BEN-GURION UNIVERSITY OF THE NEGEV

FACULTY OF NATURAL SCIENCES

DEPARTMENT OF COMPUTER SCIENCE

Distributed Graph Coloring

Thesis submitted as partial fulfillment of the requirements for

M.Sc. degree

Ariel Sapir

Under the supervision of: Prof. Michael Elkin

December 4, 2018

BEN-GURION UNIVERSITY OF THE NEGEV

FACULTY OF NATURAL SCIENCES

DEPARTMENT OF COMPUTER SCIENCE

Distributed Graph Coloring

Thesis submitted as partial fulfillment of the requirements for

M.Sc. degree

Ariel Sapir

Under the supervision of: Prof. Michael Elkin

Signature of student: _____ *Date:* 12/03/2019

Signature of supervisor: _____ *Date:* 14/03/2019

Signature of chairperson of the committee for graduate studies: _____ *Date:* 20/3/19

December 4, 2018

Abstract

Consider an n-vertex graph of maximum degree Δ, and assume that each vertex $v \in V$ hosts an autonomous processor. A processor of a vertex u is allowed to communicate with a processor of a vertex v iff $(u, v) \in E$. The communication is synchronous, i.e., it proceeds in discrete rounds. The number of rounds is referred to as the *running time*.

In the *distributed vertex coloring problem* the objective is to color G with $\Delta + 1$ (or slightly more) colors, with a running time as small as possible. Another interesting approach to the *distributed vertex coloring problem* seeks to obtain a coloring of G using $O(a)$ colors, where a is the arboricity of the graph. We will specifically look for $((2 + \varepsilon) \cdot a)$-coloring for an arbitrarily small $\varepsilon > 0$, yet we will also discuss similar related variants of the coloring problems.

A once open question was whether or not a $(\Delta + 1)$-coloring can be computed in a sublinear (in Δ) time. Barenboim [2] and Fraigniaud, Heinrich and Kosowski [11] answered this in the affirmative. This leads to opening a discussion about faster $O(a)$-colorings. Our intention is to use these algorithms (specifically, Barenboim's Algorithm) in order to answer the questions which arise from this discussion, and to improve the best currently known running time for $((2 + \varepsilon) \cdot a)$-coloring. We will also discuss improving the running time for related coloring problems, which arise from our improvement.

Acknowledgements

I would like to thank my advisor, Prof. Michael Elkin, for the guidance and assistance I have received from him, both in the topics researched in this thesis and in previously taken courses Prof. Elkin taught. By introducing me to this subject of Distributed Computing, and specifically to Distributed Coloring, I have been able both to expand my knowledge and to achieve new results, that will be described in this thesis.

I am as well most thankful to my family for their assistance, both in this thesis and in the rest of my educational courses and degree. A special thanks goes to my father, Amir, my mother, Luba, my Brother, Eytan, both my grandmothers, Tamara and Rivka, and to my fiancé, Selina. With the combined assistance of everyone, I have been able to achieve all so far.

Contents

1 Introduction **2**

1.1 Organization 3

1.2 Preliminaries 4

1.2.1 Running Time in the Distributed Model 4

1.2.2 Basic Definitions for Coloring 4

1.2.3 Defective Coloring 4

1.2.4 Orientations 4

1.2.5 Forest Decomposition 5

1.2.6 Arboricity 5

1.2.7 Arbdefective Coloring 6

1.2.8 H-Partition 6

1.2.9 List Coloring 6

1.3 Our Results 7

1.4 Techniques 8

2 Toolkit **9**

2.1 Barenboim's Sublinear $(\Delta + 1)$-Coloring Algorithm 9

2.2 Constructing an H-Partition 15

2.3 Barenboim's List Coloring Algorithm 16

3 $O(a)$-Coloring Using Barenboim's Algorithm **18**

4 List-Coloring and Coloring of Partially Colored Graphs **25**

5 Conclusions and Open Problems **34**

Chapter 1

Introduction

Distributed Computing is a large and growing field of study, which is concerned with various settings in which there are several processors working in parallel, typically on a joint problem. The area is thoroughly described in [1, 6, 14, 16, 17].

Our research is within the framework of *message-passing model*. In this model, a communication network is represented by an n-vertex graph $G = (V, E)$, whose vertices host autonomous processors. The processors communicate over the edges of G in discrete rounds where, in each round, every processor (vertex) sends messages to its neighbours.

There are two main communication models, *LOCAL* and *CONGEST*, both of which are synchronous. In the *CONGEST* model, each message has an upper bound on its size. That is, a message cannot be too long or contain too much information. In the *LOCAL* model, the message's size is unbounded. We will focus on the *LOCAL* model.

The message-passing model of distributed computing has been a subject of intensive research since the beginning of the eighties, with the most important results that were known up until the year 2000 being described in [16]. However, the area has been extremely active since then, and some of the developments in this area are described in [6]. Most relevant for this thesis is the area of *distributed graph coloring*.

Distributed Graph Coloring In a graph G of maximum degree Δ, in the distributed coloring problem one typically seeks to compute a $(\Delta + 1)$-coloring of G. One can also aim at an $O(a)$-coloring of G, where $a = a(G)$ is the *arboricity* of G. In addition to *proper coloring*, we study the notion of *defective coloring* and its applications to the computation of $(\Delta + 1)$-coloring in $O(\Delta) + \log^* n$ time. We further need the notion of *arbdefective coloring*, which is an extension of the no-

tion of defective coloring. Another line of research discusses a different variant of coloring – *edge coloring* – in which one colors the edges rather than the vertices. In this context, typically one seeks a $(2\Delta - 1)$-edge-coloring.

Forest Decomposition Coloring forests can be performed extremely efficiently in the distributed setting, both in terms of running time and number of colors. However, coloring general graphs is a significantly more challenging task. The best currently known deterministic algorithms for general graphs for $(\Delta + 1)$-coloring requires sub-linear in Δ time. Nevertheless, for a wide range of graph families, it is possible to achieve much better results. If a graph can be decomposed into a reasonably small number of oriented forests, then both the running time and the size of the employed coloring palette can be reduced. Indeed, each of the forests in the decomposition can be colored quickly with 3 colors [8]. (Even though each forest can be colored with 2 colors, this requires $\Omega(n)$ time [13], whereas 3-coloring can be computed in $\log^* n + O(1)$ time [8].) However, using the decomposition to achieve a unified proper coloring of the entire input graph is more complicated. For this purpose we will use a certain type of graph partition – *H-Partition* [3].

In this thesis we aim to improve the running time for different types of $O(a)$-coloring of a graph $G = (V, E)$. First, we will improve the current best known running time for $((2 + \varepsilon) \cdot a)$-coloring, for an arbitrarily small $\varepsilon > 0$, by applying a new algorithm [2] for $(\Delta + 1)$-coloring in sublinear (in Δ) time. Then, we will extend this result to other related problems. Specifically, we will implement our methods for *Partially-Colored Graphs* and *List-Coloring*, in which each vertex has a list of $O(a)$ colors (the exact minimal size for the list will be detailed further on).

1.1 Organization

In Chapter 1, we first present an introduction to the topic and to all the required notions. Then, in Chapter 2, we review and discuss previous results that will be of use in our results.

In Chapter 3, we combine Barenboim's Algorithm with the H-Partition in order to receive a $((2 + \varepsilon) \cdot a)$-coloring algorithm with a better running time than the previously known one.

In Chapter 4, we first adapt our previous result to the problem of partially-colored graphs. We then use this to solve the problem of list-coloring.

In Chapter 5, we summarize our results, and discuss open problems that arise

from our work.

1.2 Preliminaries

1.2.1 Running Time in the Distributed Model

The *running time* of an algorithm in the distributed model is the worst-case number of rounds of distributed communication that are required by the algorithm. Local computations are not taken into account.

1.2.2 Basic Definitions for Coloring

Let $G = (V, E)$ be a graph, and $C = \{1, 2, \ldots, |C|\}$ be a set of colors. Then:

A proper *vertex coloring* is a mapping $\varphi : V \to C$ such that for each $v_i, v_j \in V$, if $(v_i, v_j) \in E$ then $\varphi(v_i) \neq \varphi(v_j)$.

A proper *edge coloring* is a mapping $\varphi : E \to C$ such that for each $e_i, e_j \in E$, if $e_i \cap e_j \neq \emptyset$ then $\varphi(e_i) \neq \varphi(e_j)$.

An α-*vertex coloring* is a proper vertex coloring of a graph using α colors, where $\alpha \in \mathbb{N}$. Especially, we are interested in $(\Delta + 1)$-coloring, where Δ is the maximum degree among the graph vertices.

A β-*edge coloring* is a proper edge coloring of a graph using β colors, where $\beta \in \mathbb{N}$. We are particulary interested in $(2\Delta - 1)$-edge-coloring.

1.2.3 Defective Coloring

A *Defective Coloring* is a coloring that allows two adjacent vertices to hold the same color. Given a defective coloring φ, its *defect* is the smallest number $\rho \in \mathbb{N}$ such that each $v \in V$ has at most ρ neighbours with the same color $\varphi(v)$ [4, 9, 10, 12]. A ρ-*defective* α-*coloring* is a coloring φ using α colors that has a defect ρ.

1.2.4 Orientations

For a vertex $v \in V$, we will denote by $\Gamma(v)$ the set of all neighbours of v. That is, $\Gamma(v) = \{u \mid (u, v) \in E\}$.

The *out-degree* of a vertex v in a directed graph is the number of edges incident to v that are oriented out of v. That is, out-degree$(v) = |\{(v, u) \in E\}|$.

An *orientation* σ of a graph is an assignment of a direction for each edge. That is, each $(u, v) \in E$ will be oriented either towards u or towards v. A *partial orientation* is an orientation of a subset of E. That is, there exists a $E' \subseteq E$ such that each $(u, v) \in E'$ is oriented either towards u or v and the rest of the edges $(u, v) \in E \backslash E'$ are non-oriented. An *oriented cycle* is a cycle $\langle v_0, v_1, ..., v_k, v_0 \rangle$ in which (v_k, v_0) is oriented by σ towards v_0 and each edge (v_i, v_{i+1}), for $0 \leq i \leq k-1$, is oriented by σ towards v_{i+1}. An *acyclic partial orientation* is a partial orientation which does not contain an oriented cycle.

Given a partial orientation σ, the *length* of a vertex v in this orientation, denoted by $len(v) = len_\sigma(v)$, is the length k of the longest directed path $\langle v = v_0, v_1, ..., v_k \rangle$ that emanates from v, where all edges (v_i, v_{i+1}), for $0 \leq i \leq k-1$, are oriented by σ towards v_{i+1}. The *length* of the entire orientation, denoted by $len(\sigma)$, is the maximal length of a vertex in the orientation. That is, $len(\sigma) = \max_{v \in V} len(v)$.

