = p(n)$ will denote a parameter to be fixed later; the performance of the algorithm will depend on the choice of p. The algorithm uses two mutually recursive procedures $\mathcal{CD}(H)$ to compute an $(n^{\epsilon(n)}, n^{\epsilon(n)})$–decomposition of H with $\epsilon(n) = O(\sqrt{1/\log n})$, and $\mathcal{RF}(P, H)$, to compute a $(3,4)$–ruling forest with respect to P and H. The running time of $\mathcal{CD}(\cdot)$ is $\hat{O}(n^{\epsilon(n)})$. The intuition behind $\mathcal{CD}(H)$, as in [AGLP89], is the following: "small" degree vertices (*i.e.*, vertices of degree less than p) can be handled easily by making them trivial clusters and by p–coloring the graph induced by them in $O(p \log n)$ time. "High" degree vertices (*i.e.*, vertices of degree at least p) can be used to shrink the graph by collapsing their neighborhood into one super-vertex; the resulting graph will shrink (in terms of the number of vertices) by a factor of at least p. After the shrinking, a network decomposition of the collapsed graph is computed with a recursive call to $\mathcal{CD}(\cdot)$, and if the shrinking factor p is high enough the recursion will terminate fast. Finally, the two decompositions, the one of the collapsed graph and the trivial decomposition given by the p-coloring of the small degree vertices, are merged together to give a decomposition of the whole graph.

A symmetry-breaking problem arises when any two high degree vertices want to collapse and their neighborhoods intersect. This problem is handled by computing a $(3,4)$–ruling forest with respect to the set P of high degree vertices and the graph H. In a $(3,4)$–ruling forest, each vertex in P belongs to a unique tree, so that each tree can collapse onto its root without interference from other collapsing trees. Since roots are "high" degree vertices (*i.e.*, have degree at least p) and are at distance at least 3 apart, the shrinking factor is at least p.

An important difference between our algorithm and that of [AGLP89] is that we compute a $(3,4)$–ruling forest with respect to the set of high degree vertices, while they compute a $(3, 3\log n)$–ruling forest. Computing a $(3,4)$–ruling forest gives much better performance for the following reason. Once the ruling forest is computed, each tree is collapsed into a super-vertex: if each tree has $O(\log n)$ diameter, as in the $(3, 3\log n)$–ruling forest, then by the end of the recursion we will have super-vertices with $O((6\log n)^d)$ diameter, where d is the depth of the recursion. On the other hand, with $(3,4)$–ruling forests the final diameter will be $O(8^d)$, which will greatly improve the performance of the simulation of a super-vertex. Any $(3,k)$–ruling forest, for k constant, will do but the smaller the k the better; $k = 4$ is what we could achieve. The problem is that it is not known how to compute a $(3,4)$–ruling forest distributively in polylogarithmic time, whereas this is possible for $(3, 3\log n)$–ruling forests. We solve this problem by a mutually recursive call to $\mathcal{CD}(\cdot)$. Roughly speaking, the problem is solved by considering the graph induced by vertices at distance at most two from a vertex in P and by computing a maximal independent set in this graph. The maximal independent set is computed by partitioning P into roughly balanced sets, each of size $O(\ell/p)$, and by making (mutually) recursive calls in parallel to $\mathcal{CD}(\cdot)$ on graphs of smaller size, namely $O(|V(H)|/p)$.

We now give $\mathcal{CD}(H)$.

PROCEDURE $\mathcal{CD}(H)$

- INPUT: a graph H with ℓ vertices.
- OUTPUT: an $(\ell^{\epsilon(\ell)}, n^{\epsilon(n)})$–network decomposition of H.

1. Let $P = \{v \mid \deg_H(v) \geq p\}$. Compute a $(3,4)$–ruling forest with respect to

P and H with a call to $\mathcal{RF}(P, H)$. Let $\mathcal{F} = \{T_i\}$ be the resulting forest, and let $H_{\mathcal{F}}$ be the graph induced by $\mathcal{F}$, *i.e.*, $V(H_{\mathcal{F}}) = \{T_i \mid T_i \in \mathcal{F}\}$ and

$$E(H_{\mathcal{F}}) = \{(T_i, T_j) \mid i \neq j, \exists u \in T_i, v \in T_j : (u, v) \in E(H)\}.$$

Let S be the set of vertices not covered by the forest, *i.e.*, $S = \{u \mid u \notin \cup_i V(T_i)\}$.

2. Compute a network decomposition of $H_{\mathcal{F}}$ by a recursive call to $\mathcal{CD}(H_{\mathcal{F}})$.

3. Compute a p–coloring of $H[S]$, the subgraph induced by S, and merge it with the network decomposition computed by $\mathcal{CD}(H_{\mathcal{F}})$ (see Section 2.1).

First of all, observe that

$$|V(H_{\mathcal{F}})| \leq \frac{|V(H)|}{p}$$

because $H_{\mathcal{F}}$ is formed by collapsing the trees of $\mathcal{F}$ around their leaders, which are vertices of degree at least p. Hence, the depth of the recursion is at most $d = \log_p \ell$.

We now argue by induction that the diameter and the colors used by the network decomposition are, respectively, $O(8^{\log_p \ell})$ and p. We first prove the claim for the diameter. For the base case, observe that when the recurrence stops each tree has diameter at most 8. For the inductive step, assume that the diameter of clusters returned by $\mathcal{CD}(G_{\mathcal{F}})$ is $8^{\log_p |V(G_{\mathcal{F}})|}$. Each vertex of $G_{\mathcal{F}}$ is a super-vertex obtained by collapsing a tree and has diameter at most 8. Hence, the diameter bound for $\mathcal{CD}(H)$ is

$$8\ 8^{\log_p |V(H_{\mathcal{F}})|} \leq 8\ 8^{\log_p \frac{\ell}{p}} = 8^{\log_p \ell}.$$

To prove that p is the maximum number of colors used by $\mathcal{CD}(H)$ assume inductively that $\mathcal{CD}(H_{\mathcal{F}})$ uses at most p colors. By definition of S the graph $H[S]$ can be p–colored and the merging of $H[S]$ and $\mathcal{CD}(H_{\mathcal{F}})$ also uses p colors. To and $H[Q_i]$. The set $I = \bigcup_i I_i$ is a $(3,2)$–ruling set w.r.t. $Q = \bigcup_i Q_i$ and H and, by construction of the Q_i's, a $(3,4)$–ruling set w.r.t. P and H.

The time complexity of $\mathcal{RF}(P,H)$ is given by the recurrence

$$T_{\mathcal{RF}}(\ell) \leq O(\log n) + \hat{O}(p) + 2\, T_{\mathcal{CD}}\left(\frac{2\ell}{p}\right)$$

where $O(\log n)$ is the time necessary for Step 1, $O(p)$ is the time necessary for step 2 (there are p phases), and where 2 $T_{\mathcal{CD}}(2\ell/p)$ is the time needed for the recursive call to $\mathcal{CD}(\cdot)$, which is called on a square graph (each edge of $H[2,Q_i]$ can be simulated in 2 steps), and $\hat{O}(p)$ is the time needed to compute each I_i.

We now determine the best choice for the parameter p. Our goal is to minimize the running time of $\mathcal{CD}(\cdot)$. By substituting $T_{\mathcal{RF}}(\ell)$ into the equation of $T_{\mathcal{CD}}(\ell)$ we get (in what follows let $q = p/2$ and assume $p \geq \log n$)

$$\begin{aligned} T_{\mathcal{CD}}(\ell) &\leq O(\log n) + \hat{O}(p) + 2\, T_{\mathcal{CD}}\left(\frac{2\ell}{p}\right) + 8\, T_{\mathcal{CD}}(\frac{\ell}{p}) + O(p\, \log n) \\ &\leq \hat{O}(p \log n) + 10\, T_{\mathcal{CD}}\left(\frac{\ell}{q}\right) \\ &\leq \hat{O}(p\, \log n)\, 10^{\log_q \ell} \\ &\equiv f(\ell, p). \end{aligned}$$

By computing the first and second derivatives of the function $\log f(n,p)$ with respect to p (recall $q = p/2$), we can see that the minimum of $f(n,p)$ is attained when $p = 2^{O(\sqrt{\log n})} = \hat{O}(n^{\epsilon(n)})$ with $\epsilon(n) = 1/\sqrt{\log n}$. For this choice of p we get $T_{\mathcal{CD}}(n) = \hat{O}(n^{\epsilon(n)})$. The following theorem summarizes the whole discussion.

Theorem 1 *Given a graph G with n vertices, procedure $\mathcal{CD}(G)$ computes an $(n^{\epsilon(n)}, n^{\epsilon(n)})$–network decomposition of G in $\hat{O}(n^{\epsilon(n)})$ time, where $\epsilon(n) = 1/\sqrt{\log n}$.*

We now present a simple scheme to construct a $(\log n, \log n)$–decomposition, given an algorithm to compute a $(d(n), c(n))$–decomposition. This is inspired by ideas from [AP92,BC].

For this, we first need the sequential algorithm of Awerbuch & Peleg [AP92] to compute a $(\log n, \log n)$–decomposition, which works as follows on a graph $G = (V, E)$ with $|V| = n$. Start with any vertex v. Now, either there exists an index i, $0 \le i \le \lceil \log_2 n \rceil - 1$, such that

$$|\{u : d_G(u,v) \le i\}| \ge |\{u : d_G(u,v) = i+1\}|,$$

or not. If there exists no such index, then G has $O(\log n)$ diameter and hence a trivial $(\log n, \log n)$-decomposition. Otherwise, let $\ell_v \le \log n$ be the smallest such index; let $C_v \doteq \{u | d_G(u,v) \le \ell_v\}$ be the cluster "centered" at v and $B_v \doteq \{u | d_G(u,v) = \ell_v + 1\}$ its "border". Note, crucially, that

$$|B_v| \le |C_v|. \tag{2.1}$$

Remove the vertices $C_v \cup B_v$ and the edges incident at them, and repeat this process on the remaining graph. We then get a sequence of vertices $v = v_0, v_1, v_2, \ldots$ with corresponding clusters $C_{v_0}, C_{v_1}, \ldots$. Each set C_{v_i} now becomes a cluster and gets color 1; note that since each ℓ_v is $O(\log n)$, the diameter of each cluster C_{v_i} is $O(\log n)$. Also, no two clusters C_{v_i} and C_{v_j}, $v_i \ne v_j$, have an edge going from one of them to the other. Hence, these are valid clusters indeed.

Now by removing the set $\cup_i C_{v_i}$ from G and repeating this on the remaining graph $G[\cup_i B_{v_i}]$ to assign color classes $2, 3, \ldots$, we compute a network decomposition. The crucial property is that the number of vertices in the new graph $G[\cup_i B_{v_i}]$ is at most half that of G, by (2.1); thus, the number of color classes is at most $\lceil \log_2 n \rceil$. Hence, this yields a $(\log n, \log n)$–decomposition.

The next theorem shows that, given a $(c(n), d(n))$-decomposition, the above linear-time algorithm can be efficiently simulated in the distributed model of computation.

Theorem 2 *Suppose we are given a distributed algorithm $\mathcal{A}$ with running time $O(t(n))$, to compute a $(d(n), c(n))$–decomposition. Then, given any network $G =$*

(V, E) with n vertices, we can compute a $(\log n, \log n)$–decomposition of G in $O(t(n)\log n + c(n)d(n)\log^2 n)$ time distributively.

PROOF. For this, we rely heavily upon the above-seen sequential scheme; we proceed as follows.

1. Use algorithm $\mathcal{A}$ to compute a $(\log n, \log n)$–decomposition of $G[2\log n, V]$.

2. For *newcolor* $= 1, 2, \ldots, \log n$ do:

   - For $c = 1, 2, \ldots, O(c(n))$ do: in parallel, each cluster $\hat{C}$ of color c computes a maximal collection of sets C_v, for $v \in \hat{C}$, and assigns color *newcolor* to them.

The crucial observation is that if u and v belong to two different clusters $\hat{C}_1$ and $\hat{C}_2$ of color c, then C_u and C_v generated in the body of the loop do not interfere because, by step 1, $d_G(u, v) \geq 2\log n + 1$ and the radius of C_u and C_v is at most $\log n$.

Step 1 takes $O(t(n)\log n)$ time, since it takes $O(\log n)$ time to simulate each edge of $G[2\log n, V]$. The simulation of the body of the nested loop in step 2 takes $O(d(n)\log n)$ time. Thus, step 2 takes $O(c(n)d(n)\log^2 n)$ time. □

As mentioned in the introduction, several problems are reducible to network decomposition.

Corollary 1 *Given a network G with n vertices, the following functions can be computed in $\hat{O}(n^{\epsilon(n)})$ time in the distributed model of computation, with $\epsilon(n) = 1/\sqrt{\log n}$:*

- *maximal independent set,*
- *$(2\Delta - 1)$–edge coloring,*

- *maximal matching,*
- *Δ–vertex coloring [PS92c],*
- $(\log n, \log n)$*–network decomposition.*

The first three statements are a straightforward application of the general algorithm discussed in the introduction. The proof of claim 4 can be found in Chapter 3 (Section 3.5.2) or in the references, and the last claim follows from Theorems 1 and 2.

2.3 Completeness

In the previous section we gave an algorithm for computing a network decomposition of a graph in time $\hat{O}(g(n))$, where $g(n) = 2^{\sqrt{\log n}}$. In this section we characterize a class of graphs $\mathcal{G}$ that is complete for the task of computing a network decomposition. Here, completeness is to be interpreted in the following sense: if we have an algorithm for computing a decomposition for graphs in $\mathcal{G}$ in $\hat{O}(t(n))$ time, then we can compute a decomposition for all graphs in $\hat{O}(t(n))$ time.

More precisely, let $h(n)$ be any non–decreasing function, let $p(n) = g(n)^{1/h(n)}$, and let $q(n) = g(n)^{h(n)}$. Suppose we have an algorithm $\mathcal{A}$ that computes a $(p(n), p(n))$–decomposition of graphs with maximum degree $\Delta \leq q(n)$ in time $\hat{O}(p(n))$. Then, we can compute a $(p(n), p(n))$–decomposition in time $\hat{O}(p(n))$ for all graphs. This means that we need to concentrate our efforts on graphs with maximum degree at most $q(n)$; these are the difficult graphs to handle. For example, in order to have a $(polylog(n), polylog(n))$–decomposition algorithm running in $\hat{O}(\log n)$ time we just need to look at graphs of maximum degree less than $q(n) = n^{O(1/\log\log n)}$, a quantity smaller than n^{ϵ} for any ϵ.

We may further add that since it is possible to $(\Delta + 1)$–vertex color graphs in time $O(\Delta \log n)$ [GPS89], where Δ denotes the maximum degree of the graphs, the class of graphs that is complete for decomposition is that of graphs with

$\Delta \in [\Omega(p(n)), \hat{O}(q(n))]$. For example, for $p(n) = polylog(n)$ the value of $h(n)$ is $\sqrt{\log n}/\log\log n$, and the range becomes $[\Omega(polylog(n)), \hat{O}(n^{\delta(n)})]$, where $\delta(n) = 1/\log\log n$.