The *deficit* of a vertex v with respect to a partial orientation σ is the number of edges $e \in E$ that are unoriented by σ and are incident to v. The *deficit* of σ is the maximal deficit of a vertex $v \in V$ in respect to σ.

The *out-degree* of σ is the maximum out-degree of a vertex in G with respect to σ.

1.2.5 Forest Decomposition

An *oriented forest* is an acyclic graph with orientation on the edges with out-degree at most 1. Each edge in an oriented forest is directed towards the parent. (That is, v is the parent of u iff the edge (u, v) is oriented towards v.) A *k-forest decomposition* is a partition of the edge set E of a graph $G = (V, E)$ into at most k sets, such that each one is an oriented forest.

Let $\{F_1, F_2, \ldots, F_k\}$ be a k-forest decomposition of G and let $v \in V$ be a vertex. For each $1 \leq i \leq k$, denote the parent of v in F_i by u_i. The set $\overline{\Gamma}(v) = \{u_1, u_2, \ldots, u_k\}$ is called the set of *parents* of v in the forest decomposition $\{F_1, F_2, \ldots, F_k\}$.

1.2.6 Arboricity

The *arboricity* of a graph G is the minimum number of forests into which its edges can be partitioned. It is denoted by $a(G)$ (or, shortly, a). Equivalently, by Nash-

Williams theorem [15], it equals:

$$a = \max \left\lceil \frac{|E_H|}{|V_H| - 1} \right\rceil,$$

where the maximum is taken over all subgraphs H of G with at least two vertices ($|V_H| \geq 2$).

In addition to $(\Delta + 1)$-coloring, one may wish to find an $O(a)$-coloring of G.

1.2.7 Arbdefective Coloring

An *arbdefective coloring* [5] is a defective coloring with constraints on the arboricity of each color class. Let $G = (V, E)$ be a graph with arboricity a, and a pair of integer nonnegative parameters p, q. A p-coloring φ of G is called *q-arbdefective* if for every color $1 \leq i \leq p$, the subset $V_i = \{v \in V | \varphi(v) = i\}$ induces a subgraph G_i of arboricity at most q.

Finally, arbdefective colorings come along with forest decomposition for each G_i. These forest decompositions define the set $\overline{\Gamma}(v)$ of *parents* that has every vertex $v \in V_i$, for every $1 \leq i \leq p$.

1.2.8 H-Partition

For an integer $A > 2a$, an ***H-Partition*** is a vertex-partition of a graph $G = (V, E)$ into subsets $H_1, H_2, \ldots, H_l \subseteq V$, such that each vertex in $H_i, 1 \leq i \leq l$, has at most A neighbours in $\bigcup_{j=i}^{l} H_j$. The parameter A is called the *degree* of the H-Partition $H_1, H_2, \ldots, H_l$. The parameter l is called the *size* of the H-Partition.

1.2.9 List Coloring

Recall that in a proper α-coloring, each vertex $v \in V$ is assigned a color $\varphi(v) \in \{0, 1, \ldots, \alpha - 1\}$ such that for each neighbour u of v, we have that $\varphi(u) \neq \varphi(v)$. In an instance of a proper *List Coloring*, each vertex v is assigned a list $R(v)$ of *permitted colors*. In addition to having $\varphi(u) \neq \varphi(v)$, we also require $\varphi(v) \in R(v)$. That is, v's color will be chosen from the list of permitted colors. In the *α-list coloring problem*, every list is of size at least α. Note that an α-coloring is also a list coloring in which for every $v \in V$, the list of permitted colors $R(v)$ is set by $R(v) = \{0, 1, \ldots, \alpha - 1\}$.

1.3 Our Results

The first problem we deal with in this thesis is the problem of $((2+\varepsilon)\cdot a)$-coloring of a graph $G=(V,E)$, where a is the arboricity of the graph and $\varepsilon > 0$ is an arbitrarily small number. Previous to our result, the best known result was $((2+\varepsilon)\cdot a)$-coloring in $O(a\cdot \log n)$ time [3]. We will improve the running time, by combining the construction of an H-Partition with Barenboim's algorithm [2]. As a result, we obtain a $((2+\varepsilon)\cdot a)$-coloring in $O\left(\sqrt{a}\cdot \log^2 a\cdot \frac{1}{\varepsilon}\cdot \log n\right)$ time.

In addition, we adapt the above result to create an algorithm which extends a partial coloring of a graph. Then, we extend the result concerning partial-coloring and present an algorithm to compute a list-coloring. More specifically, let $d \in O(A)$, $d \geq (1+\lambda)\cdot A = (1+\lambda)\cdot(2+\varepsilon)\cdot a$, where $A=(2+\varepsilon)\cdot a$ is the out-degree of the H-Partition and λ, $\varepsilon > 0$ are arbitrarily small constants. In our scenario, we assume each vertex $v \in V$ has a set $R(v)$ of at least $\frac{\lambda}{1+\lambda}\cdot d + A$ permitted colors taken from the range $\{0,1,\ldots,d-1\}$. (Note that, under these conditions, $\frac{\lambda}{1+\lambda}\cdot d + A \leq d$.) We then present an algorithm which computes a proper list-coloring of G in $O\left(\sqrt{a}\cdot \log^2 a\cdot \frac{1}{\varepsilon}\cdot \log n\right)$ time.

Problem	**Previous running time**	**Reference**	**Our running time**	**In**
$((2+\varepsilon)\cdot a)$-coloring	$O(a\cdot \log n)$	Barenboim, Elkin [3]	$O\left(\sqrt{a}\cdot \log^2 a\cdot \frac{1}{\varepsilon}\cdot \log n\right)$	Ch. 3
Partially-Colored Graphs with $O(a)$ colors	none	–	$O\left(\sqrt{a}\cdot \log^2 a\cdot \frac{1}{\varepsilon}\cdot \log n\right)$	Ch. 4
List-Coloring with $O(a)$ colors	none	–	$O\left(\sqrt{a}\cdot \log^2 a\cdot \frac{1}{\varepsilon}\cdot \log n\right)$	Ch. 4

Table 1.1: *Results comparison for several variants of $O(a)$-coloring problems.*

While constructing several new procedures, our main theorems in this thesis will be:

Theorem 1.1. *A graph $G=(V,E)$, partitioned by an H-Partition with out-degree A and size $l=\left\lceil \frac{2}{\varepsilon}\cdot \log n\right\rceil$, can be properly colored using $(1+\lambda)\cdot A$ colors in $O\left(\sqrt{A}\cdot \log^2 A\cdot \frac{1}{\lambda}\cdot \log n\right)$ time, for arbitrarily small constants $\varepsilon=\lambda>0$.*

Theorem 1.2. *A graph $G=(V,E)$ with arboricity $a=a(G)$ can be properly colored using $(2+\varepsilon')\cdot a$ colors in $O\left(\sqrt{a}\cdot \log^2 a\cdot \frac{1}{\varepsilon'}\cdot \log n\right)$ time, for an arbitrarily small constant $\varepsilon'>0$.*

Theorem 1.3. *A graph $G=(V,E)$ that:*

- *has a partial coloring φ_0,*
- *and has been partitioned by an H-Partition with out-degree A*

can be properly colored using $(1+\lambda)\cdot A$ colors in $O\left(\sqrt{A}\cdot\log^2 A\cdot\frac{1}{\lambda}\cdot\log n\right)$ time, for arbitrarily small constants $\varepsilon=\lambda>0$.

Theorem 1.4. *A partial coloring of a graph $G=(V,E)$ with arboricity $a=a(G)$ can be extended to a proper coloring of the entire graph G which makes use of $(2+\varepsilon')\cdot a$ colors in $O\left(\sqrt{a}\cdot\log^2 a\cdot\frac{1}{\varepsilon'}\cdot\log n\right)$ time, for an arbitrarily small constant $\varepsilon'>0$.*

Theorem 1.5. *A graph $G=(V,E)$ with arboricity $a=a(G)$, where each vertex $v\in V$ has a list of permitted colors $R(v)$ of size at least $\frac{\lambda}{1+\lambda}\cdot d+A$ taken from the range $\{0,1,\ldots,d-1\}$, where:*

- *$A=(2+\varepsilon)\cdot a$,*
- *and $d\in O(a)$, $d\geq(1+\lambda)\cdot A$,*

can be properly list-colored (by using d colors) in $O\left(\sqrt{a}\cdot\log^2 a\cdot\frac{1}{\varepsilon}\cdot\log n\right)$ time, for an arbitrarily small constant $\varepsilon>0$.

1.4 Techniques

The main technique we use can be seen in our first result. We use an algorithm that computes a coloring dependent on the maximum degree Δ for the sake of constructing an algorithm which computes a coloring with a number of colors that depends on the graph's arboricity. In order to do so, we employ the idea of first applying an H-Partition on a graph. Then, its out-degree will be upper-bound by $A=(2+\varepsilon)\cdot a$, for an arbitrarily small $\varepsilon>0$. Then, we use Barenboim's fast $(\Delta+1)$-coloring algorithm, and receive an algorithm for $((2+\varepsilon)\cdot a)$-coloring. The same technique is applied for the first part of our second result, i.e., for the *Partial-Coloring*.

For the second part of our second result, i.e., for the *List-Coloring*, we use a technique that consists of two main steps: First, we use the reduction from List-Coloring to Partial-Coloring that creates, for each vertex $v\in V$, an additional set of fictional vertices, each representing a color forbidden to be chosen by v. This reduction is described in Section 2.3. Further on, we use the first part of our second result (i.e., the Partial-Coloring algorithm) as a blackbox, hence achieving an algorithm to compute a proper List-Coloring.

Chapter 2

Toolkit

In this chapter, we will present and discuss multiple results that will later on be used in our algorithms. First, we will present and discuss the result of Barenboim [2], i.e., $(\Delta + 1)$-coloring in sublinear time. Then, we will present an algorithm for constructing an H-Partition. In our first result, we will combine the usage of these two results.

Further on, we will discuss different applications of Barenboim's Algorithm, for problems such as *Partially-Colored Graphs* and *List-Coloring*.

2.1 Barenboim's Sublinear $(\Delta + 1)$-Coloring Algorithm

Barenboim's algorithm [2] colors a graph $G = (V, E)$ in $\Delta + 1$ colors in $O\left(\Delta^{\frac{3}{4}} \cdot \log \Delta + \log^* n\right)$ time. It can also be used (depending on the parameters of the algorithm) to compute a $((1 + o(1)) \cdot \Delta)$-coloring in $O\left(\sqrt{\Delta} \cdot \log^3 \Delta + \log^* n\right)$ time.

We will now briefly describe Barenboim's algorithm and its traits. The main part of his algorithm is *Procedure Color*. It is given as input two parameters, p and q, that will be assigned specific values later on. *Procedure Color* uses two helper procedures:

- Procedure Partition - A procedure that computes a vertex disjoint partition into subgraphs $G_1, ..., G_p$, such that each subgraph has an arboricity of $O\left(\frac{\Delta}{p} \cdot \log_{\frac{q}{p}} \Delta\right)$. Moreover, each subgraph G_i comes along with a forest decomposition $\mathcal{F}_i$ for G_i, with each $\mathcal{F}_i$ consisting of $O\left(\frac{\Delta}{p} \cdot \log_{\frac{q}{p}} \Delta\right)$

forests. This is an $O\left(\frac{\Delta}{p} \cdot \log_{\frac{q}{p}} \Delta\right)$-arbdefective p-coloring, computed in $O\left(q \cdot \log_{\frac{q}{p}} \Delta + \log^* n\right)$ time.

- Procedure Recolor - A procedure that, given a temporary coloring φ_i of a subgraph G_i, a list $L(v)$ of colors forbidden for each vertex $v \in V(G_i)$ from previously-colored subgraphs G_j ($j < i$), a subset $\overline{\Gamma}(v) \subseteq \Gamma(v)$ of parents of v in G_i, a list $B(v) = \varphi_i\left(\overline{\Gamma}(v)\right)$ of the colors of these parents in φ_i, an upper bound ℓ on the size of $L(v)$, and an upper bound β on the size of $B(v)$, computes a coloring ψ of vertices of G_i, which is consistent with the colors of vertices in G_j for all $j < i$.

Procedure Color first invokes *Procedure Partition* to compute an $O\left(\frac{\Delta}{p} \cdot \log_{\frac{q}{p}} \Delta\right)$-arbdefective p-coloring. The color classes of this coloring form a partition $G_1, ..., G_p$. Then, it calculates a temporary coloring φ_i for each G_i, which does not take into account neighbours from other subgraphs G_j ($j \neq i$). This temporary coloring is computed by algorithm Arb-Linial [3]. It uses $c \cdot a(G_i)^2$ colors, where:

- $c > 4$ is a universal constant (see [3]),
- and, due to *Procedure Partition*, $a(G_i) \in O\left(\frac{\Delta}{p} \cdot \log_{\frac{q}{p}} \Delta\right)$ for all $1 \leq i \leq p$.

This can be seen as the preparation phase. Next, it iterates over all G_i's, each time employing *Procedure Recolor* on each G_i, with the respective parameters required by *Procedure Recolor*. The algorithm ends when all of the subgraphs $G_1, G_2, ..., G_p$ are colored.

Procedure Recolor computes a proper coloring ψ. During each iteration on G_i, it computes the color $\psi(v)$ for each vertex $v \in V(G_i)$. The colors $\psi(u)$ of all vertices $u \in V(G_j)$ of subgraphs G_j, for $j < i$, were computed already on an earlier iteration. In order to compute ψ for all vertices of G_i, before *Procedure Recolor* begins, each vertex $v \in V(G_i)$ calculates the required sets for *Procedure Recolor*. These are:

- $L(v)$ – the ψ-colors of previously colored neighbours u of v, $u \in V(G_j)$, $j < i$,
- $\overline{\Gamma}(v)$ – the subset of parents of v in G_i,
- and $B(v)$ – the φ_i-colors of these parents $\overline{\Gamma}(v)$.

The pseudo-code of *Procedure Color* is provided below:

Algorithm 1 `Procedure Color`

1: $\langle G_1, G_2, ..., G_p, a \rangle$:= *Partition*(G, p, q)
/* a is an upper bound on the arbdefect of the coloring that Procedure Partition returns */
2: **for** $i \leftarrow 1$ to p **in parallel do**
3: φ_i:= *Arb-Linial*(G_i)
4: **if** $v \in V(G_i)$ **then**
5: $\varphi(v) := \varphi_i(v)$
6: **end if**
7: **end for**
8: $\psi(v) := \perp$
9: **for** $i = 1, 2, ..., p$ **do**
10: **if** $v \in V(G_i)$ **then**
11: $L(v) := \{\psi(u) \mid u \in \Gamma(v) \wedge \psi(u) \neq \perp\}$
12: $\overline{\Gamma}(v) := \{u \mid u \in V(G_i) \wedge u$ is a parent of v in $G_i\}$
13: $B(v) := \left\{\varphi(u) \mid u \in \overline{\Gamma}(v)\right\}$
14: $\psi(v) :=$ *Recolor*$\left(v, \varphi, L(v), \overline{\Gamma}(v), B(v), l := \Delta, \beta := a\right)$
15: **end if**
16: **end for**
17: **return** $\psi(v)$

As will be argued below, the above algorithm computes a proper coloring ψ of a graph $G = (V, E)$ that employs at most $\Delta + O\left(\sqrt{\Delta} \cdot \log_{\frac{q}{p}} \Delta \cdot \frac{\Delta}{p} + \left(\log_{\frac{q}{p}} \Delta \cdot \frac{\Delta}{p}\right)^2\right)$ colors in $O\left(q \cdot \log_{\frac{q}{p}} \Delta + \log^* n\right)$ time.
By setting different values to the parameters p and q, one may receive:

- $\Delta + 1$ colors in $O(\frac{1}{\varepsilon} \cdot \Delta^{\frac{3}{4}+\varepsilon} + \log^* n)$ time, for an arbitrarily small constant $\varepsilon > 0$, by setting $p = \left\lfloor \Delta^{\frac{3}{4}} \right\rfloor$ and $q = \left\lfloor \Delta^{\frac{3}{4}+\varepsilon} \right\rfloor$.
- $\Delta + 1$ colors in $O(\Delta^{\frac{3}{4}} \cdot \log \Delta + \log^* n)$ time, by setting $p = \left\lfloor \Delta^{\frac{3}{4}} \right\rfloor$ and $q = 2p$ (This is a slight improvement of the above.).
- $\Delta + o(\Delta)$ colors in $O(\sqrt{\Delta} \cdot \log^3 \Delta + \log^* n)$ time, by setting $p = \left\lfloor \sqrt{\Delta} \cdot \log^2 \Delta \right\rfloor$ and $q = 2p$.

We will next discuss the auxiliary procedures of the algorithm. We will be interested mostly in *Procedure Recolor*, as we will make use of its traits and properties. Let $v \in V(G_i)$. The procedure accepts as an input:

- $L(v)$ – a list of forbidden colors $\psi(u)$ of previously colored neighbours u of v (that is, $u \in V(G_j)$ with $j < i$),
- $\overline{\Gamma}(v)$ - the subset of parents of v that belong to G_i,

- $B(v)$ – the colors $\varphi(u)$ of these parents $u \in \overline{\Gamma}(v)$,
- ℓ – an upper bound on the size of $L(v)$ (that is, $|L(v)| \leq \ell$; we will mostly choose $\ell = \Delta$ for all $v \in V$),
- β – an upper bound on the size of $\overline{\Gamma}(v)$ (that is, $|\overline{\Gamma}(v)| \leq \beta$, for all $v \in V$),
- φ – an auxiliary proper coloring of G_i which employs $O\left((a\left(G_i\right))^2\right) = O\left(\left(\frac{\Delta}{p} \cdot \log_{\frac{q}{p}} \Delta\right)^2\right)$ colors and does not take into account vertices from other G_j $(j \neq i)$.

The goal of *Procedure Recolor* is to compute a coloring, denoted by ψ, that is proper and consistent with vertices from other subgraphs G_j $(j < i)$ that were previously colored by ψ. We will provide further details about *Procedure Recolor*. During the description of *Procedure Recolor*, we will use ℓ and β as the upper bounds of $|L(v)|$ and $|\overline{\Gamma}(v)|$, respectively. Hence, $\varphi = \varphi_i$ employs at most $c \cdot \beta^2$ colors, where $c > 4$ is a universal constant from algorithm Arb-Linial [3].

The coloring φ is a proper $c \cdot \beta^2$-coloring of G_i. Hence it follows that, for each $v \in V$, $\varphi(v) \notin B(v)$. However, $\varphi(v)$ may belong to $L(v)$. We now aim to find a coloring ψ such that $\psi(v) \notin L(v)$, for every $v \in V\left(G_i\right)$.

Let μ be a prime in the range $\left[\left\lceil\sqrt{\ell}\right\rceil + \left\lceil\sqrt{c \cdot \beta^2}\right\rceil, 2 \cdot \left(\left\lceil\sqrt{\ell}\right\rceil + \left\lceil\sqrt{c \cdot \beta^2}\right\rceil\right)\right]$. Note that such a prime exists by the Bertrand-Chebyshev postulate [7, Theorem 418]. We will use μ in order to construct the coloring ψ.

The computation of ψ requires only one communication round and proceeds as follows. Each vertex $v \in V$ constructs a set of μ polynomials $p_{0_v}(x), p_{1_v}(x), \ldots, p_{\mu-1_v}(x)$. These polynomials are defined over the field $\mathbb{Z}_\mu$ of integers modulo μ. For each $0 \leq i \leq \mu - 1$, the polynomial p_{i_v} is given by $p_{i_v}(x) = i + a \cdot x + b \cdot x^2$, where a and b are deduced from the color $\varphi(v)$ in the following way:

$$a = \left\lfloor \frac{\varphi(v)}{\left\lceil\sqrt{c \cdot \beta^2}\right\rceil} \right\rfloor \text{ and } b = \varphi(v) \bmod \left\lceil\sqrt{c \cdot \beta^2}\right\rceil$$

Let $u, v \in V$. Note that if $\varphi(u) \neq \varphi(v)$ then either $\left\lfloor \frac{\varphi(u)}{\left\lceil\sqrt{c \cdot \beta^2}\right\rceil} \right\rfloor \neq \left\lfloor \frac{\varphi(v)}{\left\lceil\sqrt{c \cdot \beta^2}\right\rceil} \right\rfloor$ or $\varphi(u) \bmod \left\lceil\sqrt{c \cdot \beta^2}\right\rceil \neq \varphi(v) \bmod \left\lceil\sqrt{c \cdot \beta^2}\right\rceil$. Hence, the sets of polynomials that will be constructed for such $u, v \in V$ with $\varphi(u) \neq \varphi(v)$ will be disjoint.