These results tell us what the bottleneck is for the method used in [AGLP89] and in this paper. The method's basic idea is to expand clusters as long as their degree is high enough, and to color clusters when their degree is not high enough. This approach is successful if the final diameter of the resulting clusters is not too high, *i.e.*, if the recursion terminates fast. For this to happen the expansion rate of the clusters has to be high enough. But if the maximum degree is in the range $(\Omega(polylog(n)), \hat{O}(n^{\delta(n)}))$, corresponding to $p(n) = polylog(n)$, then the maximum degree of clusters is too low and the diameter too high for the shrinking to terminate in polylogarithmic time. This is an indication that a completely different approach is needed to solve the network decomposition problem in $O(polylog(n))$ time.

We now turn to the task of showing the completeness result. The idea is to use the supposed algorithm $\mathcal{A}$ as a subroutine in conjunction with a modified version of the procedures $\mathcal{CD}(\cdot)$ and $\mathcal{RF}(\cdot)$. As before, the two procedures will call each other in a mutually recursive fashion. The initial input is a graph G with n vertices; the values $p(n)$ and $q(n)$ remain constant in the following analysis.

As before, procedure $\mathcal{CD}(H)$ splits vertices into "high" and "low" degree vertices. Here, high degree means at least $q(n)$. But, instead of coloring the graph induced by the low degree vertices as before, algorithm $\mathcal{A}$ is invoked. Another difference is that procedure $\mathcal{RF}(P, H)$ returns a $(3,6)$–ruling forest w.r.t. P and H, instead of a $(3,4)$–ruling forest. On input graph H, procedure $\mathcal{CD}(H)$ is as follows:

PROCEDURE $\mathcal{CD}(H)$

- INPUT: a graph H with ℓ vertices.
- OUTPUT: a $(p(\ell), p(n))$–network decomposition of H.

1. Let $P = \{v \mid \deg_H(v) \geq q(n)\}$. Compute a $(3,6)$–ruling forest with respect to P and H with a call to $\mathcal{RF}(P, H)$. Let $\mathcal{F} = \{T_i\}$ be the resulting forest, and $H_{\mathcal{F}}$ be the cluster graph induced by $\mathcal{F}$. Let S be the set of vertices not covered by the forest, *i.e.*, $S = \{u \mid u \notin \cup_i V(T_i)\}$. (Comment: all this is the same as before).

2 . Compute a network decomposition of $H_{\mathcal{F}}$ by a recursive call to $\mathcal{CD}(H_{\mathcal{F}})$.

3. Compute a $(p(n), p(n))$–decomposition of $H[S]$, the subgraph induced by S, by using algorithm $\mathcal{A}$, and merge it with the network decomposition computed by $\mathcal{CD}(H_{\mathcal{F}})$.

The time complexity of $\mathcal{CD}(H)$ is given by the following recurrence

$$T_{\mathcal{CD}}(\ell) \leq T_{\mathcal{RF}}(\ell) + 12\, T_{\mathcal{CD}}\left(\frac{\ell}{q(n)}\right) + \hat{O}(p(n)).$$

$\hat{O}(p(n))$ is the time needed to merge the decomposition computed by $\mathcal{A}$ on $H[S]$ and the one computed by $\mathcal{CD}(H_{\mathcal{F}})$.

The modified version of $\mathcal{RF}(P, H)$ is trickier to design. Intuitively, $q(n)$ is defined in such a way that if we partition P into groups of size $O(\ell/q(n))$, and call $\mathcal{CD}(\cdot)$ in parallel on such groups, then the recursion will terminate in $\hat{O}(p(n))$ time. The problem is that the requirements of a $(3,6)$–ruling forest can be satisfied locally in each group, but might be violated by vertices belonging to different groups. This problem was eliminated in the old algorithm by a "filtering" process of cycling through the groups. This is not possible now because there are $q(n)$ groups and we cannot afford to run a loop for that long. The problem is solved by invoking algorithm $\mathcal{A}$ on a suitable interference graph whose maximum degree is at most $q(n)$.

PROCEDURE $\mathcal{RF}(P, H)$

- INPUT: a graph H with ℓ vertices, and a set $P \subseteq V(H)$.

- OUTPUT: a $(3,6)$–ruling forest with respect to P and H.

1. Partition P into disjoint sets P_i, $i \in [q(n)]$, as before, by computing a $(3, 3\log n)$–ruling forest and by assigning numbers $1, 2, \ldots, q(n)$ cyclically inside each tree.

2. Let $H_i = H[4, P_i]$, $i \in [q(n)]$. In parallel, for all $i \in [q(n)]$, compute an MIS I_i of H_i by means of a call to $\mathcal{CD}(H_i)$. Let $I = \cup_i I_i$.

3. Let $F = H[2, I]$. (Comment: we will show that the maximum degree of F is $\Delta(F) \leq q(n)$.) Invoke algorithm $\mathcal{A}$ to compute an MIS J of F. The set J is a $(3,6)$–ruling set w.r.t. P and H. From J a $(3,6)$–ruling forest is computed.

We first show correctness of the algorithm and then that it achieves the desired time bounds. The set P is partitioned in step 1 with the same method employed in the old algorithm; it follows that $|P_i| \leq 2\ell/q(n)$, $i \in [q(n)]$. The correctness of step 2 follows from the general fact that given an (α, β)–decomposition an MIS can be computed in $O(\alpha\beta)$. For step 3, we have to show that $\Delta(F) \leq q(n)$ and that J is a MIS. Consider any vertex $u \in I_i \subseteq I = V(F)$. First notice that u cannot have a neighbor $v \in I_i$ because if $u, v \in I_i$ then, by definition of H_i and I_i, $d_H(u,v) \geq 5$ and hence $(u,v) \notin E(F)$. We will show that any $u \in I_i$ can have at most one neighbor $v \in I_j$, for $I_j \neq I_i$; since there are $q(n)$ groups the claim on the degree follows. Suppose for a contradiction that u has two neighbors v_1 and v_2 in the same group I_j. From the definition of F it follows that $d_H(v_1, v_2) \leq 4$, an impossibility because then, by definition of H_j, $(v_1, v_2) \in E(H_j)$, and v_1 and v_2 cannot both belong to I_j.

In order to show that J is a $(3,6)$–ruling forest w.r.t. P and H we need to show that: *i)* any vertex $u \in P$ is at distance at most 6 from some $v \in J$, and *ii)* for any two vertices $v_1, v_2 \in J$, $d_H(v_1, v_2) \geq 3$. Let P_i be the group u belongs to; by definition of I_i and H_i, vertex u is at distance at most 4 from some vertex $v \in I_i$. If v also belongs to the final MIS J then we are done, otherwise v must be adjacent to a vertex $w \in J$ in $H[2, I]$, because J is an MIS of H. From the definition of F, $d_H(v, w) \leq 2$, which implies $d_H(u, w) \leq 6$. Finally, we show that any two vertices in J are at distance at least 3 from each other. Let v_1, v_2 be any two vertices in J; if they come from the same I_i then they are at distance at least 5 apart (by definition of H_i), otherwise they are at distance at least 3 (by the definition of F and J).

We now come to the complexity analysis. Step 1 is $O(\log n)$. Step 2 takes $4\ T_{\mathcal{CD}}(2\ell/q(n))$ because we need 4 time units to simulate one edge of any H_i, and each H_i has size at most $2\ell/q(n)$. The complexity of Step 3 is dominated by the complexity of algorithm $\mathcal{A}$, which is $\hat{O}(p(n))$. Hence, we get the following recurrence

$$T_{\mathcal{RF}}(\ell) \leq O(\log n) + 4\ T_{\mathcal{RF}}\left(\frac{\ell}{q(n)/2}\right) + \hat{O}(p(n))$$

By substituting in the expression for $T_{\mathcal{CD}}(\ell)$ we get

$$\begin{aligned} T_{\mathcal{CD}}(\ell) &\leq 16\ T_{\mathcal{CD}}\left(\frac{\ell}{q(n)/2}\right) + \hat{O}(p(n)) \\ &\leq \hat{O}(p(n))\ 16^{\frac{\log \ell}{\log q(n) - 1}} \\ &= \hat{O}\left(p(n)\ \exp\left(\frac{\log \ell}{\log q(n)}\right)\right) \\ &= \hat{O}\left(p(n)\ \exp\left(\frac{\log \ell}{h(n) \log g(n)}\right)\right) \\ &= \hat{O}\left(p(n)\ \exp\left(\frac{\sqrt{\log \ell}}{h(n)}\right)\right) \\ &= \hat{O}(p(n)). \end{aligned}$$

We have thus shown the following theorem.

Theorem 3 *Let $h(n)$ be any non–decreasing function and let $p(n) = g(n)^{1/h(n)}$, and $q(n) = g(n)^{h(n)}$. Suppose we have an algorithm $\mathcal{A}$ that computes a $(p(n), p(n))$–decomposition of graphs with maximum degree $\Delta \leq q(n)$ in time $\hat{O}(p(n))$. Then, we can compute a $(p(n), p(n))$–decomposition in time $\hat{O}(p(n))$ for all graphs.*

Chapter 3

The Local Nature of Δ–colorings and Its Algorithmic Applications

3.1 Introduction

In many situations, an independent set defines a set of processes that can compute in parallel without mutual interference. A good vertex coloring (*i.e.*, one that uses few colors) partitions the vertices of a network into independent sets, and hence defines a schedule for the processors to work in parallel. This makes vertex coloring a useful primitive in parallel and distributed computing. Lynch and Choy & Singh use vertex coloring to give distributed solutions to classical resource allocation problems such as the *Dining Philosophers* of Dijkstra [CS92,Lyn81].

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 chapter 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 [Bro41] (see also the discussion in Bollobás [Bol79]), 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 chapter.

There is a simple and beautiful randomized distributed algorithm to compute a $(\Delta + 1)$-coloring in $O(\log n)$ expected time, due to Luby [Lub88]. 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 [LPV81]). 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 $O(\Delta \text{ polylog}(n))$ 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 [Lub88], 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 [HS90], Karchmer & Naor [KN88], and Karloff [Kar89]).

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 thesis. In a sequential setting, the small radius search yields a Las Vegas algorithm running in $O(\sqrt{n})$ time with success probability of at least $1/2$. The error probability can be made arbitrarily small by repeated runs of the algorithm.

By making use of the notion of network decomposition 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.

3.2 Definitions

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 that vertex 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 3.1). Clearly, a walk transforms a partial Δ–coloring into another partial Δ–coloring.

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 $\Delta \geq 3$.*

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 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

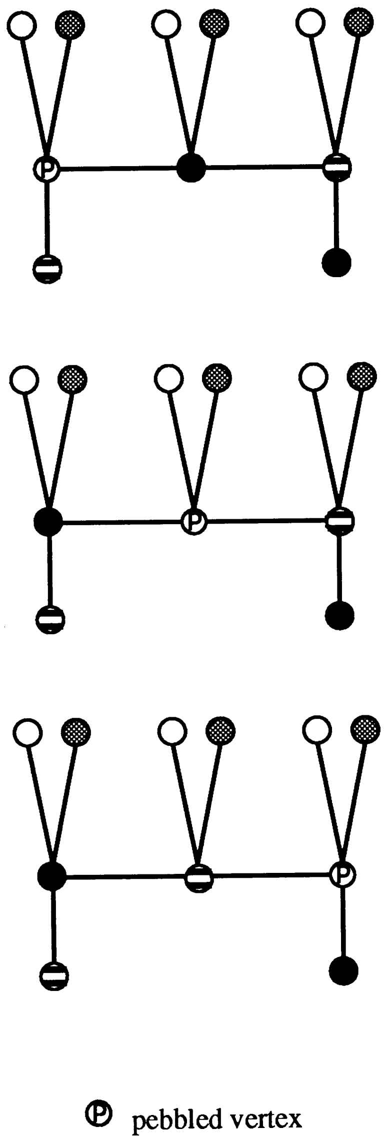

Figure 3.1: Steps and walks

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.

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. The next definition is crucial.

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.*

The basic tool to prove our main theorem is a forthcoming lemma called the Spawning Lemma. The lemma 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 $O(\log_\Delta n)$. Before the proof of the Spawning Lemma we need a few more definitions and a lemma.

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 $v_{j+k} \in \{v_{i+1}, \ldots, v_{p-1}\}$, where $k \geq 2$, 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 3.2). In the sequel, when there is no danger of confusion, we shall refer to a P-detour simply as a detour.

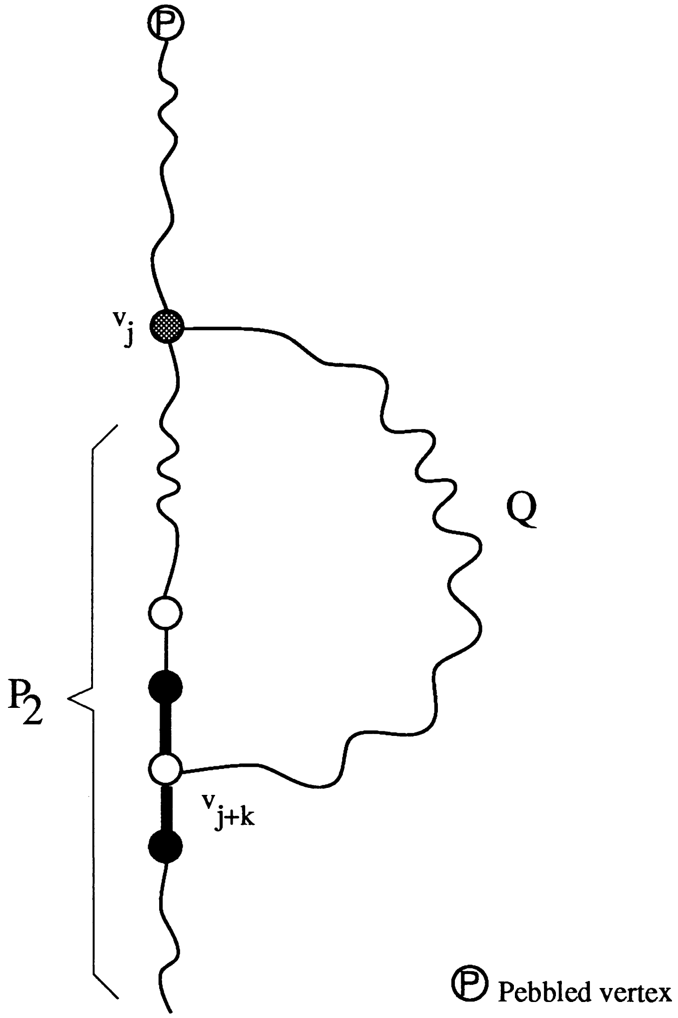

Figure 3.2: A detour yields a T-path

Let $P = v_0, \ldots, v_p$, and let P' be a path originating from some vertex $v \in P$. If $P \cap P' = \{v\}$ we say that P and P' are *divergent.*

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

The next lemma is used in the proof of the Spawning Lemma.