The color $\psi(v)$ will be determined by using the above polynomials. For this purpose, we aim, for each $v \in V$, to find a polynomial with the smallest number of conflicts with the set of previously forbidden colors $L(v)$. We will do this in the following way. Each vertex $v \in V$ finds an index $0 \leq i^* \leq \mu - 1$, such that the polynomial $p_{i^*_v}(x)$ of v satisfies the following condition: the set

$L_{i^*}(v) = L(v) \cap \{p_{i^*_v}(k) + k \cdot \mu \mid k \in \mathbb{Z}_\mu\}$ is of the smallest size among the sets $L_0(v), L_1(v), \ldots, L_{\mu-1}(v)$. Note that while $p_{i^*_v}(k)$ is computed over $\mathbb{Z}_\mu$, the sum $p_{i^*_v}(k) + k \cdot \mu$ is computed over the field of all integers. (A more intuitive way to think of the values $p_{i^*_v}(k) + k \cdot \mu$ is as a pair $(k, p_{i^*_v}(k)) \in \mathbb{Z}_\mu^2$. These pairs are in one-to-one correspondence with numbers in $\mathbb{Z}_{\mu^2}$.) Therefore, $0 \leq p_{i^*_v}(k) \leq \mu - 1$ and $0 \leq p_{i^*_v}(k) + k \cdot \mu \leq \mu^2 - 1$. Recall that $|L(v)| \leq \ell$ and there are μ sets $L_0(v), L_1(v), \ldots, L_{\mu-1}(v)$. Note that for every $0 \leq i < j \leq \mu - 1$, the substitution of the same value $k \in \mathbb{Z}_\mu$ both in $p_{i_v}(x)$ and in $p_{j_v}(x)$ will result in $p_{i_v}(k) \neq p_{j_v}(k)$ (since $i \neq j$), and for $k < k'$ the values $p_{i_v}(k) + k \cdot \mu$ and $p_{j_v}(k') + k' \cdot \mu$ are in different ranges and can never be equal. Hence, the sets $L_0(v),\ L_1(v), \ldots,\ L_{\mu-1}(v)$ are mutually disjoint. Thus, by the Pigeonhole Principle, $|L_{i^*}(v)| \leq \frac{|L(v)|}{\mu} \leq \frac{\ell}{\mu}$.

Since $\mu - \frac{\ell}{\mu} \geq \mu - \sqrt{\ell} \geq \left(\sqrt{\ell} + \sqrt{c} \cdot \beta\right) - \sqrt{\ell} > 2\beta$, the size of the set $\hat{L}_{i^*}(v) = \{p_{i^*_v}(k) + k \cdot \mu \mid k \in \mathbb{Z}_\mu\} \setminus L(v) = \{p_{i^*_v}(k) + k \cdot \mu \mid k \in \mathbb{Z}_\mu\} \setminus L_{i^*}(v)$ is greater than $\mu - \frac{\ell}{\mu} > 2\beta$. Since $\psi(v)$ will be chosen from the set $\hat{L}_{i^*}(v)$, we have argued that $\psi(v)$ will not be among the forbidden colors of $L(v)$. Next, we argue that ψ is also a proper coloring within G_i.

Let $u \in \overline{\Gamma}(v)$ be a parent of u in G_i. Denote by j^* the index $0 \leq j^* \leq \mu - 1$ such that $L_{j^*}(u)$ has minimum size among $L_0(u), L_1(u), \ldots, L_{\mu-1}(u)$. The polynomials $p_{i^*_v}(x)$ of v and $p_{j^*_u}(x)$ of u are both of degree 2. Moreover, these two polynomials are distinct. Hence, they intersect in at most 2 points. By the given input to *Procedure Recolor*, v has at most β parents in G_i (denoted by $\overline{\Gamma}$). Thus, its set $\hat{L}_{i^*}(v)$ has at most 2β intersections with the union of the sets $\hat{L}_{j^*}(u)$, taken over all $u \in \overline{\Gamma}(v)$, where $j^* = j_u$ is the index that the vertex u has selected. Recall that $|\hat{L}_{i^*}(v)| > 2\beta$. Hence, there exists at least one possible value $p_{i^*_v}(k) + k \cdot \mu \in \hat{L}_{i^*}(v)$ and $p_{i^*_v}(k) \neq p_{j^*_u}(k)$ for all $u \in \overline{\Gamma}(v)$. The color $\psi(v)$ will be set as $p_{i^*_v}(k) + k \cdot \mu$ for the smallest value of $k \in \mathbb{Z}_\mu$ that satisfies the above.

Let us consider the chosen color $\psi(v)$. Note that for every parent $u \in \overline{\Gamma}(v)$: for any $k' \in \mathbb{Z}_\mu$ such that $k' \neq k$, the values $p_{i_v}(k) + k \cdot \mu$ and $p_{j_u}(k') + k' \cdot \mu$ are in different ranges and thus differ. Also, if $k' = k$, recall that k was chosen such that $p_{i_v}(k) \neq p_{j_u}(k)$. Thus, it holds that $\psi(u) \neq \psi(v)$. That is, the coloring ψ is proper in G_i. Furthermore, the color $\psi(v) \notin L(v)$. Hence, it is consistent with previously colored subgraphs. In addition, since k is selected as the smallest appropriate integer, it holds that $k \leq |L_{i^*}(v)| + 2 \cdot |\overline{\Gamma}(v)| + 1 \leq \frac{\ell}{\mu} + 2 \cdot \beta + 1$. Thus,

$$\begin{aligned}\psi(v) &= p_{i^*_v}(k) + k \cdot \mu \leq \mu - 1 + \left(\tfrac{\ell}{\mu} + 2 \cdot \beta + 1\right) \cdot \mu \\ &= \mu - 1 + \ell + 2 \cdot \beta\mu + \mu < 2\mu + \ell + 2 \cdot \beta\mu \\ &= \ell + (2\beta + 2) \cdot \mu\end{aligned}$$

Recall that $\mu \in O\left(\sqrt{\ell} + \beta\right)$. Hence, the number of colors of ψ is $\ell + O\left(\sqrt{\ell} \cdot \beta + \beta^2\right)$.

This completes the description of *Procedure Recolor*. We provide its pseudo-code below:

Algorithm 2 `Procedure Recolor` $(\varphi, v, L(v), \overline{\Gamma}(v), B(v), l, \beta)$

1: $a := \left\lfloor \frac{\varphi(v)}{\left\lceil \sqrt{c \cdot \beta^2} \right\rceil} \right\rfloor$
2: $b := \varphi(v) \mod \left\lceil \sqrt{c \cdot \beta^2} \right\rceil$
3: $\mu :=$ a prime such that $\left\lceil \sqrt{l} \right\rceil + \lceil \sqrt{c} \rceil \cdot \beta \leq \mu \leq 2 \cdot \left(\left\lceil \sqrt{l} \right\rceil + \lceil \sqrt{c} \rceil \cdot \beta \right)$
4: **for** $i = 0, 1, ..., \mu - 1$ **do**
5: **define** $p_{i_v}(x) = i + a \cdot x + b \cdot x^2$
6: **end for**
7: **find** $i \in \{0, 1, ..., \mu - 1\}$
such that $L_i(v) = L(v) \cap \{p_{i_v}(k) + k \cdot \mu \mid k = 0, 1, ..., \mu - 1\}$
has minimum size among $L_0(v), L_1(v), ..., L_{\mu-1}(v)$.
8: **send** $p_{i_v}(x)$ to all neighbours in $\Gamma(v)$.
9: **receive** $p_{j_u}(x)$ from all neighbours $u \in \Gamma(v)$.
10: **find** smallest $k \in \{0, 1, ..., \mu - 1\}$
such that $p_{i_v}(k) + k \cdot \mu \notin L(v)$ and $p_{i_v}(k) \neq p_{j_u}(k)$ for all $u \in \overline{\Gamma}(v)$.
11: $\psi(v) := p_{i_v}(k) + k \cdot \mu$

We summarize the properties of *Procedure Recolors* below.

Lemma 2.1. *Let $v \in G_i$. The above algorithm satisfies the following four properties:*

1. *The coloring ψ is proper in G_i. (That is, for each $u \in \overline{\Gamma}(v)$ it holds that $\psi(v) \neq \psi(u)$.)*
2. *The coloring ψ is consistent with previously colored vertices. (That is, $\psi(v) \notin L(v)$.)*
3. *The coloring ψ employs $\ell + O\left(\sqrt{\ell} \cdot \beta + \beta^2\right)$ colors.*

4. Procedure Recolor requires one communication round.

A discussion regarding these properties was given above and their proof can be found in [2].

Furthermore, we recall the properties of *Procedure Partition* from [2]:

Lemma 2.2. *[2]* Procedure Partition *computes a vertex disjoint partition into p subgraphs $G_1, ..., G_p$, such that each subgraph has an arboricity of $O\left(\frac{\Delta}{p} \cdot \log_{\frac{q}{p}} \Delta\right)$ in $O\left(q \cdot \log_{\frac{q}{p}} \Delta + \log^* n\right)$ rounds. Moreover, each subgraph G_i comes along with a forest decomposition $\mathcal{F}_i$ of $O\left(\frac{\Delta}{p} \cdot \log_{\frac{q}{p}} \Delta\right)$ forests.*

In Chapter 3, we will make use of the above properties, as we will make a different pre-processing phase before iterating on the subgraphs, using *Procedure Recolor* in each round.

2.2 Constructing an H-Partition

Recall the definition of an H-Partition from the preliminaries section. In this section we will briefly explain how to construct an H-Partition, as detailed in [3] and in [6, ch. 5]. We will assume that all the vertices know the arboricity a (or a common upper bound on a) of the input graph and the number of vertices n. See [3] for a scenario when one of these parameters is not known to the algorithm.

The procedure that computes an H-Partition receives as an input the arboricity a of the graph and an arbitrarily small positive constant $\varepsilon \leq 2$. This parameter will

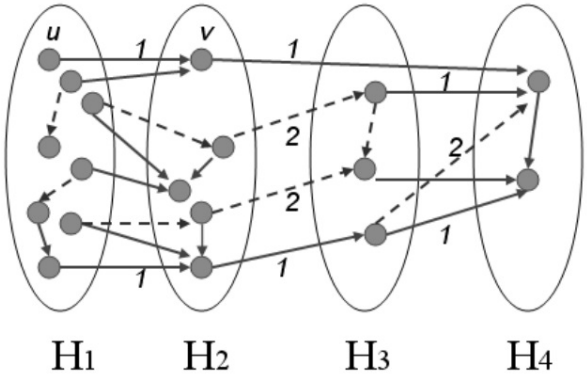


Figure 2.1: H-partition and Forest-Decomposition. An outgoing edge from a vertex u to a vertex v labeled with a label i means that v is the parent of u in a tree of the i^{th} forest F_i. Solid edges represent edges with the label '1'. Dashed edges represent edges with the label '2'.

affect the size and the degree of the H-Partition. A smaller value of ε will result in an H-Partition with a smaller degree, yet with a larger size and will require more running time. Specifically, the procedure computes an H-Partition with degree at most $A = (2+\varepsilon) \cdot a$ and size $l = \left\lceil \frac{2}{\varepsilon} \cdot \log n \right\rceil$

See figure 2.2 for an illustration.

The procedure works by marking each vertex as either 'active' or 'inactive'. An 'active' vertex is a vertex that did not yet join a subset H_i (for $1 \leq i \leq l$), and an 'inactive' one is a vertex that already joined one of them. Initially, all vertices are active. For every $i = 1, 2, ..., l$, in the i^{th} iteration each active vertex with at most $(2+\varepsilon) \cdot a$ active neighbours will join the set H_i, become inactive and will inform all its neighbours that it became inactive.