Lemma 3 *Let $P = v_0 \bullet P_1 \bullet P_2$ be a stem with $P_2 = v_i, \ldots, v_p$, and $\chi(P_2) = \{\alpha, \beta\}$. Let Q be a bichromatic simple path originating from $x \in \{v_{i+1}, \ldots, v_{p-1}\}$. Then, either Q is a $|Q|$-branch or there exists a T-path of length at most $|P| + |Q|$.*

PROOF. If there is a sanctuary in $P \cup Q$ then we are done. Suppose not and let, without loss of generality, $\chi(Q) = \{\alpha, \gamma\}$ where $\gamma \neq \beta$, and $\chi(x) = \alpha$. Let x^- and x^+ be the two neighbors of x lying on P, with x^- being closer to v_0 than x^+. Suppose that Q is not a $|Q|$-branch and let $w \in P \cap Q$ be the first point of P that is met when walking along Q coming from x. If $w \in \{v_1, \ldots, v_{p-1}\}$ then Q is a P-detour and we are done. Otherwise, $w = v_p$ and we distinguish two cases (refer to Figure 3.3). Let $A = v_0, \ldots, x$, and $B = x, \ldots, v_p \equiv w$. Notice that B is an (α, β)-path and recall Q is an (α, γ)-path. Then, it must be that $\chi(w) = \alpha$.

- If $N(w) \cap A = \emptyset$ then either $A \bullet B$ or $A \bullet Q$ is a T-path of the desired length. To see this, suppose that w has another neighbor u colored β; then $A \bullet Q$ is a T-path because $u \notin A \cup Q$. Similarly, if w has another neighbor colored γ then $A \bullet B$ is a T-path. If w has neither then both $A \bullet B$ and $A \bullet Q$ are T-paths because $|N(w) - (B \cup Q)| = \Delta - 2$, and $|\chi(N(w) - (B \cup Q))| \leq \Delta - 3$.

- If $N(w) \cap A \neq \emptyset$ then a T-path of the desired length can be found as follows. Again, we distinguish two cases. Let $z \in N(w) \cap A$; if $z = x^-$ then $P' = v_0, \ldots, x^-$ is a T-path, because $\chi(x) = \chi(w) = \alpha$. Otherwise, the path $P' = v_0, \ldots, z, w \bullet Q^R$ is a T-path, where Q^R is the path Q taken "backwards" from w to x. In other words, there is a way of walking the pebble from v_0 to x without touching x^- and x^+. (Recall that $\chi(x^-) = \chi(x^+) = \beta$).

□

We can now state and prove the Spawning Lemma. From now on, let α, β and γ be three distinct colors.

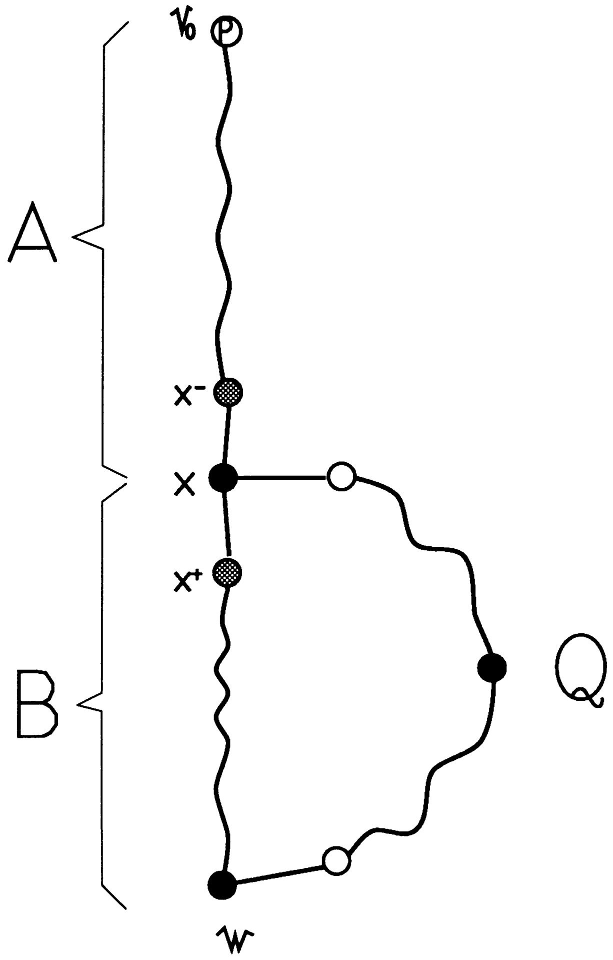

Figure 3.3: Situation when $v_p \in Q \cap P$

Lemma 4 [Spawning lemma] *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 $S = v_0 \bullet P_1 \bullet P_2$ 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+2} be the path obtained by following an (α, γ)-path Q_2 starting

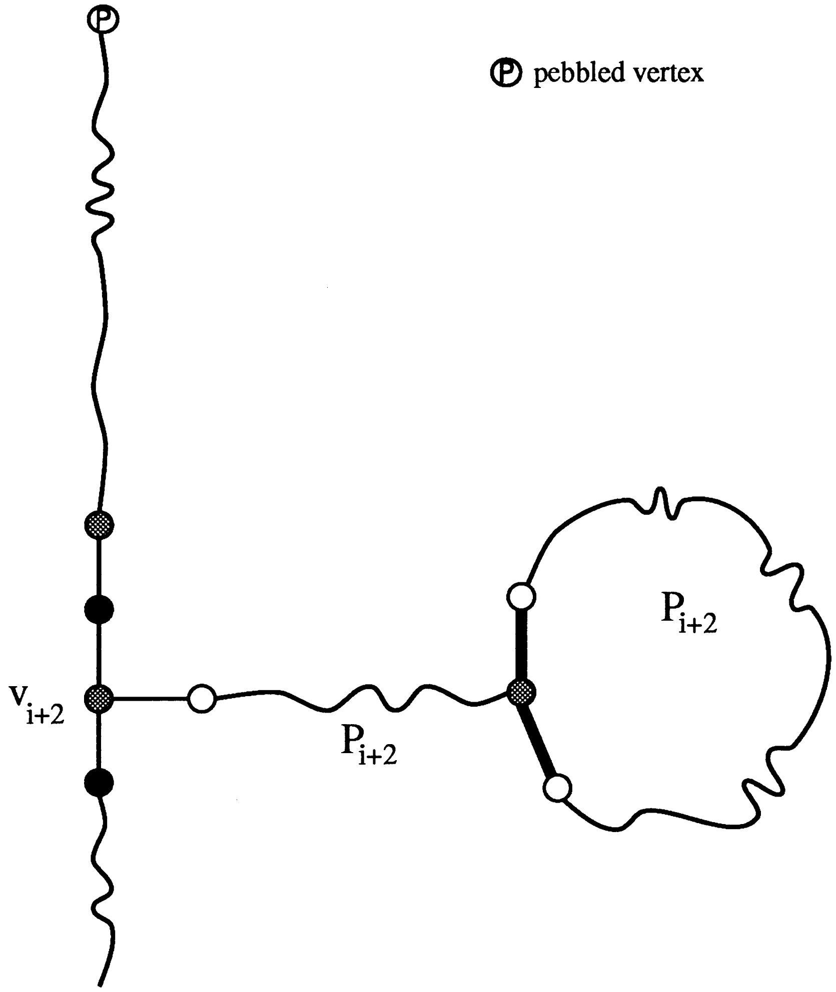

Figure 3.4: If P_{i+2} is not simple we find a T-path

from v_{i+2} for ℓ edges or till Q_2 ends, whichever occurs earlier. We first show that either P_{i+2} is simple and divergent from the stem S, or there is a T-path of length at most 2ℓ.

First, by Lemma 3 P_{i+2} and P must be divergent, or there is a T-path of length at most 2ℓ.

Second, suppose that P_{i+2} is not simple; then $P' = v_0, \ldots, v_{i+2} \bullet P_{i+2}$ contains a T-path (see Figure 3.4).

If the length of P_{i+2} is ℓ, then we are done. Otherwise, we show that either there is an edge between v_{i+1} and the last vertex of P_{i+2} or $v_0, \ldots, v_{i+2} \bullet P_{i+2}$ is a T-path. Let z^- and z be the next-to-last vertex and the last vertex of P_{i+2}, respectively. First, notice that $N(z) \cap P_{i+2} = \{z^-, z\}$ because P_{i+2} is simple. Then, for $v_0, \ldots, v_{i+2} \bullet P_{i+2}$ not to be a T-path there must be an edge between z and $v_0, \ldots, v_{i+1}$. But if $N(z) \cap \{v_0, \ldots, v_i\} \neq \emptyset$ then there is a detour to v_{i+2}. The only possibility then is that (v_{i+1}, z) is an edge.

Let P_{i+1} be the path obtained by following an (α, γ)-path Q_1 starting with the edge (v_{i+1}, z) and continuing for ℓ edges or till Q_1 ends, whichever occurs earlier. P_{i+1} has the same properties of P_{i+2}. Namely, it is simple and divergent from the stem S. By the same analysis it also follows that either $v_0, \ldots, v_{i+1} \bullet P_{i+1}$ is a T-path or there is an edge between v_i and the last vertex of P_{i+1}. But now this 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 3.5). □

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 2 *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 S 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 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 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 a T-path of length at most four.*

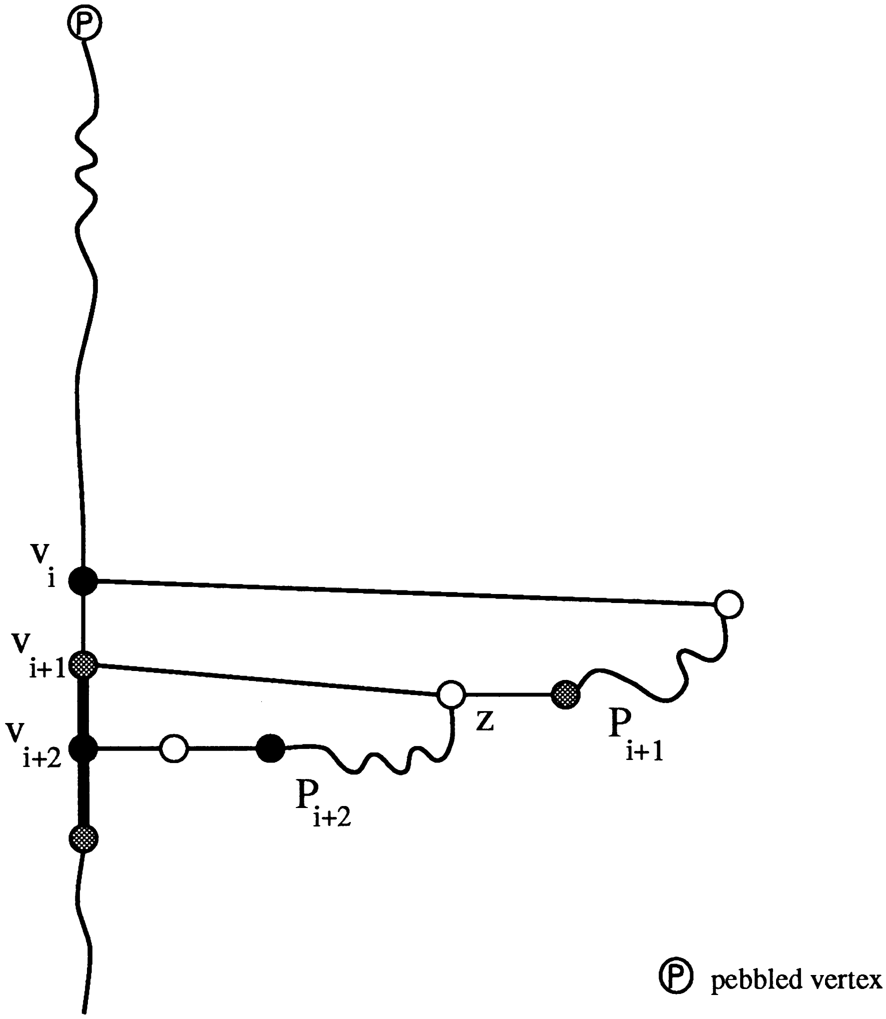

Figure 3.5: If P_{i+1} and P_{i+2} fail we find a T-path

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 3 *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 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 5 ensures that we can find a first stem P of length four. Given P, with one application of Lemma 4, 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 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, then there is an S-detour of length at most 2ℓ, and hence a T-path of length at most $3\ell + 4$ (see Figure 3.6). Any Q_i which is not divergent from the stem S yields a T-path of length at most $2\ell + 4$.

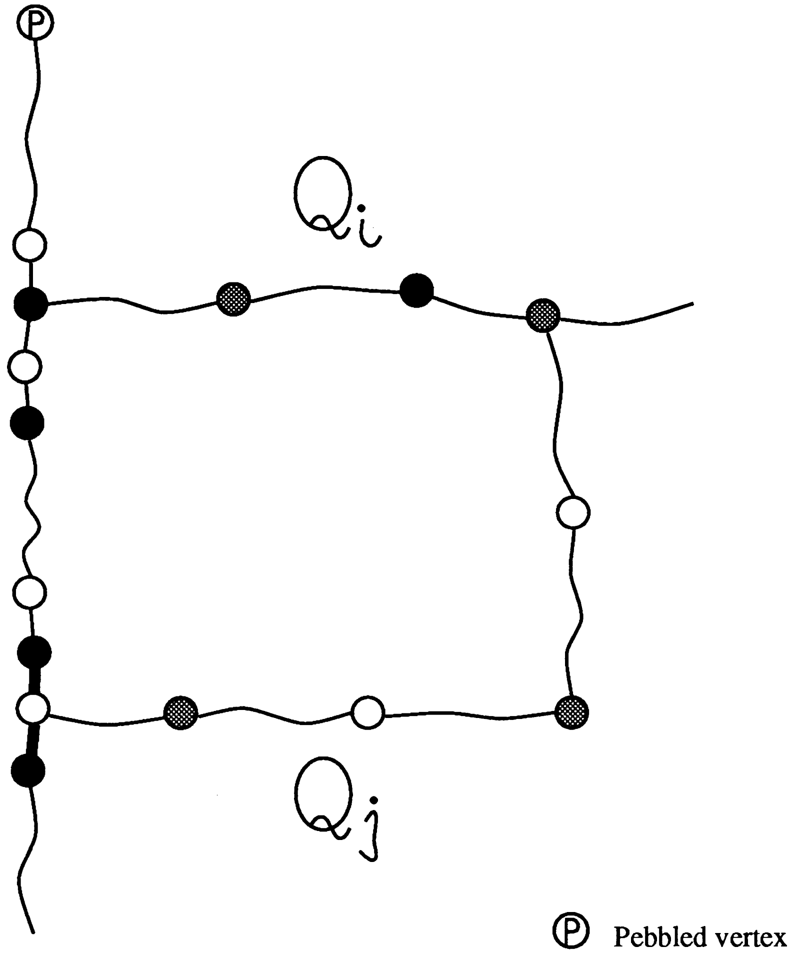

Figure 3.6: Q_i and Q_j cannot intersect

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 spawned off at least $\ell/3$ ℓ-branches.