For completeness, the procedure's pseudo-code is described below:

Algorithm 3 `Procedure Construct` H`-Partition` (a, ε)

This procedure accepts as input the arboricity a of the graph and an arbitrarily small $0 \leq \varepsilon \leq 2$.
Initially all vertices are active.
A procedure to be executed by each vertex $v \in V$:

```
1: for round i = 1, 2, ..., l do
2:    if v is active and has at most (2+ε)·a active neighbours then
3:       make v inactive
4:       add v to H_i
5:       send the messages 'inactive' and 'v joined H_i' to all u ∈ Γ(v).
6:    end if
7:    for each received 'inactive' message do
8:       mark the sender neighbour as inactive
9:    end for
10: end for
```

As extensively detailed in [6, ch. 5], the above procedure computes an H-Partition of degree $A \leq (2+\varepsilon) \cdot a$ and size $l \leq \left\lceil \frac{2}{\varepsilon} \cdot \log n \right\rceil$ in $O(\frac{1}{\varepsilon} \cdot \log n)$ time, for an arbitrarily $0 \leq \varepsilon \leq 2$.

The construction of an H-Partition can be used for computing a forest decomposition of a graph, as detailed in [6, ch. 5]. Yet, we will use it in a different way, as will be described in Chapter 3.

2.3 Barenboim's List Coloring Algorithm

This result of [2] continues the first result mentioned in Section 2.1, which computes a $\Delta + O\left(\sqrt{\Delta} \cdot \log_{\frac{q}{p}} \Delta \cdot \frac{\Delta}{p} + \left(\log_{\frac{q}{p}} \Delta \cdot \frac{\Delta}{p}\right)^2\right)$-coloring in

$O\left(q \cdot \log_{\frac{q}{p}} \Delta + \log^* n\right)$ time [2], for given parameters p and q ($p < q$). First, we would like to discuss the notion of the *Partially-Colored Graph Problem*, which will be used later on for the sake of the *List Coloring Problem*. In this scenario, one is already given a partial coloring φ_0 of $U \subseteq V$ which makes use of at most $\Delta + 1$ colors. This coloring φ_0 is a proper coloring of the induced subgraph on U, yet for vertices $w \in V \backslash U$ the color $\varphi_0(w)$ is not defined. We would like to find a proper $(\Delta + 1)$-coloring ψ of the entire graph that extends φ_0. Note that this is a generalization of the standard vertex coloring problem, in which $U = \emptyset$.

We will use *Procedure Color* which was previously described, yet we will slightly modify it. The modification will be in line 8 of *Procedure Color*. Instead of setting $\psi(v) = \perp$ we will do it only if $\varphi_0(v)$ is not defined. Else, $\psi(v) = \varphi_0(v)$. This algorithm is called *Procedure Mod-Color*. It computes a proper $\Delta + O\left(\sqrt{\Delta} \cdot \log_{\frac{q}{p}} \Delta \cdot \frac{\Delta}{p} + \left(\log_{\frac{q}{p}} \Delta \cdot \frac{\Delta}{p}\right)^2\right)$-coloring of the entire graph $G = (V, E)$ in at most $O\left(q \cdot \log_{\frac{q}{p}} \Delta + \log^* n\right)$ time. The correctness argument of this modified procedure can be found in [2].

We will now return to the discussion of the list-coloring problem. We will call this algorithm *Procedure List-Color*. Let $d \in O(\Delta)$ be an integer such that each vertex $v \in V$ holds a list $R(v)$ of size at least $\Delta + 1$ permitted colors, taken from the range $0, 1, \ldots, d-1$. We will present a reduction from the list-coloring problem to the partially-colored graph problem. The reduction works as follows. For each vertex $v \in V$, we construct the set of forbidden colors $F(v) = \{0, 1, \ldots, d-1\} \setminus R(v)$. Then, for each color $c \in F(v)$ we add a new vertex v_c, an edge (v, v_c) and define $\varphi_0(v_c) = c$. Hence, each vertex $v \in V$ has an additional set of neighbours colored in φ_0 with all of v's forbidden colors. The set of free colors that can still be used by v is exactly the set $R(v)$. Denote the new graph by $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ and its maximum degree $\widetilde{\Delta} = \Delta(\widetilde{G})$. Since $|R(v)| \geq \Delta + 1$, then $|F(v)| \leq d - \Delta - 1$. Hence, each vertex v has at most $d - \Delta - 1$ neighbours of the form v_c (for a color $c \in F(v)$). In addition, it has at most Δ neighbours from the original graph. Thus, it has at most $d - 1$ neighbours in $\widetilde{G}$. That is, $\widetilde{\Delta} < d$. After we generate $\widetilde{G}$, we run *Procedure Mod-Color* on $\widetilde{G}$ and φ_0. Since $\widetilde{\Delta} < d$, *Procedure Mod-Color* will return a coloring ψ which employs at most d colors. By omitting the vertices of the form v_c from ψ, we receive a proper list-coloring of the original graph G.

We will summarize this as follows [2]. Given a graph $G = (V, E)$ with permitted colors lists $R(v)$ for every $v \in V$ such that $|R(v)| \geq \Delta + 1$ and $R(v) \subseteq \{0, 1, \ldots, d-1\}$, *Procedure List-Color* computes a proper list-coloring ψ of G in $O(d^{\frac{3}{4}} \cdot \log d + \log^* n)$ time. If $d \in O(\Delta)$ then the running time is $O(\Delta^{\frac{3}{4}} \cdot \log \Delta + \log^* n)$.

Chapter 3

$O(a)$-Coloring Using Barenboim's Algorithm

Barenboim's algorithm [2], as briefly described in Section 2.1, colors a graph $G = (V, E)$ with a maximum degree Δ by $\Delta + o(\Delta)$ colors in $O(\sqrt{\Delta} \cdot \log^3 \Delta + \log^* n)$ time. The algorithm has a main part - *Procedure Color* - which first constructs a partition of the graph into subgraphs with bounded arboricity, by using *Procedure Partition*, and then colors each vertex, in the order defined by the partition, by calling *Procedure Recolor*. We will describe the specific algorithm and its properties below.

A different notion, which is important to us, is that of an *H-Partition* [3], as defined in Chapter 1 and in [6, ch. 5]. This is a way to partition the graph into several subgraphs and to orient the edges, such that each vertex has an upper bounded out-degree. (with respect to the orientation.)

In our first result, we present an algorithm to color a graph by $(2 + \varepsilon) \cdot a$ colors (for an arbitrarily constant $\varepsilon > 0$), in $O(\sqrt{a} \cdot \log^2 a \cdot \frac{1}{\varepsilon} \cdot \log n)$ time, by combining procedures H-Partition with Barenboim's algorithm. Both the description of Barenboim's Algorithm and of the H-Partition Algorithm can be found in Chapter 2.

Let us recall Theorem 1.1 and Theorem 1.2 from Section 1.3. We will construct a new procedure – *Procedure H-Color* – That will satisfy both of which.

Recall that, given a graph G and a parameter p, *Procedure Partition* partitions G into p subgraphs $G_1, G_2, ..., G_p$. Let $1 \leq i \leq p$. Consider a vertex $v \in V(G_i)$. We will recall the following notions, as they will be used in our algorithm in the same manner as in Chapter 2:

- $L(v)$ – the list of forbidden colors $\psi(u)$ of previously colored neighbours u of v (that is, $u \in V(G_j)$ where $j < i$),

- φ – the temporary coloring, computed by algorithm Arb-Linial, that is proper only within each G_i, for $1 \leq i \leq p$,
- $\overline{\Gamma}(v)$ - the subset of parents of v that belong to G_i,
- $B(v)$ – the colors $\varphi(u)$ of these parents $u \in \overline{\Gamma}(v)$,
- ℓ – an upper bound on the size of $L(v)$ (that is, $|L(v)| \leq \ell$),
- β – an upper bound on the size of $\overline{\Gamma}(v)$ (that is, $|\overline{\Gamma}(v)| \leq \beta$).

Given a graph G and parameters p and q ($p < q$), we will use the following results, as they are already proven in [2]:

1. *Procedure Partition* computes a partition of G into p subgraphs $G_1, ..., G_p$ each having arboricity $a(G_i) \in O\left(\frac{\Delta}{p} \cdot \log_{\frac{q}{p}} \Delta\right)$. This is an $O\left(\frac{\Delta}{p} \cdot \log_{\frac{q}{p}} \Delta\right)$-arbdefective p-coloring, computed in $O\left(q \cdot \log_{\frac{q}{p}} \Delta + \log^* n\right)$ time. We denote it by η.
2. *Procedure Recolor* computes a coloring ψ, which does not conflict with previously colored vertices. That is, for each $v \in V$ and $u \in \overline{\Gamma}(v)$, it holds that $\psi(v) \neq \psi(u)$ and $\psi(v) \notin L(v)$. The coloring employs $\ell + O\left(\sqrt{\ell} \cdot \beta + \beta^2\right)$ colors. Recoloring a single color class of the arbdefective coloring η requires a single communication round.
3. *Procedure Color* computes a $\left(\Delta + O\left(\sqrt{\Delta} \cdot \log_{\frac{q}{p}} \Delta \cdot \frac{\Delta}{p} + \left(\log_{\frac{q}{p}} \Delta \cdot \frac{\Delta}{p}\right)^2\right)\right)$-coloring in $O\left(q \cdot \log_{\frac{q}{p}} \Delta + \log^* n\right)$ time.

Our idea is based upon the above algorithm. We now describe the main phases of our algorithm, which will be referred to as *Procedure H-Color*.

Phase 1 - Constructing an H-Partition

Construct an *H-Partition* of $G = (V, E)$ of length $l = \left\lceil \frac{2}{\varepsilon} \cdot \log n \right\rceil$ and degree $A = (2 + \varepsilon) \cdot a$, for an arbitrarily small $\varepsilon > 0$. This is done in $O(\frac{1}{\varepsilon} \cdot \log n)$ time, using the H-Partition algorithm presented in [6, ch. 5]. Denote the partition by $\mathcal{H} = (H_1, H_2, ..., H_l)$.

Phase 2 - Adjusting the Partition

Given the H-Partition $\mathcal{H} = (H_1, H_2, ..., H_l)$, we employ *Procedure Partition* of [2]. This procedure receives as input a graph $G = (V, E)$, an integer parameter $p > 0$, and an integer parameter $q > p$. For each H_i (where $1 \leq i \leq l$), we will construct, in parallel, an arbdefective coloring η_i with p colors and arbdefect $O\left(\frac{A}{p} \cdot \log_{\frac{q}{p}} A\right)$. To do so, we will use *Procedure Partition* in parallel on each of the subgraphs, in the following way:

1. Any H_i is viewed as the entire graph (unaware of the partitioning process).
2. The parameters p, q are common for all subgraphs H_i. (They will be determined later.)
3. Each subgraph H_i will be partitioned into p subsets $H_{i,1}, H_{i,2}, ..., H_{i,p}$.

Thus, the overall result will be a partition of the vertex set V of the input graph G into $p \cdot l$ subsets – the second-level partitioning.

Phase 3 - Main Coloring Algorithm

After obtaining the second-level partition $H_{1,1}, H_{1,2}, ..., H_{1,p}, ..., H_{l,1}, ..., H_{l,p}$, we use *Procedure Color* of [2], with modified upper bounds as follows: $\ell = A$ and $\beta \in O\left(\frac{A}{p} \cdot \log_{\frac{q}{p}} A\right)$, where $A = (2 + \varepsilon) \cdot a$. Each vertex $v \in H_{i,h}$ takes into account forbidden colors to $L(v)$ both from previously colored subgraphs H_t ($t < i$) and previously colored second-level subgraphs $H_{i,k}$ ($k < h$). The rest of the procedure remains unchanged.

The above algorithm will be referred to as *Procedure H-Color*. We will now further explain it and prove its correctness. First, we will prove the correctness of *Procedure H-Color* depending on the out-degree A of the H-Partition. Later on, we will recall that $A = (2 + \varepsilon) \cdot a$, and by doing so, we will conclude that *Procedure H-Color* can be used to color a graph $G = (V, E)$, with a bounded arboricity $a = a(G)$, by $(2 + \varepsilon) \cdot a$ colors. This generalization will be done in order for *Procedure H-Color* to be available for use in case of other limited out-degree vertex partitions, as will be seen further on in Chapter 4.

Let us begin with analyzing the correctness of *Procedure H-Color*. First, recall that in Phase 1 we used *Procedure Construct H-Partition*. Hence, it follows that:

Corollary 3.1. *[3] After running Phase 1, we receive an H-Partition $\mathcal{H} = (H_1, H_2, ..., H_l)$ of the graph with degree A, where $l \in O(\log n)$, $A \leq (2 + \varepsilon) \cdot a$, and each H_i is a graph with a bounded degree $\Delta_i \leq A$.*

Later on, we apply *Procedure Partition* of Barenboim's algorithm on each of the subgraphs H_i ($1 \leq i \leq l$). Note that by the correctness of *Procedure Construct H-Partition*, we know that each H_i has a bounded outgoing degree of A towards vertices of $\bigcup_{t=1}^{i} V(H_t)$. Let $1 \leq h \leq p$. For a vertex $v \in V(H_{i,h})$, assuming that all vertices in $\left(\bigcup_{t=1}^{i-1} H_t\right) \cup \left(\bigcup_{k=1}^{h-1} H_{i,k}\right)$ were already colored, the size $|L(v)|$ of the set of forbidden colors for v is at most A. Hence, $|L(v)| \leq A$. Recall that in Phase 2, each $H_{i,h}$ in the second-level partition has an arboricity of at most $a(H_{i,h}) \in O\left(\frac{\Delta(H_i)}{p} \cdot \log_{\frac{q}{p}} \Delta(H_i)\right) \leq O\left(\frac{A}{p} \cdot \log_{\frac{q}{p}} A\right)$. Hence the number of parents $|\overline{\Gamma}(v)|$ that a vertex $v \in V(H_{i,h})$ has (in $H_{i,h}$) is bounded by $|\overline{\Gamma}(v)| \leq O\left(\frac{A}{p} \cdot \log_{\frac{q}{p}} A\right)$. Thus:

Corollary 3.2. *After running Phase 2, we have a partition to $l \cdot p$ subsets, and for each vertex v the list $L(v)$ of forbidden colors for v satisfies $|L(v)| \leq A$, and also $|\overline{\Gamma}(v)| \leq O\left(\frac{A}{p} \cdot \log_{\frac{q}{p}} A\right)$.*

That is, the upper bounds on $|L(v)|$ and $|\overline{\Gamma}(v)|$ are A and $O\left(\frac{A}{p} \cdot \log_{\frac{q}{p}} A\right)$, respectively. Thus, we use $\ell = A$ and $\beta = O\left(\frac{A}{p} \cdot \log_{\frac{q}{p}} A\right)$ in our invocations of Barenboim's algorithm. Recall that:

Corollary 3.3. [2] Procedure Recolor *colors v in a color $\psi(v) \notin L(v)$. It uses $\ell + O\left(\sqrt{\ell} \cdot \beta + \beta^2\right)$ colors, where ℓ is an upper bound on $|L(v)|$ and β is an upper bound on $|\overline{\Gamma}(v)|$. The running time is a single communication round for each color class of one of the arbdefective colorings η_i, $1 \leq i \leq l$.*

Note that the above corollary follows directly from the properties of *Procedure Recolor*, that were described in Lemma 2.1 in Section 2.1.

Now we analyze our algorithm. First, we analyze the invocations of *Procedure Recolor* in Phase 3.

Theorem 3.4. *For every pair of indices $1 \leq i \leq l$ and $1 \leq h \leq p$, the second level subgraph $H_{i,h}$ can be properly colored by* Procedure Recolor*, when for each vertex $v \in V(H_{i,h})$, the list of forbidden colors $L(v)$ contains previously colored (by ψ) vertices both from $\bigcup_{k=1}^{h-1} H_{i,k}$ and from $\bigcup_{t=1}^{i-1} H_t$.*

Proof. Consider a vertex $v \in H_{i,h}$. We will prove that $\psi(v) \notin L(v)$. Recall that $\ell = A$ and $\beta = O\left(\frac{A}{p} \cdot \log_{\frac{q}{p}} A\right)$. By Corollary 3.3, we deduce that *Procedure Recolor* can compute a coloring $\psi(v)$ such that $\psi(v) \notin L(v)$. The number of colors used is $\ell + O\left(\sqrt{\ell} \cdot \beta + \beta^2\right)$, and the running time for recoloring a single set $H_{i,h}$ is one communication round.

□

Now we know that *Procedure Recolor* computes a proper coloring which makes use of $\ell + O\left(\sqrt{\ell} \cdot \beta + \beta^2\right)$ colors and is consistent with previously colored vertices. We will prove that the coloring computed in the end of the algorithm is proper as well and uses $(1+\lambda)\cdot A$ colors, for a fixed arbitrarily small constant $\lambda > 0$. Then, we will recall that $A = (2+\varepsilon) \cdot a$, hence making the computed coloring a proper $\left(\left(2+\varepsilon^{'}\right) \cdot a\right)$-coloring, for an arbitrarily small $\varepsilon^{'} > \varepsilon$.

Theorem 3.5. Procedure H-Color *produces a* $((1+\lambda) \cdot A)$*-coloring, for an arbitrarily small constant* $\lambda > 0$.

Proof. Note that by Corollary 3.3, *Procedure Recolor* computes a coloring ψ, which is consistent with previously colored vertices. That is, for each $v \in V$ and $u \in \overline{\Gamma}(v)$, it holds that $\psi(v) \neq \psi(u)$, and also $\psi(v) \notin L(v)$.

Let v be an arbitrarily vertex. Denote the second level subgraph v belongs to by $H_{i,h}$. Recall that in our algorithm, $L(v)$ includes colors of neighbours of v that belong to previously colored subgraphs H_t ($t < i$), and not only neighbours from $H_{i,k}$ ($k < h$). By Theorem 3.4, such a coloring, that takes all of these forbidden colors into account, can be computed by *Procedure Recolor*. Thus, the resulting coloring ψ is proper.

Next, we will show that ψ uses $A + o(A)$ colors. By Corollary 3.2, we can bound $L(v)$ and $\overline{\Gamma}(v)$ with a tighter bound than before. That is, $\ell = A$ and $\beta = O\left(\frac{A}{p} \cdot \log_{\frac{q}{p}} A\right)$. Thus,

$$l + O\left(\sqrt{l} \cdot \beta + \beta^2\right) = A + O\left(\sqrt{A} \cdot \log_{\frac{q}{p}} A \cdot \frac{A}{p} + \left(\log_{\frac{q}{p}} A \cdot \frac{A}{p}\right)^2\right)$$

By Corollary 3.3, it follows that *Procedure Recolor* maintains an $A + O\left(\sqrt{A} \cdot \log_{\frac{q}{p}} A \cdot \frac{A}{p} + (\log_{\frac{q}{p}} A \cdot \frac{A}{p})^2\right)$-coloring by the end of each iteration, for the set of already colored vertices.

Now, set $p = \left\lfloor \sqrt{A} \cdot \log^2 A \right\rfloor$ and $q = 2p$. Hence, we receive an $A + o(A) = ((1+\lambda) \cdot A)$-coloring of G, for an arbitrarily small constant $\lambda > 0$.

□

By now we know the coloring produced is a proper $((1+\lambda) \cdot A)$-coloring of the input graph G. Let us recall the pseudo-code of *Procedure Color*, as was described in Algorithm 1 in Section 2.1:

Algorithm 4 `Procedure Color`

1: $\langle G_1, G_2, ..., G_p, a \rangle$:= *Partition*(G, p, q)
/* a is an upper bound on the arbdefect of the coloring that Procedure Partition returns */
2: **for** $i \leftarrow 1$ to p **in parallel do**
3: φ_i:= *Arb-Linial*(G_i)
4: **if** $v \in V(G_i)$ **then**
5: $\varphi(v) := \varphi_i(v)$
6: **end if**
7: **end for**
8: $\psi(v) := \perp$
9: **for** $i = 1, 2, ..., p$ **do**
10: **if** $v \in V(G_i)$ **then**
11: $L(v) := \{\psi(u) \mid u \in \Gamma(v) \wedge \psi(u) \neq \perp\}$
12: $\overline{\Gamma}(v) := \{u \mid u \in V(G_i) \wedge u$ is a parent of v in $G_i\}$
13: $B(v) := \big\{\varphi(u) \mid u \in \overline{\Gamma}(v)\big\}$
14: $\psi(v) :=$ *Recolor*$\big(v, \varphi, L(v), \overline{\Gamma}(v), B(v), l := \Delta, \beta := a\big)$
15: **end if**
16: **end for**
17: **return** $\psi(v)$

Note that the above algorithm is *Procedure Color*. In *Procedure H-Color*, we first run phases 1 to 3, where phase 3 begins in line 2 of *Procedure Color*, yet with modified upper-bounds, as was explained before.

We will next analyze *Procedure H-Color*'s running time.

Theorem 3.6. *In order to produce a* $((1+\lambda) \cdot A)$*-coloring,* Procedure H-Color *requires* $O(\sqrt{A} \cdot \log^2 A \cdot \frac{1}{\varepsilon} \cdot \log n)$ *time, where* $\varepsilon > 0$ *is an arbitrarily small constant which comes from the size* $l = \left\lceil \frac{2}{\varepsilon} \cdot \log n \right\rceil$ *of the* H*-Partition.*

Proof. Phase 1 requires $O(\frac{1}{\varepsilon} \cdot \log n)$ rounds, as shown in [6, ch. 5], Theorem 5.5.