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 (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 4 *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 $O(\log_\Delta n)$ is 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 2. 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 5. 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 $P' = w_0, w_1 \bullet B_1 \bullet B_2$. 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 2, 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 4, 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 3.7).

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 3.8). 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 or 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. $\square$

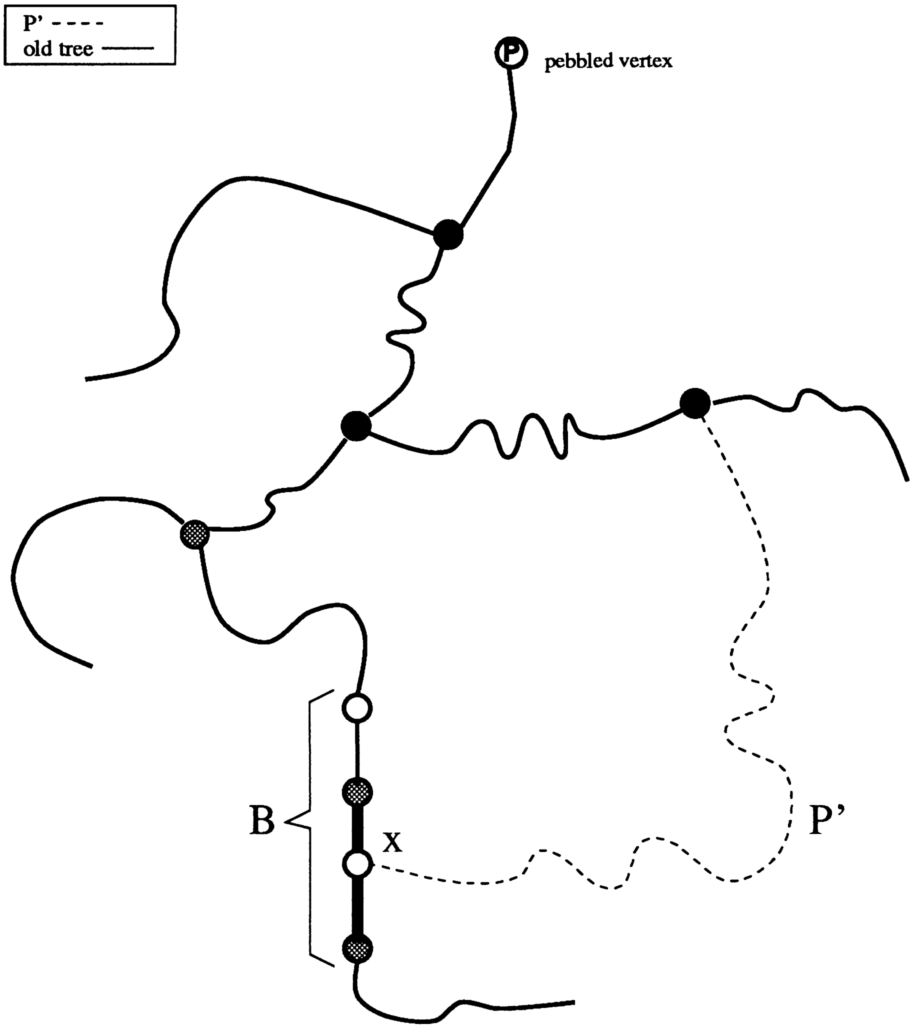

Figure 3.7: New bichromatic paths are divergent from the old tree

Note that for $\Delta \geq 3$, $7\log_{2\Delta-4} n + 11 = \Theta(\log_{\Delta} n)$. Theorem 4 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 4 can be implemented in $O(\log_{\Delta} n)$ time in the distributed model of computation, and with linearly many processors in the PRAM model.

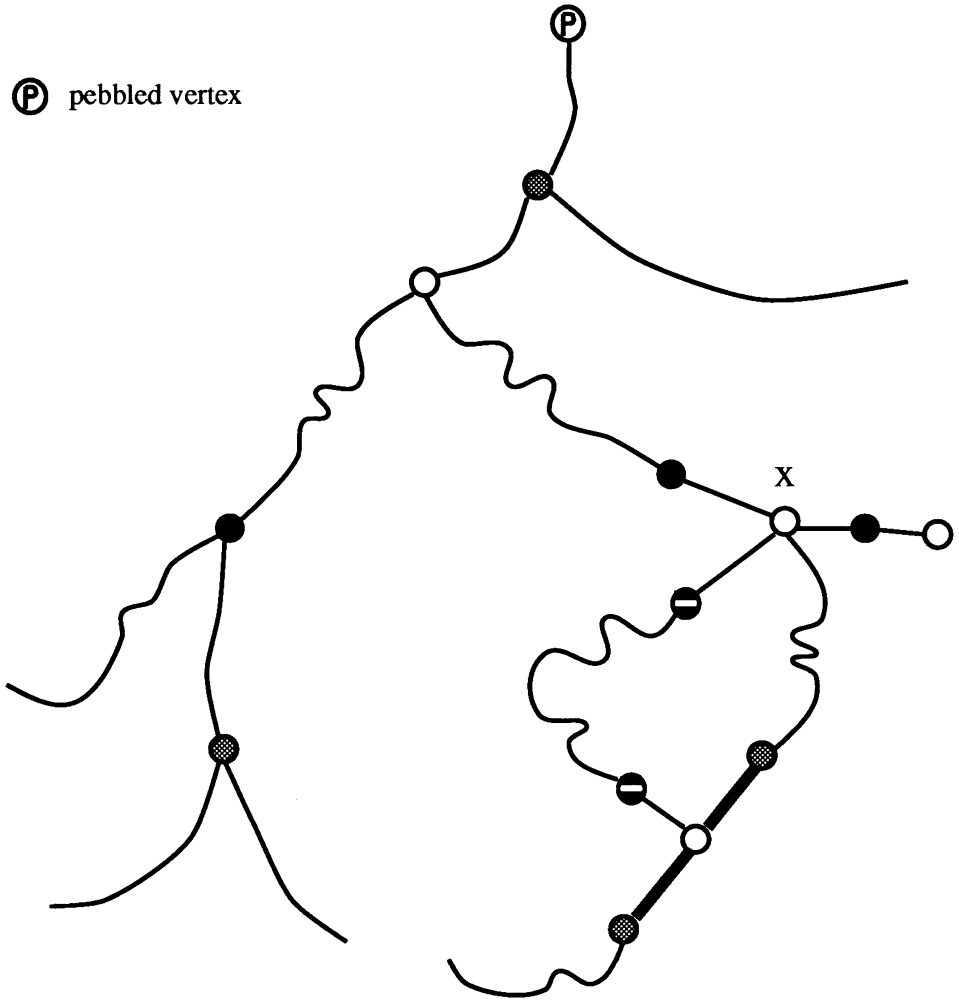

Figure 3.8: Bichromatic paths cannot meet

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.

3.4 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 deterministic algorithm. These yield, respectively, randomized and deterministic distributed algorithms for Δ–coloring with the same complexity bounds.

3.4.1 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 4; 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 4 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 [Lub88]. 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

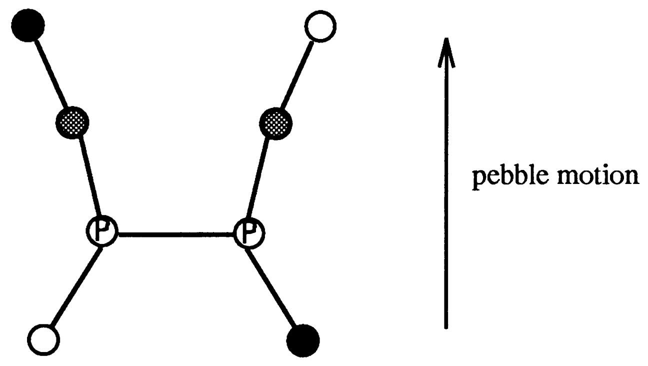

Figure 3.9: Steps must be scheduled

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 3.9).

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 6 *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, with high probability, all non-root pebbles are removed within $O(\log^3 n/\log\Delta)$ time: 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 7 *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 [Kar91]. 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 [Lub88]. 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 [Kar91], 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 [Lub88].

- 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 [AGLP89].

- 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 5 *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.*

3.4.2 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 [GPS89].

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 [GPS89]. 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 [AGLP89].

- 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 6 *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.*

3.5 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 3.4.1 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 $\epsilon(n) = O(1/\sqrt{\log n})$.

3.5.1 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 [Lub88]. To our knowledge, this is the first linear processor NC algorithm for Δ–coloring; the existing algorithms seem to require superlinear processors [HS90,KN88,Kar89].

The distributed randomized algorithm of section 3.4.1 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 [Lub88].

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 [Lub88] 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 7 *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.*

3.5.2 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 $\epsilon(n) = O(1/\sqrt{\log n})$ [PS92b]. 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 8 *A nice graph G is Δ-colorable in $O(n^{\epsilon(n)})$ time in the distributed model of computation, where $\epsilon(n) = O(1/\sqrt{\log n})$.*

3.5.3 A Sublinear Time Las Vegas Algorithm

The proof of the existence of a T-path of $O(\sqrt{n})$ length can be modified to give a randomized procedure that with probability at least $1/2$ succeeds at removing the pebble.

As in the proof, we first generate an initial stem of length 4 and then apply the Spawning Lemma with $\ell = 6\sqrt{n}$, to spawn off an ℓ-branch S. S is subdiveded into $2\sqrt{n}$ contiguous blocks, each made of three consecutive edges. Of these blocks, we arbitrarily choose $\sqrt{n}$ of them to form a collection $\mathcal{B}$. If we applied the Spawning Lemma in each block to spawn off an ℓ-branch at least one block would generate a T-path or find a path to a sanctuary. We *mark* this "successful" block, remove it from the collection $\mathcal{B}$, and insert a new "unused" block in the collection. Again one of them must succeed, since we have $\sqrt{n}$ blocks. We mark this new block and repeat. At the end of this procedure we have marked $\sqrt{n}$ successful blocks. Hence if we pick one block at random among the initial $2\sqrt{n}$–many blocks we pick a successful block with probability at least $1/2$.

The failure probability can be made arbitrarily small by repeated runs of the algorithm.

The running time of this procedure is proportional to the length of the ℓ–branches that are generated, which is $O(\sqrt{n})$.

3.6 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 [Lin92]

3.6.1 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 9 *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

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

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. □

3.6.2 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

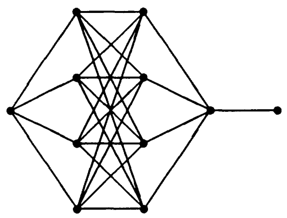

Figure 3.10: A 5-widget

of the network is the same in a neighborhood of radius t, but not outside this neighborhood.

Theorem 10 *For any $\Delta \geq 2$, 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 3.10.

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

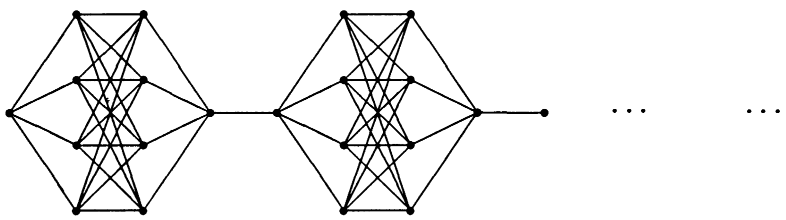

Figure 3.11: A chain of widgets

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 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. □

3.6.3 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 11 *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 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:

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

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 12 *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 11. □

Corollary 4 *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 11 and 12 go through if we use the chain of widgets of Theorem 10 instead of a path. □

Chapter 4

Fast Randomized Edge Coloring Via an Extension of the Chernoff-Hoeffding Bounds

4.1 Introduction

The edge coloring problem can be used to model certain types of jobshop scheduling, packet routing, and resource allocation problems in a distributed setting. For example, the problem of scheduling I/O operations in a certain parallel architecture can be modeled as follows [JSWB92]. We are given a set of processes $\mathcal{P}$ and a set of resources $\mathcal{R}$ such that each process $p \in \mathcal{P}$ needs a subset $f(p) \subseteq \mathcal{R}$ of the resources where: *(i)* each process p needs every resource in $f(p)$ for a unit of time each, and *(ii)* p can use the resources in $f(p)$ in any order. From this, we can construct a bipartite graph $G_{\mathcal{P},\mathcal{R}} = (\mathcal{P}, \mathcal{R}, E_{\mathcal{P},\mathcal{R}})$ where $E_{\mathcal{P},\mathcal{R}} = \{(p,r) |\ p \in \mathcal{P} \wedge r \in f(p)\}$. An edge coloring of $G_{\mathcal{P},\mathcal{R}}$ with c colors yields a schedule for the processes to use the resources within c time units. Optimal colorings correspond to optimal schedules.

Edge coloring can also be used in distributed models in situations where broadcasts are infeasible or undesirable: an edge coloring of the network results in a schedule for each processor to communicate with at most one neighbor at every step; at time step i, processors communicate via the edges colored i only. Using a "small" number of colors reduces the wastage of time in this schedule.

Note that Δ colors are necessary to edge color a graph with maximum degree Δ. Vizing showed that it is always possible to edge color a graph with $\Delta + 1$ colors and gave a polynomial time algorithm to compute such a coloring [Bol79]. Efforts to parallelize Vizing's theorem have failed so far. The best known algorithm is an RNC algorithm of Karloff & Shmoys that uses $\Delta + o(\Delta)$ colors. The algorithm can be derandomized in NC by standard techniques by Berger & Rompel, and Motwani, Naor & Naor [BR91,MNN89]. In the distributed model, the best known edge coloring algorithm is to apply a vertex coloring algorithm to the line graph $L(G)$ of G. We saw in Chapter 3 that there are fast (polylogarithmic) randomized $(\Delta+1)$– and Δ–vertex coloring algorithms which translate to $(2\Delta-1)$– and $(2\Delta-2)$– edge coloring algorithms respectively [Lub88,PS92b]. There are no known polylogarithmic deterministic algorithms in the distributed setting for $(2\Delta-1)$–edge coloring [AGLP89,PS92b]. Moreover, as we saw in Chapter 3, distributed Δ–edge coloring for bipartite graphs requires $\Omega(diameter(G))$ time, even with randomization[PS92b].

In this chapter, we present fast and simple randomized algorithms to edge color G with at most $(1.6+\epsilon)\Delta + 0.4\log^{2+\delta} n$ colors with high probability for any fixed $\epsilon, \delta > 0$, where Δ is the maximum degree of the vertices of G. At the heart of our analysis is an extension of the Chernoff–Hoeffding bounds, which are key tools in bounding the tail probabilities of certain random variables [Che52,Hoe63,Rag90].

The edge coloring algorithm is based on a very simple randomized algorithm to color bipartite graphs. The algorithm can be explained in a few lines. Given a bipartite graph $G = (A, B, E)$ with maximum degree Δ, each vertex in B picks distinct colors from $\{1, 2, \ldots, \Delta\}$ at random for its edges without replacement, *i.e.*, edges incident to the same vertex get different colors. Then, each vertex $v \in A$

checks, for each color α, if more than one of its incident edges has color α and if so, chooses one of them at random as the *winner*, and all the other edges of color α which are incident to v are decolored. The key claim is that for every vertex, the number of decolored edges incident to it is at most $\Delta(1+\epsilon)/e$ with high probability (e is the Euler number). Assuming that this holds, we can repeat the above iteration with a set of $\Delta(1+\epsilon)/e$ *fresh* colors, and so on. In spite of its simplicity the algorithm requires an interesting probabilistic analysis which requires requires an extension of the Chernoff-Hoeffding bounds to a certain case of dependence among the random variables that we call λ–correlation. This extension has several other applications besides the ones given in this thesis [Sri93].

4.2 Notation

In this section we introduce some notation we will be using in the paper.

Given an undirected graph $G = (V, E)$ we denote by Δ its maximum degree, *i.e.*, the maximum number of edges incident with any node; by d_u we denote the degree of vertex u, and by $\delta(u)$ we denote the set of edges incident with vertex u.

Given a positive integer n, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. Given a matrix M with c columns and r rows where $c \leq r$, the generic entry of M is denoted as M_{ik} where i denotes the *column* number and k the *row* number. (This is the reverse of the usual definition but it is more natural when we define the permanent of M.) The permanent of a (possibly non–square) matrix M is defined as the natural extension of the permanent of square matrices. Let $\mathcal{P} = \{\pi \mid \pi : [c] \rightarrow [r],\ \pi \text{ is one–to–one}\}$. Then,

$$perm(M) = \sum_{\pi \in \mathcal{P}} M_{1,\pi(1)}\ M_{2,\pi(2)}\ \cdots\ M_{c,\pi(c)}.$$

An event $\mathcal{A}$ is said to happen *with high probability* (*w.h.p.*) if $Pr(\mathcal{A}) \geq 1 - 1/f(n)$ for some superpolynomial function $f(n)$ (*i.e.*, $n^c = o(f(n))$ for all $c > 0$, and the failure probability of $\mathcal{A}$ is bounded by $1/f(n)$).

4.3 The Chernoff-Hoeffding Bounds Extension

In this section we introduce our extension of the Chernoff-Hoeffding bounds which are important tools used in estimating the tail probabilities of random variables. Given n *independent* random variables $X_1, X_2, \ldots, X_n$, these bounds are used in deriving an upper bound on the upper tail probability $Pr(X \geq (1+\epsilon)\mu)$, where $X = \sum_{i=1}^{n} X_i$, $\mu = E[X]$, and $\epsilon > 0$. We extend these bounds to a certain case of dependency among the X_i's, which we call λ–correlation. Since the general definition of λ–correlation is involved, we first introduce this notion in the case where the X_i's are 0-1 random variables and prove our extension of the Chernoff-Hoeffding bounds. This special case is simpler to handle and more intuitive but contains all the essential difficulties of the general case. After the special case is dealt with, we will take care of the more general case.

Before giving our extension of the Chernoff-Hoeffding bounds let us review Chernoff's classical approach to upper bound the upper tail probability of a random variable X, which is the sum of *independent* binary random variables $X_1, X_2, \ldots, X_n$ [Che52]. Chernoff's basic idea is to use Markov's inequality on the random variable e^{tX} for an arbitrary, but fixed, $t > 0$, and minimize with respect to t. That is, to use the fact that

$$\begin{aligned} Pr(X > (1+\epsilon)\mu) &= Pr(e^{tX} > e^{t(1+\epsilon)\mu}) \\ &\leq \frac{E[e^{tX}]}{e^{t(1+\epsilon)\mu}}, \end{aligned}$$

and minimize the last ratio for $t > 0$ [Che52]. This is achieved by finding a good upper bound for the numerator $E[e^{tX}]$ by using the fact that X is the sum of *independent* random variables. It is standard (see, *e.g.*, Raghavan[Rag90]) to use this for showing that in this case,

$$\min_{t>0} \frac{E[e^{tX}]}{e^{t(1+\epsilon)\mu}} \leq F(\mu, \epsilon) \doteq \left[\frac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}} \right]^{\mu}. \tag{4.1}$$

Hoeffding [Hoe63] considers a more general case where X is the sum of n independent and *bounded* random variables $X_i \in [a_i, b_i]$, and uses the above approach to show that if $E[X] = \mu$, then for $\epsilon > 0$,

$$\min_{t>0} \frac{E[e^{tX}]}{e^{t(1+\epsilon)\mu}} \leq G(\mu, \epsilon, \vec{a}, \vec{b}) \doteq \exp\left[-\frac{2\,\mu^2\epsilon^2}{\sum_{i\in[n]}(b_i - a_i)^2}\right]. \tag{4.2}$$

Equations 4.1 and 4.2 will be used in our proofs. Henceforth, we refer to these bounds of Chernoff and Hoeffding as the CH-bounds. If ϵ is a fixed positive quantity no greater than 1 (which will be true in all our applications), then $F(\mu, \epsilon) \leq e^{-\epsilon^2\mu/3}$. Hence, if $\mu = \Omega(\log^{1+\delta} n)$, then $F(\mu, \epsilon)$ is the inverse of a superpolynomial function of n, for any fixed $\delta > 0$ (similar considerations apply to $G(\mu, \epsilon, \vec{a}, \vec{b})$). This fact makes the CH-bounds a powerful tool for deriving strong performance guarantees for randomized algorithms and will be used repeatedly in this paper.

4.3.1 0-1 Random Variables

We now present our extension of the CH-bounds to the case of 0-1 λ–correlated random variables. We begin by defining λ–correlation[1] for 0-1 random variables.

Definition 7 *Let $X_1, X_2, \ldots, X_n$ be binary random variables. The X_i's are λ–*correlated *if, for all nonempty $I \subseteq [n]$,*

$$Pr\left(\bigwedge_{i\in I} X_i = 1\right) \leq \lambda \prod_{i\in I} Pr(X_i = 1).$$

Before proving the extension of the CH-bounds, let us develop some intuition by giving an example of λ–correlation. Suppose we have Δ balls that are thrown uniformly and independently at random into Δ bins. Let B_i be an indicator random variable that is 1 if bin i is empty and 0 otherwise. These variables are 1–correlated. To see this, consider a subset $J \subseteq [\Delta]$ of bins and suppose that all these bins are empty. Then, given this information, the probability that some other bin, say i, remains empty decreases. That is, for all $J \subseteq [\Delta]$,

[1]This was defined as "self–weakening with parameter λ" in [PS92a].

$$\begin{aligned} Pr(B_i = 1 \mid \bigwedge_{j \in J} B_j = 1) &= \left(1 - \frac{1}{\Delta - |J|}\right)^{\Delta} \\ &\leq \left(1 - \frac{1}{\Delta}\right)^{\Delta} \\ &= Pr(B_i = 1) \end{aligned}$$

by straightforward induction, this implies that for all nonempty $I \subseteq [\Delta]$,

$$Pr\left(\bigwedge_{i \in I} B_i = 1\right) \leq \prod_{i \in I} Pr(B_i = 1).$$

The CH-bounds say that if X is the sum of n *independent* 0-1 random variables then $Pr(X > (1+\epsilon)\mu) \leq F(\mu, \epsilon)$. The next theorem shows that if the X_i's are λ–correlated then $Pr(X > (1+\epsilon)\mu) \leq \lambda\ F(\mu, \epsilon)$. In the statement of the following theorem, μ is an upper bound on $E[X]$; this is sufficient because we are upper bounding the upper tail probability of X.

Theorem 13 *Let X be the sum of λ-correlated 0-1 random variables, and let $E[X] \leq \mu$. Then,*

$$Pr(X > (1+\epsilon)\mu) \leq \lambda\ F(\mu, \epsilon).$$

PROOF. As in the classical proof, we start by introducing a positive parameter t and by applying Markov's inequality to the variable e^{tX},

$$\begin{aligned} Pr(X > (1+\epsilon)\mu) &= Pr(e^{tX} > e^{t(1+\epsilon)\mu}) \\ &\leq \frac{E[e^{tX}]}{e^{t(1+\epsilon)\mu}}. \end{aligned}$$

To find a good upper bound on the numerator (we cannot use independence of the X_i's) we wright down its Taylor expansion, and use the fact that e^{tX} is bounded by e^{tn} to apply linearity of expectation to the infinite series:

$$
\begin{aligned}
E[e^{tX}] &= E\left[\sum_{k=0}^{\infty} \frac{t^k X^k}{k!}\right] \\
&= \sum_{k=0}^{\infty} \frac{t^k E[X^k]}{k!}.
\end{aligned}
$$

Focus on the generic term $E[X^k]$ of this series. From the fact that $X = \sum X_i$ and that the X_i's are 0-1 random variables it follows that $E[X^k]$ is a linear combination of the form

$$E[X^k] = \sum_{I \subseteq [n]} \alpha_I \; Pr\left(\bigwedge_{i \in I} X_i = 1\right)$$

for $I \neq \emptyset$, and where all α_I's are non-negative (this can be verified by unfolding X^k and by observing that $X_i^c = X_i$ for all i and all $c \neq 0$). In order to upper bound the term $E[X^k]$ we introduce n twin 0-1 random variables $\hat{X}_i$ which have the same distribution as the X_i's except that they are independent. From the hypothesis of λ–correlation and from the fact the twin variables are independent it follows that

$$
\begin{aligned}
Pr\left(\bigwedge_{i \in I} X_i = 1\right) &\leq \lambda \prod_{i \in I} Pr(X_i = 1) \\
&= \lambda \prod_{i \in I} Pr(\hat{X}_i = 1) \\
&= \lambda \; Pr\left(\bigwedge_{i \in I} \hat{X}_i = 1\right).
\end{aligned}
$$

Hence,

$$E[X^k] \leq \lambda \; E[\hat{X}^k].$$

By upper bounding each term of the series expansion of $E[e^{tX}]$ we get

$$E[e^{tX}] \leq \lambda \; E[e^{t\hat{X}}].$$

Notice now that $\hat{X}$ is the sum of n *independent* random variables and $E[\hat{X}] = E[X] \leq \mu$. Hence, by equation 4.1

$$\begin{aligned} Pr(X > (1+\epsilon)\mu) &\leq \frac{E[e^{tX}]}{e^{t(1+\epsilon)\mu}} \\ &\leq \lambda \frac{E[e^{t\hat{X}}]}{e^{t(1+\epsilon)\mu}} \\ &\leq \lambda\, F(\mu,\epsilon). \end{aligned}$$

□

To see an application of this theorem, let us go back to the balls-and-bins experiment, and suppose we are interested in estimating the number of empty bins after the balls are thrown; this number is given by the random variable $B = \sum_i B_i$. By linearity of expectation and by the fact that the balls are thrown at random independently of each other,

$$\begin{aligned} E[B] &= \sum_{i\in[\Delta]} E[B_i] \\ &= \sum_{i\in[\Delta]} \left(1 - \frac{1}{\Delta}\right)^{\Delta} \\ &\leq \frac{\Delta}{e}. \end{aligned}$$

We already saw that the B_i's are 1–correlated. Hence, by Theorem 13,

$$Pr(B > (1+\epsilon)\Delta/e) \leq F(\Delta/e, \epsilon).$$

Remark. Jain has proved the following lemma [Ram90]: *Let $a_1, a_2, \ldots, a_n$ be n random trials (not necessarily independent) such that the probability that trial a_i 'succeeds' is bounded above by p regardless of the outcomes of the other trials. Then if X is the random variable that represents the number of 'successes' in these n trials, and Y is a binomial variable with parameters (n,p), then: $Pr[X \geq k] \leq Pr[Y \geq k]$, $0 \leq k \leq n$.*

The assumptions of Jain's lemma are strictly stronger than those of 1–correlation. For instance, in the balls and bins example,

$$Pr(B_\Delta = 1 | \bigwedge_{i\in[\Delta-1]} B_1 = 0) = \frac{\Delta - 1}{\Delta + 1},$$

which, for $\Delta \geq 3$, is greater than $Pr(B_\Delta = 1)$ ($\approx 1/e$).

4.3.2 The General Case

In this section we introduce the more general definition of λ-correlation and prove the more general extension of the CH-bounds.

The proof of Theorem 13 is based on the observation that if we can upper bound each term $E[X^k]$ of the Taylor expansion of $E[e^{tX}]$ by $\lambda\ E[\hat{X}^k]$ where $\hat{X}$ is the sum of independent random variables, and if $E[e^{t\hat{X}}] \leq B$, then $E[e^{tX}] \leq \lambda\ B$. This motivates the following definition.

Definition 8 *Let $X_1, X_2, \ldots, X_n$ be bounded random variables such that $X_i \in [a_i, b_i]$ and let $X = \sum_{i\in[n]} X_i$. The X_i's are λ-correlated if there exists a collection of independent twin random variables $\hat{X}_i \in [a_i, b_i]$ such that,*

- *$E[X] \leq E[\hat{X}]$, where $\hat{X} = \sum_{i\in[n]} \hat{X}_i$, and*
- *for all nonempty $I \subseteq [n]$ and positive integers $s_1, s_2, \ldots, s_{|I|}$,*

$$E[\prod_{i\in I} X_i^{s_i}] \leq \lambda \prod_{i\in I} E[\hat{X}_i^{s_i}].$$

If we apply this definition to the case of 0-1 random variables we see that to have λ–correlated 0-1 random variables, it suffices to find twin variables $\hat{X}_i$ such that $E[X] \leq E[\hat{X}]$ and, for all nonempty $I \subseteq [n]$,

$$Pr\left(\bigwedge_{i\in I} X_i = 1\right) \leq \lambda \prod_{i\in I} Pr(\hat{X}_i = 1)$$

which reduces to definition 7 when $Pr(X_i = 1) = Pr(\hat{X}_i = 1)$. Notice that for this to hold, it is not necessary that $Pr(X_i = 1) = Pr(\hat{X}_i = 1)$; in typical applications we do not know the exact value of $Pr(X_i = 1)$ but only an approximation (usually an upper bound; we will see examples in later sections), but this is good enough to apply the CH-bounds extension.