Phase 2 consists of many parallel activations of *Procedure Partition* from [2]. Note that for each H_i, the local (in H_i) maximum degree, denoted by Δ_i, is bounded by A. That is, $\Delta_i \leq A$ for each $1 \leq i \leq l$. By Lemma 2.2, an activation of *Procedure Partition* (that is, computing an arbdefective coloring) requires $O(q \cdot \log_{\frac{q}{p}} \Delta_i + \log^* n) \leq O(q \cdot \log_{\frac{q}{p}} A + \log^* n)$ rounds. Note that the invocations of *Procedure Partition* are conducted in parallel. Thus the overall running time of Phase 2 remains $O(q \cdot \log_{\frac{q}{p}} A + \log^* n)$.

Note that in Phase 3 we have $l \cdot p$ second level subgraphs. Phase 3 begins in line 2 of Algorithm Color in [2] (the algorithm was described both in Algorithm 1 in Section 2.1 and, for the reader's convenience, its pseudo-code is mentioned above.). The parallel executions of *Procedure Arb-Linial* [3] in lines 2 - 7 require $O(\log^* n)$ time. Line 8 requires one round. Learning the neighbourhoods in lines 11 - 13

requires one additional round, and invoking *Procedure Recolor* in line 14 requires one round as well (See Corollary 3.3). Since there are $p \cdot l$ iterations, lines 9 - 16 will take $O(l \cdot p)$ rounds. Thus, the running time of Phase 3 is $O(l \cdot p) = O(p \cdot \frac{1}{\varepsilon} \cdot \log n)$ rounds.

Hence, the overall running time of *Procedure H-Color* is $O\left(q \cdot \log_{\frac{q}{p}} A + \log^* n + p \cdot \frac{1}{\varepsilon} \cdot \log n\right)$ rounds. Recall that in Theorem 3.5, we chose $p = \left\lfloor \sqrt{A} \cdot \log^2 A \right\rfloor$ and $q = 2p$. By setting the above values of p and q, the time *Procedure H-Color* requires is:

$$\begin{aligned}
&O\left(q \cdot \log_{\frac{q}{p}} A + \log^* n + p \cdot \tfrac{1}{\varepsilon} \cdot \log n\right) && = \\
&O\left(\sqrt{A} \cdot \log^2 A \cdot \log A + \log^* n + \sqrt{A} \cdot \log^2 A \cdot \tfrac{1}{\varepsilon} \cdot \log n\right) && \leq \\
&O\left(\sqrt{A} \cdot \log^2 A \cdot \log n + \log^* n + \sqrt{A} \cdot \log^2 A \cdot \tfrac{1}{\varepsilon} \cdot \log n\right) && = \\
&O\left(\sqrt{A} \cdot \log^2 A \cdot \tfrac{1}{\varepsilon} \cdot \log n\right)
\end{aligned}$$

□

Recall that *Procedure H-Color* is constructed by three phases, as was seen in the beginning of this chapter. Since $\lambda > 0$ is an arbitrarily small constant, we may simplify and select $\lambda = \varepsilon$. We summarize our results below:

Corollary 3.7. Procedure H-Color *colors a graph $G = (V, E)$ partitioned by an H-Partition with out-degree A and size $l = \left\lceil \frac{2}{\varepsilon} \cdot \log n \right\rceil$ properly using $(1+\lambda) \cdot A$ colors in $O\left(\sqrt{A} \cdot \log^2 A \cdot \frac{1}{\lambda} \cdot \log n\right)$ time, for arbitrarily small constants $\varepsilon = \lambda > 0$.*

Now, let us recall that $A = (2 + \varepsilon) \cdot a$. Thus, we conclude the following:

Conclusion 3.8. Procedure H-Color *colors a graph $G = (V, E)$ with arboricity $a = a(G)$ properly using $(2 + \varepsilon') \cdot a$ colors in $O\left(\sqrt{a} \cdot \log^2 a \cdot \frac{1}{\varepsilon'} \cdot \log n\right)$ time, for an arbitrarily small constant $\varepsilon' > 0$.*

Chapter 4

List-Coloring and Coloring of Partially Colored Graphs

In addition to computing a $(\Delta + 1)$-coloring, Barenboim's algorithm can also be used for the case of a *Partially-Colored Graph*, that is, a graph $G = (V, E)$ such that some of its vertices $U \subseteq V$ already have a color (and the rest do not). The algorithm can extend the coloring. Denote this partial coloring by φ_0. This problem is later used to solve the problem of *List-Coloring*, where each vertex $v \in V$ has a set $R(v)$ of at least $\Delta + 1$ permitted colors taken from the range $\{0, 1, \ldots, d-1\}$ in $O(d^{\frac{3}{4}} \cdot \log d + \log^* n)$ time. This result is further detailed and explained in Section 2.3 in Chapter 1 and in [2].

In this chapter, we will construct two new procedures, such that they will satisfy the following theorems: Theorem 1.3, Theorem 1.4 and Theorem 1.5 from Section 1.3.

Recall that *Procedure Mod-Color* differs from *Procedure Color* only on line 8, where instead of initializing the coloring ψ to be null for each vertex, it is initialized as null only if the vertex is not from U. Otherwise, it is given the vertex's initial color. That is, $\psi(v) = \begin{cases} \varphi_0(v), & \text{if } v \in U \\ \perp, & \text{if } v \notin U \end{cases}$.

First, we will look at the problem of Partially-Colored Graphs. In this problem, we are given a graph $G = (V, E)$ and a coloring φ_0 of a subset $U \subseteq V$. We assume that for each $u \in U$, $\varphi_0(u) \in \{0, 1, \ldots, (1+\lambda) \cdot A - 1\}$, where $A = (2+\varepsilon) \cdot a$ for an arbitrarily small $\varepsilon > 0$ and $a = a(G)$ is the arboricity of the graph. The rest of the vertices $v \in V \backslash U$ are uncolored.

We will devise *Procedure H-Mod-Color*, an algorithm similar to *Procedure H-Color*, yet it slightly differs in order for it to suit for the problem of Partial-Coloring.

The difference will be the same as the difference between *Procedure Color* and *Procedure Mod-Color* from [2]. That is, in line 8 of *Procedure Color* we will initialize $\psi(v) = \begin{cases} \varphi_0(v), & \text{if } v \in U \\ \perp, & \text{if } v \notin U \end{cases}$.

Let $v \in V$ be an arbitrarily vertex, and denote the second-level partition it belongs to by $H_{i,h}$ ($1 \leq i \leq l$ and $1 \leq h \leq p$). It is very important to notice that, as in *Procedure H-Color*, generating the list $L(v)$ is done in Phase 3, which actually continues to invoke *Procedure Color*, from a certain starting point (after the graph was repartitioned in a different way than in *Procedure Color*). As defined in *Procedure Color*, the list $L(v)$ is composed of all the colors (by ψ) of neighbours of v. While in *Procedure H-Color*, it was enough for us to take notice only of v's neighbours from previously colored subgraphs H_t ($1 \leq t < i$) and $H_{i,k}$ ($1 \leq k < h$), as they were the only ones to have been colored before v, in *Procedure H-Mod-Color*, we must take notice of all of v's neighbours, whether they belong to a previous subgraph or not – as long as they are neighbours of v and have an initial color – this color must be taken into account. That is, when generating $L(v)$, the colors of all of v's neighbours will be included in $L(v)$.

We will now turn to proving the correctness of *Procedure H-Mod-Color* and analyzing its running time.

First, let us recall the following parameters and their usages, as extendedly detailed in Section 2.3:

- $L(v)$ – the list of forbidden colors $\psi(u)$ of previously colored (whether by *Procedure H-Mod-Color* or by an initial color) neighbours u of v,
- φ – the temporary coloring, computed by algorithm Arb-Linial, that is proper only within each G_i, for $1 \leq i \leq p$,
- $\overline{\Gamma}(v)$ - the subset of parents of v that belong to G_i,
- $B(v)$ – the colors $\varphi(u)$ of these parents $u \in \overline{\Gamma}(v)$,
- ℓ – an upper bound on the size of $L(v)$ (that is, $|L(v)| \leq \ell$),
- β – an upper bound on the size of $\overline{\Gamma}(v)$ (that is, $|\overline{\Gamma}(v)| \leq \beta$).

Now, let us recall Corollary 3.1, Corollary 3.2 and Corollary 3.3, that can be found in Chapter 3:

Corollary 4.1. *The following assertions hold:*

1. *After running Phase 1, we receive an H-Partition $\mathcal{H} = (H_1, H_2, ..., H_l)$ of the graph with degree A, where $l \in O(\log n)$, $A \leq (2+\varepsilon) \cdot a$, and each H_i is a graph with a bounded degree $\Delta_i \leq A$.*

2. *After running Phase 2, we have a partition to $l \cdot p$ subsets, and for each vertex v the list $L(v)$ of forbidden colors for v satisfies $|L(v)| \leq A$, and also $|\overline{\Gamma}(v)| \leq O\left(\frac{A}{p} \cdot \log_{\frac{q}{p}} A\right)$.*

3. *For an uncolored vertex v,* Procedure Recolor *colors v in a color $\psi(v) \notin L(v)$. It uses $\ell + O\left(\sqrt{\ell} \cdot \beta + \beta^2\right)$ colors, where ℓ is an upper bound on $|L(v)|$ and β is an upper bound on $|\overline{\Gamma}(v)|$. The running time is a single communication round for each color class of one of the arbdefective colorings η_i, $1 \leq i \leq l$.*

Let us elaborate on the reason the above remains true in our new scenario. Since assigning a color $\psi(u)$ for a vertex $u \in U$ does not change anything in the partition of the graph at all, Corollary 3.1 and Corollary 3.2 still hold. In regards to Corollary 3.3, we note that for a vertex $u \in U$, its color $\psi(u)$ will not change during the activation of *Procedure H-Mod-Color*. That is, u will be assigned to a certain subgraph, but *Procedure Recolor* will not color it. Instead, it will remain with its original initial color. Hence, *Procedure Recolor* will only color vertices with no initial color. A vertex $v \in V \backslash U$ does not know where the colors of $L(v)$ originated from. From the perspective of v, these are the forbidden colors, regardless of their origin. That is, it does not matter whether a forbidden color in $L(v)$ originated from a color computed by *Procedure Recolor* or from an initial color of a vertex $u \in U$. *Procedure Recolor* will work in the same way. Hence, Corollary 3.3 still holds.

Let us continue with proving the correctness of *Procedure H-Mod-Color*.