Theorem 14 *Let X be the sum of λ–correlated random variables $X_1, X_2, \ldots, X_n$, where $X_i \in [a_i, b_i]$ and let $\hat{X}$ be the sum of the n twin variables $\hat{X}_i$. Then,*

$$Pr(X > (1+\epsilon)\ E[\hat{X}]) \leq \lambda\ G(\mu, \epsilon, \vec{a}, \vec{b}).$$

PROOF. Let $\mu = E[\hat{X}]$. As in the proof of Theorem 13, we start with the inequality

$$\begin{aligned} Pr(X > (1+\epsilon)\mu) &= Pr(e^{tX} > e^{t(1+\epsilon)\mu}) \\ &\leq \frac{E[e^{tX}]}{e^{t(1+\epsilon)\mu}}. \end{aligned}$$

By the hypotheses of boundedness and of λ–correlation it follows that

$$\begin{aligned} E[e^{tX}] &= E\left[\sum_{k=0}^{\infty} \frac{t^k X^k}{k!}\right] \\ &= \sum_{k=0}^{\infty} \frac{t^k E[X^k]}{k!} \\ &\leq \lambda \sum_{k=0}^{\infty} \frac{t^k E[\hat{X}^k]}{k!} \\ &= \lambda\ E[e^{t\hat{X}}] \end{aligned}$$

By the already discussed result of Hoeffding [Hoe63], when $\hat{X}$ is the sum of n independent bounded random variables $\hat{X}_i \in [a_i, b_i]$

$$\min_{t>0} \frac{E[e^{t\hat{X}}]}{e^{t(1+\epsilon)\mu}} \leq G(n, \epsilon, \vec{a}, \vec{b}).$$

□

In this paper we will use the special case of Theorem 14 where $X_i \in [0,1]$, $i \in [n]$. In this case, $F(\mu, \epsilon)$ is also an upper bound for the upper tail of X.

Corollary 5 *Let X be the sum of n λ-correlated random variables $X_i \in [0,1]$. Then,*

$$Pr(X > (1+\epsilon)E[\hat{X}]) \leq \lambda\ F(\mu,\epsilon).$$

PROOF. Let $E[\hat{X}] = \mu$. When $\hat{X}$ is the sum of n independent random variables $\hat{X}_i \in [0,1]$ Hoeffding shows that, for $t \in (0, 1-\mu/n)$, (cf. Theorem 1 of [Hoe63])

$$Pr\left(\frac{\hat{X}-\mu}{n} \geq t\right) \leq \left(\frac{\mu}{\mu+nt}\right)^{\mu+nt}\left(1+\frac{n\,t}{n-\mu-nt}\right)^{n-\mu-nt}.$$

By setting $\epsilon = nt/n$ and by applying the standard approximation $1+x \leq e^x$ for $x = nt/(n-\mu-nt)$ we get

$$Pr(X \geq (1+\epsilon)\mu) \leq \left(\frac{e^{\epsilon}}{(1+\epsilon)^{(1+\epsilon)}}\right)^{\mu} = F(\mu,\epsilon).$$

□

4.4 The Edge Coloring Algorithm

We now present our randomized distributed edge coloring algorithm. The algorithm uses an idea of Karloff & Shmoys to reduce the problem of edge coloring general graphs to that of edge coloring a special class of bipartite graphs [KS87].

The Karloff & Shmoys algorithm was proposed for the PRAM model and is as follows. The input to the algorithm is a graph $G = (V,E)$ of maximum degree $\Delta(G) = \Delta$.

1. Compute a random partition of V into black and white vertices (all vertices flip a fair coin independently and in parallel). Let $G[B]$ be the subgraph induced by the black vertices, $G[W]$ be the subgraph induced by the white vertices, and $G[B,W]$ be the bipartite subgraph formed by the edges having endpoints of different colors.

2. Optimally edge color $G[B, W]$, *i.e.*, with $\Delta(G[B, W])$ colors, using the PRAM algorithm of Lev, Pippenger & Valiant [LPV81].

3. Recurse on $G[B]$ and $G[W]$ using the same set of fresh new colors on both graphs.

The key fact is that, as long as Δ is large enough, the maximum degree of the three subgraphs is at most $\Delta/2 + \Delta^{1/2+\epsilon} = \Delta/2 + o(\Delta)$ *w.h.p.*, for any fixed $\epsilon > 0$. (This can be proved by applying the standard CH-bounds.) Note that the *same* set of fresh new colors can be used on both $G[B]$ and $G[W]$ because $G[B, W]$ completely separates the two graphs. The recurrence for the number of colors used is

$$\begin{aligned} C(\Delta) &\leq \left(\frac{\Delta}{2} + \Delta^{1/2+\epsilon}\right) + C\left(\frac{\Delta}{2} + \Delta^{1/2+\epsilon}\right) \\ &\leq \Delta + o(\Delta) \end{aligned}$$

and holds *w.h.p.* The "high probability" statement holds as long as $\Delta = \Omega(\log^{1+\delta} n)$, for any fixed $\delta > 0$. In this case, the failure probability given by the CH-bounds is the inverse of a superpolynomial function. For the case when the degree goes below the $\Omega(\log^{1+\delta})$ threshold, Karloff & Shmoys give a PRAM algorithm to $(\Delta + 1)$–edge color graphs of this low maximum degree [KS87].

The non–distributed parts of the Karloff & Shmoys scheme are the subroutine which colors the bipartite graph optimally (it can be shown that this is impossible in the distributed model of computation [PS92b]), and the handling of the case $\Delta \leq \log^{1+\delta} n$. The latter can be handled by Luby's randomized distributed $(\Delta+1)$–*vertex* coloring algorithm [Lub88]. Luby's algorithm is simulated on the line graph of the given graph G thus giving a $2\Delta(G) - 1$ edge coloring[2]. Hence, if we could

[2]When $\Delta(G) = O(polylog(n))$ we can compute a $2\Delta(G) - 1$ colorings *deterministically* in $O(polylog(n))$ time with an algorithm based on the idea of removing maximal matchings. We prefer to use Luby's algorithm here for conciseness.

get a good distributed algorithm for bipartite edge coloring then we could use the Karloff & Shmoys scheme to get a good coloring of any graph. We now show how this can be achieved.

4.4.1 Distributed Edge Coloring of Bipartite Graphs

We now present a simple Monte Carlo algorithm for edge coloring the special type of bipartite graphs generated by the Karloff & Shmoys scheme.

Given a bipartite graph $G = (A, B, E)$ we assume that each vertex knows whether it belongs to A or B. This is an important assumption because such information cannot be computed fast distributively [PS92b], but it is verified by the bipartite graphs generated by the Karloff & Shmoys scheme. From now on, we will refer to vertices in A as the *top* vertices and to the vertices in B as the *bottom* vertices.

The algorithm takes $O(\log n)$ time and colors a bipartite graph G of maximum degree Δ with at most $1.6\Delta + 16\log^{2+\delta} n$ colors *w.h.p.*, for any $\delta > 0$, which is approximately $1.6\Delta + \log^2 n$ (the failure probability will depend on δ). In the algorithm we use a variable Δ_C that is initialized to $\Delta(G)$. During any iteration of the algorithm, Δ_C is meant to be an upper bound on the degree of the current graph; we will prove later that this holds *w.h.p.* Recall that $\delta(u)$ denotes the set of edges incident on vertex u, and let THRESHOLD $= \log^{2+\delta} n$.

The algorithm is given two parameters $\epsilon, \delta > 0$, and is as follows:

1. PART I: Repeat until $\Delta_C <$ THRESHOLD:

   Let G_C be the current graph. Pick a set χ of Δ_C fresh new colors.

   - *(Random proposal of bottom vertices)* In parallel and independently of the other vertices in B, each vertex $v \in B$ assigns a temporary color to each edge in $\delta(v)$ with uniform probability *without replacement, i.e.* edge e_1 is assigned color $\alpha \in \chi$ with probability $1/\Delta_C$, e_2 is assigned $\beta \in \chi - \{\alpha\}$ with probability $1/(\Delta_C - 1)$ and so on.

- *(Lottery of top vertices)* (Comment: The coloring so far is consistent around any vertex $v \in B$ but can be inconsistent around a vertex $u \in A$.) For each $u \in A$, let $C_u(\alpha)$ be the set of edges in $\delta(u)$ with temporary color α. Each vertex $u \in A$ selects a *winner* uniformly at random in $C_u(\alpha)$, for each nonempty $C_u(\alpha)$. All other edges, the *losers*, are decolored and assigned $\perp$.
- Set $\Delta_C := \Delta_C(1+\epsilon)/e$. $G_\perp$, the subgraph of G_C induced by the losers (*i.e.*, by the $\perp$-edges), becomes the new current graph.

2. PART II: Let G_r be the remaining graph. Edge color G_r with $2\Delta(G_r) - 1 <$ 2 THRESHOLD colors by executing Luby's vertex coloring algorithm on the line graph of G_r [Lub88].

The key claim is that in every iteration of part (I) above, the maximum degree of the graph shrinks by a factor of at least $(1+\epsilon)/e$ *w.h.p.*, as long as $\Delta \geq$ THRESHOLD. That is

$$\Delta(G_\perp) \leq (1+\epsilon)\,\frac{\Delta(G_C)}{e}$$

with high probability, for any fixed $\epsilon > 0$. The condition $\Delta_C \geq$ THRESHOLD ensures that the failure probability given by the extension of the CH-bounds is the inverse of a superpolynomial function. The reason for setting THRESHOLD $= \log^{2+\delta} n$ will be apparent from the probabilistic analysis.

Once the key claim is established, we can bound both the total number of colors used and the running time of the algorithm. To bound the number of colors used observe that if the degree of the graph shrinks at every iteration by at least a $(1+\epsilon)/e$ factor *w.h.p.* then the maximum degree of G_r is at most $\log^{2+\delta} n$ *w.h.p.* Hence, *w.h.p.*, the number of colors used is at most

$$C(\Delta) \leq \Delta + \frac{\Delta}{e}(1+\epsilon) + \ldots + \frac{\Delta}{e^k}(1+\epsilon)^k + 2\log^{2+\delta} n,$$

where k is the smallest integer such that $\Delta(1+\epsilon)^k/e^k \leq \log^{2+\delta} n$. The total number of colors used is at most (here, ϵ' depends on ϵ and can be made arbitrarily small)

$$\begin{aligned} C(\Delta) &\leq \left(\frac{e}{e-1}+\epsilon'\right)\Delta + \left(2-\frac{e}{e-1}-\epsilon'\right)\log^{2+\delta} n \\ &< 1.585\ \Delta + 0.4\ \log^{2+\delta} n \\ &< 1.59\ \Delta \end{aligned}$$

when $\Delta > 8\ \log^{2+\delta} n$. The running time of the algorithm is $O(\log n)$ because part (I) takes $O(\log \Delta)$ time (by the key claim), and part (II), *i.e.*, Luby's algorithm, takes $O(\log n)$ time.

If $\Delta > 8\ \log^{2+\delta} n$ and we use our distributed subroutine for bipartite graphs in the Karloff & Shmoys algorithm the total number of colors is given by the recurrence

$$\begin{aligned} T(\Delta) &\leq C\left(\frac{\Delta}{2}+\Delta^{1/2+\epsilon}\right) + T\left(\frac{\Delta}{2}+\Delta^{1/2+\epsilon}\right) \\ &< 1.59\ \Delta + o(\Delta) \\ &< 1.6\ \Delta\ . \end{aligned}$$

If $\Delta \leq 8\ \log^{2+\delta} n$, we apply Luby's algorithm directly to get an edge coloring with $2\ \Delta - 1 \leq 16\ \log^{2+\delta} n$. Hence, the total number of colors to color *any* graph is at most $1.6\ \Delta + 16\ \log^{2+\delta} n$, for any $\delta > 0$, which is approximately $1.6\ \Delta + \log^2 n$ colors.

The rest of the paper is devoted to establishing the key claim.

4.5 Probabilistic Analysis

This section is devoted to proving the key claim of the preceding section, namely that given a graph G and Δ such that $\Delta \geq \Delta(G)$ and $\Delta \geq$ THRESHOLD then, after one iteration of Part (I) of the bipartite algorithm, the maximum degree of the new graph, $\Delta(G_\perp)$, is at most $(1+\epsilon)\Delta/e$ *w.h.p.*, for any fixed $\epsilon > 0$.

It turns out that the analysis is considerably easier for the top vertices than for the bottom vertices. We begin with the easy part.

4.5.1 Analysis of Top Vertices

Let u be a generic top vertex with neighbors v_i, $i \in [d_u]$, and incident edges $i = (u, v_i)$. Recall that a *loser* is an edge which, after having got a tentative color in the random proposal, lost the lottery and got decolored. So, the new degree of u is given by the number of losers incident with it.

From the point of view of a top vertex, the random proposal and the lottery are equivalent to the following random experiment. For each edge i incident on u we introduce a ball i, and for each color k we introduce a bin k; the assignment of a tentative color to an edge by the algorithm is equivalent to throwing each ball into one of the Δ bins independently and uniformly at random, since the bottom vertices assign tentative colors with uniform probability and independently of the other bottom vertices. Recalling that we have at most Δ balls and exactly Δ bins:

$$\begin{aligned} \sharp losers &= \sharp balls - \sharp winners \\ &\leq \sharp bins - \sharp nonempty\ bins \\ &= \sharp empty\ bins. \end{aligned}$$

Let X be a random variable denoting the number of losers. To estimate X and its tail distribution we will study the random variable $B = \sharp empty\ bins$. For this purpose, we introduce Δ many indicator random variables

$$B_i = \begin{cases} 1 & \text{bin } i \text{ is empty} \\ 0 & \text{otherwise} \end{cases}$$

and hence, $B = \sum_{i\in[\Delta]} B_i$. Notice that $X \leq B$ always. The variable B was studied in Section 4.3 where it was shown that $E[B] \leq \Delta/e$ and that the B_i's are 1-correlated, which implies that $Pr(B > (1+\epsilon)\Delta/e) \leq F(\Delta/e, \epsilon)$. Since $E[X] \leq E[B]$ and $Pr(X > (1+\epsilon)\Delta/e) \leq Pr(B > (1+\epsilon)\Delta/e)$ we get

Theorem 15 *Let u be a top vertex and X be the random variable denoting the number of losers incident on it. Then, $E[X] \leq \Delta/e$ and*

$$Pr(X > (1+\epsilon)\Delta/e) \leq F(\Delta/e, \epsilon).$$

4.5.2 Analysis of Bottom Vertices

In this section we analyze what happens to the new degree of a generic bottom vertex v_B. This case is considerably harder to handle than the previous one, because of the way in which the random variables describing the process are correlated. For a top vertex, the dependency among the variables was playing for us; given that some edges incident on a top vertex are losers, the probability of having another loser decreases. For a bottom vertex the situation is reversed: having some edges losing the lottery might even make the probability of having another loser increase. The problem can be seen in Figure 4.1. Suppose we are given that e_1 got tentative color α and lost the lottery, and that e_2 got tentative color β; we will argue intuitively that given this, the probability of e_2 losing the lottery has increased. Since e_1 lost, the probability of e_3 getting tentative color α increases, which implies that the probability of e_4 getting tentative color β also increases, and this increases the probability of e_2 losing the lottery.

For the sake of the analysis we modify the algorithm as follows: instead of performing all random proposals in parallel, suppose that the bottom vertices perform their random proposals sequentially, one after the other. This does not modify the probability distributions because the choices are still done independently. We want to focus our attention on the last vertex v_B performing the random proposal. We will use the fact that when v_B performs its random proposal, all edges not incident on v_B already have a tentative color. By symmetry, any upper bound on the probabilities we can find for v_B will hold for all bottom vertices.

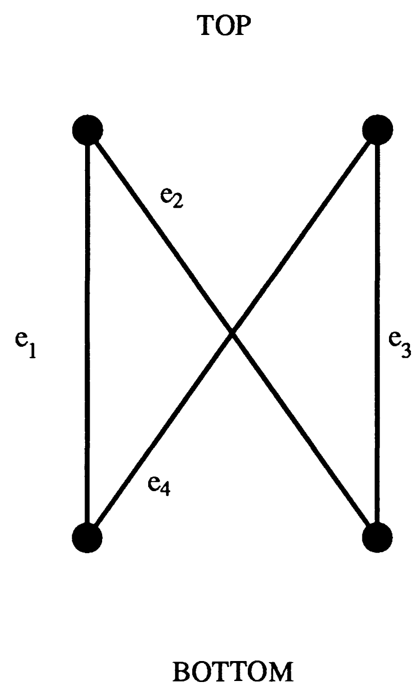

Figure 4.1: $\chi(e_1) = \alpha$, $\chi(e_2) = \beta$, e_1 lost the lottery

Let $i \in [d_v]$ denote an edge incident with the bottom vertex v_B, with the other endpoint of i being u_i. We introduce the indicator random variables

$$X_i = \begin{cases} 1 & i \text{ loses the lottery} \\ 0 & \text{otherwise} \end{cases}$$

and want to study the expectation and tail probability distribution of the random variable $X = \sum_{i \in [d_v]} X_i$. Computing the expectation is easy.

Lemma 8 $E[X] \leq \Delta/e$.

PROOF. Let v_B be the bottom vertex. It is sufficient to show that $Pr(X_i = 1) \leq 1/e$, for all $i \in [d_v]$. From the analysis of the top vertices, we know that the expected number of losers incident with u_i is at most Δ/e and hence that $\sum_{j \in \delta(u_i)} Pr(j \text{ loses}) \leq \Delta/e$. By symmetry, $Pr(j) \leq 1/e$ for all $j \in \delta(u_i)$, and hence $Pr(X_i = 1) \leq 1/e$. $\square$

We now study the tail probability distribution of X. Our goal is to show that $X \leq (1+\epsilon)\Delta/e$ *w.h.p.*, for any fixed $\epsilon > 0$. Establishing this claim will take several lemmas.

For technical reasons that will be clear later, we subdivide the edges incident on v_B into $\sqrt{\Delta}$ groups, each of at most $\sqrt{\Delta}$ edges. Let G be any such group and define $X_G = \sum_{i \in G} X_i$; we would like to show that X_G is the sum of λ–correlated random variables, so that

$$\forall \epsilon > 0, \;\; Pr(X_G > (1+\epsilon)\sqrt{\Delta}/e) \leq \lambda \; F(\sqrt{\Delta}/e, \epsilon).$$

Rather than proving this claim, we will prove a weaker one namely that X_G is the sum of λ–correlated random variables *with high probability!* More precisely, we will define an event $\mathcal{A}$ such that $\mathcal{A}$ happens *w.h.p.*, and such that, given that $\mathcal{A}$ occurs, then X_G is the sum of λ–correlated random variables. That is,

$$Pr(X_G > (1+\epsilon)\sqrt{\Delta}/e \;\mid\; \mathcal{A}) \leq \lambda \; F(\sqrt{\Delta}/e, \epsilon).$$

Showing this weaker claim is sufficient because

$$\begin{aligned}
Pr(X > (1+\epsilon)\Delta/e) &\leq Pr(\exists X_G > (1+\epsilon)\sqrt{\Delta}/e) \\
&= Pr(\exists G \; X_G > (1+\epsilon)\sqrt{\Delta}/e \mid \mathcal{A}) \; Pr(\mathcal{A}) \; + \\
&\quad\; Pr(\exists G \; X_G > (1+\epsilon)\sqrt{\Delta}/e \mid \mathcal{A}^c) \; Pr(\mathcal{A}^c) \\
&\leq Pr(\exists G \; X_G > (1+\epsilon)\sqrt{\Delta}/e \mid \mathcal{A}) \; + \; Pr(\mathcal{A}^c) \\
&\leq \sqrt{\Delta} \; Pr(X_G > (1+\epsilon)\sqrt{\Delta}/e \mid \mathcal{A}) \; + \; Pr(\mathcal{A}^c) \\
&\leq \sqrt{\Delta} \; \lambda \; F(\sqrt{\Delta}/e, \epsilon) \; + \; Pr(\mathcal{A}^c).
\end{aligned}$$

Another interesting point in the analysis is that λ will not be a constant term but an exponentially growing function. However we will be able to show the rate of growth of λ is much slower than that of $1/F(\sqrt{\Delta}/e, \epsilon)$. In other words, we will show that $\lambda \; F(\sqrt{\Delta}/e, \epsilon)$ goes to zero superpolynomially fast.

We now turn to the task of showing that X_G is the sum of λ-correlated random variables *w.h.p.* Recall from the definition that the X_i's, $i \in G$, are λ-correlated if, for all nonempty $I \subseteq G$,

$$Pr\left(\bigwedge_{i\in I} X_i = 1\right) \leq \lambda \prod_{i\in I} Pr(X_i = 1).$$

Consider then a generic subset $I \subseteq G$ of edges incident on the bottom vertex v_B, and let us see how to compute $Pr\left(\bigwedge_{i\in I} X_i = 1\right)$. Recall that we are analyzing the situation where v_B is the last vertex to perform its random proposal. This means that prior to the assignment of a tentative color to edge $i = (v, u_i)$, all other edges incident on u_i already have their tentative color. Using the balls-and-bins language, we can say that prior to throwing ball i at random into one of the bins, all balls coming from the other neighbors of u_i have been thrown. We will think of i as a red ball and of the other edges at u_i as white balls. Once the red ball is thrown in, say, bin $k \in [\Delta]$, a winner is selected uniformly at random among all (*i.e.*, red *and* white) balls in bin k. All other balls, the losers, are discarded. Notice that the probability of discarding the red ball is itself a random variable which depends on the particular placement of the white balls prior to throwing the red ball.

Given any placement of white balls at u_i, we construct a vector of probabilities C_i as follows. Let a_{ik} denote the number of white balls in bin $k \in [\Delta]$ of vertex u_i, and let $p_{ik} = a_{ik}/(1 + a_{ik})$ denote the probability that the red ball loses the lottery given that it was thrown in bin k (equivalently, p_{ik} is the probability that edge i loses, given that it got tentative color k). For each neighbor u_i of our bottom vertex v_B, we construct the corresponding vector $C_i = (p_{i1}, p_{i2}, \ldots, p_{i\Delta})$. Given a set I of neighbors of v_B we construct the matrix M_I whose i-th column is the vector C_i. The next lemma explains why this matrix is relevant to us. From now on, let $p(m, k) \doteq m(m-1)\cdots(m-k+1)$.

Lemma 9 *Let $I \subseteq G$ and $k = |I|$. Then,*

$$Pr\left(\bigwedge_{i\in I} X_i = 1\right) = \frac{perm(M_I)}{p(\Delta, k)}.$$

PROOF. The random proposal of v_B is equivalent to choosing an one-to-one function $\pi : I \rightarrow [\Delta]$ uniformly at random among the set $\mathcal{P}$ of all such functions. Recall that the entry M_{ik} of M_I is the probability p_{ik} that edge i loses given that it is given color k. Hence,

$$\begin{aligned} Pr(\wedge_{i\in I} X_i = 1) &= \sum_{\pi\in\mathcal{P}} Pr(\wedge_{i\in I} X_i = 1 \mid \pi \, is\ selected)\ Pr(\pi \, is\ selected) \\ &= \sum_{\pi\in\mathcal{P}} \left(M_{1,\pi(1)} \cdots M_{k,\pi(k)}\right) \left(\frac{1}{\Delta} \cdot \frac{1}{\Delta - 1} \cdot \ldots \cdot \frac{1}{\Delta - k + 1}\right) \\ &= \frac{perm(M_I)}{p(\Delta, k)}. \end{aligned}$$

□

We now want to find a good upper bound of $perm(M_I)$. The following lemma gives a simple upper bound that is sufficient for our purposes. Given a matrix M let S_i denote the sum of the elements of C_i, the i-th column of M.

Lemma 10 *Let $M = [M_{ik}]$ be a matrix with c columns and r rows, and non-negative entries M_{ik}. Then, $perm(M) \leq \prod_{i\in[c]} S_i$.*

PROOF. Let $\mathcal{P} = \{\pi \mid \pi : [c] \rightarrow [r],\ \pi \text{ is one-to-one}\}$. Then,

$$\begin{aligned} perm(M) &= \sum_{\pi\in\mathcal{P}} M_{1,\pi(1)}\, M_{2,\pi(2)} \cdots M_{c,\pi(c)} \\ &\leq (M_{1,1} + \ldots + M_{1,r})(M_{2,1} + \ldots + M_{2,r}) \ldots (M_{c,1} + \ldots + M_{c,r}) \\ &= \prod_{i\in[c]} S_i. \end{aligned}$$

□

The next lemma relates the value of S_i to that of $1/e \geq Pr(i \text{ loses})$, $i \in \delta(v)$. It is an application of the most general definition of λ–correlation. Before the proof of the lemma, we establish

Proposition 1 *If $0 \leq p \leq 1$, $q = 1 - p$ and m is a positive integer, then*

$$\sum_{r=1}^{m} \binom{m}{r} p^r q^{m-r} \frac{r}{r+1} = 1 - \frac{(1 - q^{m+1})}{p(m+1)}.$$

PROOF. Let

$$\begin{aligned} f(p) &= \sum_{r=1}^{m} \binom{m}{r} p^r q^{m-r} \frac{k}{k+1} \\ &= 1 - q^m - \sum_{r=1}^{m} \binom{m}{r} p^r q^{m-r} \frac{1}{k+1}. \end{aligned}$$

Integrating both sides of the binomial expansion

$$(x+q)^m = \sum_{r=0}^{m} \binom{m}{r} x^r q^{m-r}$$

between 0 and p, we get

$$\frac{1 - q^{m+1}}{m+1} = p\,(1 - f(p)),$$

from which the proposition follows. □

We now return to our scenario where v_B is the last bottom vertex to pick tentative colors for its edges, with its edges having been split into groups. Recall that we are focusing on a particular group G, and that we want a good upper bound on $Pr\left(\bigwedge_{i \in I} X_i = 1\right)$, for any nonempty $I \subseteq G$. Lemma 9 gives the upper-bound $Pr\left(\bigwedge_{i \in I} X_i = 1\right) \leq perm(M_I)/p(\Delta, |I|)$, and Lemma 10 gives the upper-bound $perm(M_I) \leq \Pi S_i$. So, a good upper-bound of S_i will hopefully translate into a good upper-bound for $Pr\left(\bigwedge_{i \in I} X_i = 1\right)$. The next lemma says that $S_i \leq \Delta/e$

w.h.p. Recall that we are analyzing the situation where v_B is to give tentative colors for its edges and that all other edges have already been assigned tentative colors. If we focus on a particular edge $i = (v_B, u_i)$ the situation is equivalent to throwing a red ball i uniformly at random into Δ bins where each bin has some number of white balls in it (possibly zero). Notice that the number of white balls in each bin is itself a random variable.

Lemma 11 *Let $i = (v, u_i)$ be any edge in $I \subseteq G$, and S_i be the sum of the elements of C_i, the i-th column of M_I. Then, $E[S_i] \leq \Delta/e = \mu$ and*

$$\forall \epsilon_1 > 0, \quad Pr(S_i > (1+\epsilon_1)\mu) \leq F(\mu, \epsilon_1).$$

PROOF. Let Z_k be the random variable denoting the number of white balls in bin k and $Y_k = Z_k/(Z_k+1)$ be the random variable denoting the probability that the red ball loses the lottery given that it ends in bin k. Then, $S_i = Y \doteq \sum_k Y_k$. Note that the Y_k's are bounded random variables with values in $[0,1]$. We will show that $E[Y] \leq \Delta/e$ and that the Y_k's are 1–correlated (under the most general definition of λ–correlation), which will give our claim.

Firstly, we may assume that we have $d = \Delta - 1$ white balls (*i.e.*, the degree of u_i is Δ): $Pr(Y \geq (1+\epsilon_1)\Delta/e)$ is maximized at $d = \Delta - 1$, as d varies from 1 to $\Delta - 1$. (To see this, assume $d = \Delta - 1 - k < \Delta - 1$. Add k *yellow* balls to the white balls, and run two experiments. In one experiment throw the white and red balls and compute the probability that the red ball loses the lottery. In the other experiment, throw white, yellow and red balls and again compute the probability that the red ball loses. In both experiments, let us look at the bin where the red ball fell. The probability that the red ball loses is $b/(b+1)$ for the first experiment, and $(b+y)/(b+y+1)$ for the second, where b and y are, respectively, the number of white and yellow balls in the bin. Since $y \geq 0$, $b/(b+1) \leq (b+y)/(b+y+1)$. If $Y_i(d)$ indicates the variable Y_i when u_i has degree d, then $Y_i(d) \leq Y_i(\Delta)$ for all $i \in [\Delta]$ and $d \in [\Delta]$.)

First, we will show that, for all i, $E[Y_i] \leq 1/e$ and then we will show that, for any set of k indices $I \subseteq [\Delta]$ and strictly positive integers s_i,

$$E[\prod_{i \in I} Y_i^{s_i}] \leq 1/e^k. \tag{4.3}$$

Given this we can apply Corollary 5 by introducing n independent twin 0-1 random variables $\hat{Y}_i$ such that $E[\hat{Y}_i] = Pr(\hat{Y}_i = 1) = 1/e$. Since the $\hat{Y}_i$'s are binary, equation 4.3 is the same as

$$E[\prod_{i \in I} Y_i^{s_i}] \leq \prod_{i \in I} E[\hat{Y}_i] = \prod_{i \in I} E[\hat{Y}_i^{s_i}],$$

which is to say that the Y_i's are 1–correlated. Noting that $0 \leq Y_i \leq 1$, it suffices to show that

$$E[\prod_{i \in I} Y_i] \leq 1/e^k. \tag{4.4}$$

Without loss of generality we can assume $I = [k]$. We will prove inequality 4.4 by induction on $k \geq 1$; when $k = 1$,

$$\begin{aligned} E[Y_1] &= \sum_{r=0}^{\Delta-1} \binom{\Delta-1}{r} \left(\frac{1}{\Delta}\right)^r \left(1 - \frac{1}{\Delta}\right)^{\Delta-1-r} \frac{r}{r+1} \\ &= (1 - 1/\Delta)^{\Delta} \leq 1/e, \end{aligned}$$

where the second equality follows from Proposition 1. Notice that for all $j \in [\Delta]$, $E[Y_j] = E[Y_1] \leq 1/e$. When $k > 1$, the law of conditional probabilities gives

$$E[\prod_{i \in [k]} Y_i] = E[Y_1 Y_2 \cdots Y_{k-1}\, E[Y_k \mid Y_1 Y_2 \cdots Y_{k-1}]] \tag{4.5}$$

Suppose we show that for all *non-zero* $c_i \in [0, 1]$ with $i \in [k-1]$,

$$E[Y_k \mid \bigwedge_{i=1}^{k-1} Y_i = c_i] \leq \frac{1}{e}; \tag{4.6}$$

then, since the product $Y_1Y_2\cdots Y_{k-1}$ in equation (4.5) is zero when any c_i is zero, we see by induction on k that

$$\begin{aligned} E[\prod_{i=1}^{k} Y_i] &= E[Y_1Y_2\cdots Y_{k-1}\, E[Y_k \mid Y_1Y_2\cdots Y_{k-1}]] \\ &\leq \frac{1}{e} E[\prod_{i=1}^{k-1} Y_i] \\ &\leq \frac{1}{e^k}. \end{aligned}$$

Hence, the claim follows if we can show that inequality (4.6) holds.

If a_i denotes the number of white balls that fell into bin i, then $c_i = a_i/(a_i+1)$. Let $a = \sum_{i=1}^{k-1} a_i \geq k-1$, $p = 1/(\Delta - k + 1)$, and $q = 1 - p$. Then

$$\begin{aligned} E[Y_k| \bigwedge_{i=1}^{k-1} Y_i = c_i] &= E[Y_k| \bigwedge_{i=1}^{k-1} Z_i = a_i] \\ &= \sum_{r=1}^{\Delta-1-a} t(r,a), \end{aligned}$$

where

$$t(r,a) \doteq \binom{\Delta - 1 - a}{r} p^r q^{\Delta-1-a-r} \frac{r}{r+1}.$$

It is easy to check that $t(r,a) \geq t(r,a+1)$. As a consequence, the maximum value of $E[Y_k| \bigwedge_{i=1}^{k-1} Y_i = c_i]$ is attained at $a = k-1$, in which case we have

$$\begin{aligned} \sum_{r=1}^{\Delta-1-a} t(r,a) &= \sum_{r=1}^{\Delta-k} t(r,k-1) \\ &= \sum_{r=1}^{\Delta-k} \binom{\Delta - k}{r} p^r q^{\Delta-k-r} \frac{r}{r+1} \\ &= q^{\Delta-k+1} \leq 1/e, \end{aligned}$$

by Proposition 1. $\square$

Putting all the pieces together (recall that $k = |I|$):

$$\begin{aligned} Pr(\wedge_{i \in I} X_i = 1) &= \frac{perm(M_I)}{p(\Delta, k)} \\ &\leq \frac{\prod_{i \in I} S_i}{p(\Delta, k)} \\ &\leq (1+\epsilon_1)^k \frac{\Delta^k}{e^k} \frac{1}{p(\Delta, k)} \\ &\leq (1+\epsilon_1)^k \frac{\Delta^k}{p(\Delta, k)} \frac{1}{e^k}, \end{aligned}$$

for any $\epsilon_1 > 0$. The first line follows from Lemma 9, the second from Lemma 10, and the third from Lemma 11 and holds *w.h.p.* The last line is just a re-writing of the third one. Given $\epsilon_1 > 0$, let $\mathcal{A}_{\epsilon_1}$ be the event that, for all $i \in G$, $S_i \leq (1+\epsilon_1)\Delta/e$. Lemma 11 says that $\mathcal{A}_{\epsilon_1}$ happens *w.h.p.*, for any $\epsilon_1 > 0$. The last line of the above chain of inequalities means that, given that $\mathcal{A}_{\epsilon_1}$ occurred, the X_i's are λ-correlated with $\lambda = (1+\epsilon_1)^k \Delta^k / p(\Delta, k)$. (Alternatively, we can say that the X_i's are λ–correlated *w.h.p.*) We now want an estimate of λ. The next lemma explains why we subdivided the edges incident on v_B into groups.

Lemma 12 *For positive integers t and k, $t^k/p(t,k) \leq e^{k^2/t}$, if $k \leq t/2$.*

PROOF. We first note that $\ln(1-x) \geq -2x$, for $0 \leq x \leq 1/2$. This is true, since if we define $f(x) \doteq \ln(1-x) + 2x$, then $f(0) = 0$ and $f'(x) = \frac{1-2x}{1-x}$, which is non-negative for $0 \leq x \leq 1/2$. Now,

$$\begin{aligned} \frac{p(t,k)}{t^k} &= \prod_{i=1}^{k-1} \frac{(t-i)}{t} \\ &= \exp\left(\sum_{i=1}^{k-1} \ln(1 - \frac{i}{t})\right) \\ &\geq \exp\left(-\sum_{i=1}^{k-1} \frac{2i}{t}\right) \qquad \text{(since } k \leq t/2) \\ &= \exp\left(-\frac{(k-1)k}{t}\right) \end{aligned}$$

$$\geq e^{-\frac{k^2}{t}}.$$

□

For $t = \Delta$ and $k = |I| \leq \sqrt{\Delta}$ this lemma implies that $\Delta^k/p(\Delta, k) \leq e$ and hence, given $\mathcal{A}_{\epsilon_1}$, the X_i are λ–correlated with $\lambda = (1+\epsilon_1)^k e \leq e^{1+\epsilon_1\sqrt{\Delta}}$, via the standard approximation $1 + x \leq e^x$. Though λ goes to infinity superpolynomially fast as Δ goes to infinity, we can still upper bound $Pr(X_G > (1+\epsilon)\Delta/e)$ by a term that goes to zero superpolynomially fast, as follows. (Here, $\mu = \sqrt{\Delta}/e$)

$$\begin{aligned}
Pr(X_G > (1+\epsilon)\mu) &= Pr(X_G > (1+\epsilon)\mu \mid \mathcal{A}_{\epsilon_1})\, Pr(\mathcal{A}_{\epsilon_1}) \; + \\
&\quad\; Pr(X_G > (1+\epsilon)\mu \mid \mathcal{A}^c_{\epsilon_1})\, Pr(\mathcal{A}^c_{\epsilon_1}) \\
&\leq Pr(X_G > (1+\epsilon)\mu \mid \mathcal{A}_{\epsilon_1}) \; + \; Pr(\mathcal{A}^c_{\epsilon_1}) \\
&\leq \lambda\, F(\mu, \epsilon) \; + \; Pr(\exists G\; X_G > (1+\epsilon_1)\mu) \\
&\leq e^{1+\epsilon_1\sqrt{\Delta}}\, F(\mu, \epsilon) \; + \; \sqrt{\Delta}\, F(\mu, \epsilon_1) \\
&\leq e^{1+\epsilon_1\sqrt{\Delta}}\, e^{-\epsilon^2\sqrt{\Delta}/3} \; + \; \sqrt{\Delta}\, F(\mu, \epsilon_1) \\
&\leq e^{1+(\epsilon_1-\epsilon^2/3)\sqrt{\Delta}} \; + \; \sqrt{\Delta}\, F(\mu, \epsilon_1).
\end{aligned}$$

Since ϵ_1 can be chosen arbitrarily small, we can make the above term go to zero exponentially fast by choosing ϵ_1 such that $\epsilon_1 - \epsilon^2/3 < 0$.

Remark. We can now see why the parameter THRESHOLD must be $\Omega(\log^{2+\delta} n)$: the above failure probability goes to zero superpolynomially fast as long as $\Delta = \Omega(\log^{2+\delta} n)$, for any fixed $\delta > 0$.

Hence, we have proven that $X_G < (1+\epsilon)\sqrt{\Delta}/e$ *w.h.p.*, for any fixed $\epsilon > 0$. In turn, this implies that the new degree of v_B after one iteration of the algorithm is at most $(1+\epsilon)\Delta/e$ *w.h.p.* To see this,

$$\begin{aligned} Pr(X > (1+\epsilon)\Delta/e) &\leq Pr(\exists G\ X_G > (1+\epsilon)\sqrt{\Delta}/e) \\ &\leq \sqrt{\Delta}\ Pr(X_G > (1+\epsilon)\sqrt{\Delta}/e) \\ &\leq \sqrt{\Delta}\ \left(e^{1+(\epsilon_1-\epsilon^2/3)\sqrt{\Delta}} + \sqrt{\Delta}F(\mu,\epsilon_1)\right). \end{aligned}$$

This inequality holds for any ϵ and ϵ_1. Hence the new degree of *all* the vertices is at most $(1+\epsilon)\Delta/e$ *w.h.p.* and we get

Theorem 16 *The new degree of the graph after one iteration of Part (I) of the algorithm is at most* $(1+\epsilon)\Delta/e$ w.h.p., *for any fixed* $\epsilon > 0$.

Bibliography

[ABI86] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7:567–583, 1986.

[AGHP92] N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple constructions of almost k–wise independent random variables. *Random Structures and Algorithms*, 3(3):289–303, 1992.

[AGLP89] 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.

[AP90] B. Awerbuch and D. Peleg. Sparse partitions. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 503–513, 1990.

[AP92] B. Awerbuch and D. Peleg. Routing with polynomial communication-space trade-off. *SIAM J. on Discrete Mathematics*, 5(2):151–162, 1992.

[ASE92] N. Alon, J. Spencer, and P. Erdős. *The Probabilistic Method.* Wiley–Interscience Series, John Wiley & Sons, Inc., New York, 1992.

[Awe85] B. Awerbuch. Complexity of network synchronization. *J. Assoc. Comput. Mach.*, 32:804–823, 1985.

[BC] B. Berger and L. Cowen. Fast deterministic constructions of low-diameter network decompositions. MIT-LCS Technical Memo #460, April 1991.

[BO83] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 27–30, 1983.

[Bol79] B. Bollobás. *Graph Theory.* Springer Verlag, New York, 1979.

[BR91] B. Berger and J. Rompel. Simulating ($\log^c n$)-wise independence in NC. *J. Assoc. Comput. Mach.*, 38(4):1026–1046, 1991.

[Bro41] R.L. Brooks. On colouring the nodes of a network. *Proc. Cambridge Phil. Soc.*, 37:194–197, 1941.

[Che52] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–509, 1952.

[CS92] M. Choy and A.K. Singh. Efficient fault tolerant algorithms for resource allocation in distributed systems. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 593–602, 1992.

[CV86] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70:32–53, 1986.

[ES73] P. Erdős and J.L. Selfridge. On a combinatorial game. *Journal of Combinatorial Theory, Series A*, 14:298–301, 1973.

[FLP85] M.J. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32:374–382, 1985.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[GPS89] A. V. Goldberg, S. A. Plotkin, and G. E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM J. Disc. Math.*, 1:434–446, 1989.

[GS] M. Goldberg and T. Spencer. A new parallel algorithm for the maximal independent set problem. *SIAM Journal of Computing*, pages 419–428.

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, pages 13–30, 1963.

[HS90] P. Hajnal and E. Szemerédi. Brooks coloring in parallel. *SIAM J. Disc. Math*, 3(1):74–80, 1990.

[II86] A. Israeli and A. Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):77–80, 1986.

[JSWB92] R. Jain, K. Somalwar, J. Werth, and J. C. Browne. Scheduling parallel i/o operations in multiple bus systems. *Journal of Parallel and Distributed Computing*, 16(4):352–362, 1992.

[Kar86] H. Karloff. A las vegas RNC algorithm for maximum matching. *Combinatorica*, 6:187–191, 1986.

[Kar89] H. Karloff. An NC algorithm for Brooks' theorem. *Theoretical Computer Science*, 68:89–103, 1989.

[Kar91] R. M. Karp. Probabilistic recurrence relations. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 190–197, 1991.

[KB87] H. Karloff and J. Boyar. Coloring planar graphs in parallel. *Journal of Algorithms*, 8:470–479, 1987.

[KN88] M. Karchmer and J. Naor. A fast parallel algorithm to color a graph with Δ colors. *Journal of Algorithms*, 9:83–91, 1988.

[KS87] H. J. Karloff and D. B. Shmoys. Efficient parallel algorithms for edge coloring problems. *Journal of Algorithms*, 8:39–52, 1987.

[KW85] R. M. Karp and A. Wigderson. A fast parallel algorithm for the maximal independent set problem. *J. Assoc. Comput. Mach.*, 32:762–773, 1985.

[Lei92] T. Leighton. Methods for message routing in parallel machines. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 77–96, 1992.

[Lin92] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[LPV81] 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.

[LS91] N. Linial and M. Saks. Decomposing graphs into regions of small diameter. In *Proceedings of the ACM/SIAM Symposium on Discrete Algorithms*, pages 320–330, 1991.

[Lub86] M. Luby. A fast parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986.

[Lub88] 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.

[Lyn81] N. A. Lynch. Upper bounds for static resource allocation in a distributed system. *Journal of Computer and System Sciences*, 23:254–278, 1981.

[MNN89] R. Motwani, J. Naor, and M. Naor. The probabilistic method yields deterministic parallel algorithms. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 8–13, 1989.

[Nao91a] J. Naor. A fast parallel coloring of planar graphs with five colors. *SIAM Journal Disc. Math.*, 4(3):409–412, 1991.

[Nao91b] M. Naor. A lower bound on probabilistic algorithms for distributed ring coloring. *SIAM Journal Disc. Math.*, 4(3):409–412, 1991.

[NN90] J. Naor and M. Naor. Small–bias probability spaces: efficient constructions and applications. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 213–223, 1990.

[NS92] M. Naor and L. Stockmeyer. What can be computed locally? Unpublished manuscript, to appear in STOC 93, 1992.

[Pan] A. Panconesi. Unpublished manuscript.

[PS92a] A. Panconesi and A. Srinivasan. Fast randomized algorithms for distributed edge coloring. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 251–262, 1992.

[PS92b] 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.

[PS92c] A. Panconesi and A. Srinivasan. The local nature of Δ–colorings and its algorithmic applications. Submitted for publication to COMBINATORICA, 1992.

[Rag90] P. Raghavan. Lecture notes on randomized algorithms. Technical Report RC 15340 (#68237), IBM T.J.Watson Research Center, January 1990.

[Ram90] R. Raman. The power of Collision: Randomized parallel algorithms for chaining and integer sorting. In *Proceedings, 10th Annual FST & TCS Conference*, Lecture Notes in Computer Science # 472, pages 161–175. Springer-Verlag, Berlin, December 1990.

[Spe87] J. Spencer. *Ten Lectures on the Probabilistic Method.* SIAM, Philadelphia, 1987.

[Sri] A. Srinivasan. Personal Communication.

[Sri93] A. Srinivasan. *Techniques for probabilistic analysis and randomness-efficient computation.* Ph.D. dissertation, Cornell University, August 1993.

[SP88] E. Styer and G.L. Peterson. Improved algorithms for distributed resource allocation. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 105–116, 1988.