Theorem 4.2. *For every pair of indices $1 \leq i \leq l$ and $1 \leq h \leq p$, the second level subgraph $H_{i,h}$ can be properly colored by* Procedure Recolor*, when for each vertex $v \in V(H_{i,h})$, the list of forbidden colors $L(v)$ contains colors that were previously assigned by ψ to v's neighbours with respect to the entire graph G.*

Proof. Consider a vertex $v \in H_{i,h}$. If $v \in V \backslash U$, we will prove that $\psi(v) \notin L(v)$. Recall that $\ell = A$ and $\beta = O\left(\frac{A}{p} \cdot \log_{\frac{q}{p}} A\right)$. By Corollary 3.3, we deduce that *Procedure Recolor* can compute a coloring $\psi(v)$ such that $\psi(v) \notin L(v)$. The number of colors used is $\ell + O\left(\sqrt{\ell} \cdot \beta + \beta^2\right)$, and the running time for recoloring a single set $H_{i,h}$ is one communication round.

Otherwise, if $v \in U$, its color will not change. Furthermore, for each neighbour w of v, we have $\psi(v) = \varphi_0(v) \in L(w)$, as $L(w)$ is defined to include all colors (by

ψ) of neighbours of w. Hence, if $w \in V \backslash U$, its color $\psi(w) \notin L(w)$ (as was proven in the paragraph above), making sure that $\psi(v) \neq \psi(w)$. Otherwise, we have $w \in U$. In this case, we recall that the initial coloring φ_0 is proper. Thus, $\psi(v) \neq \psi(w)$. That is, in any case, for any neighbour w of v, it holds that $\psi(v) \neq \psi(w)$.

□

Now we know that *Procedure Recolor* computes a proper coloring which makes use of $\ell + O\left(\sqrt{\ell} \cdot \beta + \beta^2\right)$ colors and is consistent with previously colored vertices, some of which have been assigned a color by a previous iteration of *Procedure Recolor* and others were given an initial color, which does not change during the execution of the algorithm. We will prove that the coloring computed in the end of *Procedure H-Mod-Color* is proper as well and makes use of the same number of colors as *Procedure H-Color*. This number of colors is at most $(1 + \lambda) \cdot A$, for a fixed arbitrarily small constant $\lambda > 0$, which is the same as $\left(2 + \varepsilon^{'}\right) \cdot a$ colors, for a small $\varepsilon^{'} = 2 \cdot \lambda + \varepsilon + \varepsilon \cdot \lambda$. Note that since λ is arbitrarily small, we may select $\lambda = \varepsilon$.

Theorem 4.3. Procedure H-Mod-Color *produces a* $((1 + \lambda) \cdot A)$*-coloring which extends the initial coloring* φ_0*, for an arbitrarily small constant* $\lambda > 0$*.*

Proof. Note that by Corollary 3.3, *Procedure Recolor* computes a coloring ψ, which is consistent with previously colored vertices. That is, for each $v \in V$ and $u \in \overline{\Gamma}(v)$, it holds that $\psi(v) \neq \psi(u)$, and also $\psi(v) \notin L(v)$.

Let v be an arbitrarily vertex. Denote the second level subgraph v belongs to by $H_{i,h}$. Recall that in *Procedure H-Mod-Color*, the set $L(v)$ contains colors of neighbours of v (and not just those from H_t ($t < i$) and $H_{i,k}$ ($k < h$) [1]). By Theorem 4.2, if $v \in U$ then its color will be different from the colors of its neighbours, and otherwise, if $v \in V \backslash U$, its color will be chosen from a set of colors disjoint from $L(v)$. (That is, not from any of its forbidden colors nor will it be chosen by any other neighbour $u \in \Gamma(v)$). Hence, such a coloring, that takes all of these forbidden colors into account, can be computed by *Procedure Recolor*. For every vertex from U the algorithm did not change its initial color (given by φ_0). Hence, the computed coloring ψ extends the original coloring φ_0. Thus, the resulting coloring ψ is proper and extends φ_0.

Next, we will show that ψ uses $A + o(A)$ colors.

[1] In the general scenario of *Procedure H-Mod-Color*, vertices with an initial color may be found in every H-set. Yet the reader should note, however, that when we will later invoke *Procedure H-List-Color* by using *Procedure H-Mod-Color* as a blackbox, all the vertices with an initial color will be placed in H_0. Hence, the only colored neighbours v will have will belong either to H_0 (auxiliary vertices) or to $\left(\bigcup_{k=1}^{h-1} H_{i,k}\right) \cup \left(\bigcup_{t=1}^{i-1} H_t\right)$ (previously colored vertices from the original graph).

Let us recall we assumed that all of the initial colors of vertices of U are in the range $\{0, 1, \ldots, (1+\lambda) \cdot A - 1\}$, hence φ_0 makes use of at most $(1+\lambda) \cdot A$ colors.

As for the colors of the rest of the vertices $V \backslash U$, by Corollary 3.2, we can bound $L(v)$ and $\overline{\Gamma}(v)$ in the following way: $\ell = A$ and $\beta = O\left(\frac{A}{p} \cdot \log_{\frac{q}{p}} A\right)$. Thus,

$$l + O\left(\sqrt{l} \cdot \beta + \beta^2\right) = A + O\left(\sqrt{A} \cdot \log_{\frac{q}{p}} A \cdot \frac{A}{p} + \left(\log_{\frac{q}{p}} A \cdot \frac{A}{p}\right)^2\right)$$

By Corollary 3.3, it follows that *Procedure Recolor* maintains an $A + O\left(\sqrt{A} \cdot \log_{\frac{q}{p}} A \cdot \frac{A}{p} + (\log_{\frac{q}{p}} A \cdot \frac{A}{p})^2\right)$-coloring by the end of each iteration, for the set of already colored vertices.

Now, set $p = \left\lfloor \sqrt{A} \cdot \log^2 A \right\rfloor$ and $q = 2p$. Hence, we receive an $A + o(A) = ((1+\lambda) \cdot A)$-coloring of G, for an arbitrarily small constant $\lambda > 0$.

□

By now we know the coloring produced is a proper $((1+\lambda) \cdot A)$-coloring of the input graph G which extends the initial coloring φ_0. We will briefly analyze *Procedure H-Mod-Color*'s running time. Note that the only change between *Procedure H-Mod-Color* and *Procedure H-Color* is in the initialization of the colors of ψ. This does not change the running time. Since this is the only difference, their running times are identical.

Corollary 4.4. Procedure H-Mod-Color *requires* $O(\sqrt{A} \cdot \log^2 A \cdot \frac{1}{\varepsilon} \cdot \log n)$ *time, where* $\varepsilon > 0$ *is an arbitrarily small constant which arrives from the size* $l = \left\lceil \frac{2}{\varepsilon} \cdot \log n \right\rceil$ *of the* H*-Partition.*

Recall we chose $\lambda = \varepsilon$. Hence, let us summarize our results below:

Corollary 4.5. Procedure H-Mod-Color *colors a graph* $G = (V, E)$ *that:*

- *has a partial coloring* φ_0*,*
- *and has been partitioned by an* H*-Partition with out-degree* A

properly using $(1+\lambda) \cdot A$ *colors in* $O\left(\sqrt{A} \cdot \log^2 A \cdot \frac{1}{\lambda} \cdot \log n\right)$ *time, for arbitrarily small constants* $\varepsilon = \lambda > 0$*.*

Now, as we did in the end of Chapter 3, let us recall that $A = (2+\varepsilon) \cdot a$. Thus, we conclude the following:

Conclusion 4.6. Procedure H-Mod-Color *extends a partial coloring of a graph* $G = (V, E)$ *with arboricity* $a = a(G)$ *to a proper coloring of the entire graph* G *which makes use of* $(2+\varepsilon') \cdot a$ *colors in* $O\left(\sqrt{a} \cdot \log^2 a \cdot \frac{1}{\varepsilon'} \cdot \log n\right)$ *time, for an arbitrarily small constant* $\varepsilon' > 0$*.*

We return to the problem of List-Coloring. In our scenario, we have an arbitrarily small constant $\lambda > 0$ and an integer parameter $d \in O(A)$, $d \geq (1+\lambda) \cdot A$, such that each vertex $v \in V$ holds a list $R(v)$ of size at least $\frac{\lambda}{1+\lambda} \cdot d + A$ permitted colors, taken from the range $0, 1, \ldots, d-1$. Here A is the out-degree of the H-Partition that was computed for G, i.e., $A = (2+\varepsilon) \cdot a$, for an arbitrarily small constant $\varepsilon > 0$. Our goal is to compute a List-Coloring of G. That is, the objective for each vertex $v \in V$ is to select a color from the list of allowed colors $R(v)$ such that it will not be used by any of its neighbours.

We will present an algorithm which is based both upon *Procedure H-Mod-Color* and the reduction from List-Coloring to Partially-Coloring (as was presented in Section 2.3). The algorithm consists of the following phases:

Phase 1 - Constructing an H-Partition

Construct an *H-Partition* of $G = (V, E)$ of length $l = \lceil \frac{2}{\varepsilon} \cdot \log n \rceil$ and degree $A = (2+\varepsilon) \cdot a$, for an arbitrarily small $\varepsilon > 0$. This is done in $O(\frac{1}{\varepsilon} \cdot \log n)$ time, using the H-Partition algorithm presented in [6, ch. 5]. Denote the partition by $\mathcal{H} = (H_1, H_2, ..., H_l)$.

Phase 2 - Creating the new vertices

Let φ_0 be our partial coloring. For each $v \in V$ and every $c \in F(v)$, where $F(v) = \{0, 1, \ldots, d-1\} \setminus R(v)$, we construct a new vertex v_c and define $\varphi_0(v_c) = c$. In addition, we create the edge (v, v_c).

We define $\hat{G} = \left(\hat{V}, \hat{E}\right)$, where the set of vertices $\hat{V}$ will be the set of all the original vertices $v \in V$ and of all the constructed auxiliary vertices v_c, and the set of edges $\hat{E}$ will be all the original edges from E and all of the newly added edges (v, v_c).

Phase 3 - Repartitioning the new graph

Recall our H-Partition from Phase 1: $\mathcal{H} = (H_1, H_2, ..., H_l)$. We will repartition $\hat{G}$ in accordance with $\mathcal{H}$ and denote the new partition by $\hat{\mathcal{H}}$. All $v \in V$ will remain in the subgraph H_i they were assigned to in Phase 1. The rest of the vertices $\hat{V} \setminus V$ will be put in a new subgraph H_0. Note that each vertex $v \in V(H_i)$ has at most A neighbours from previous H_j's ($0 < j < i$), and in addition at most $d - \left(\frac{\lambda}{1+\lambda} \cdot d + A\right) = \frac{1}{1+\lambda} \cdot d - A$ neighbours from H_0, since it has at least $\frac{\lambda}{1+\lambda} \cdot d + A$ permitted colors from a range of d colors. Hence, each vertex has at

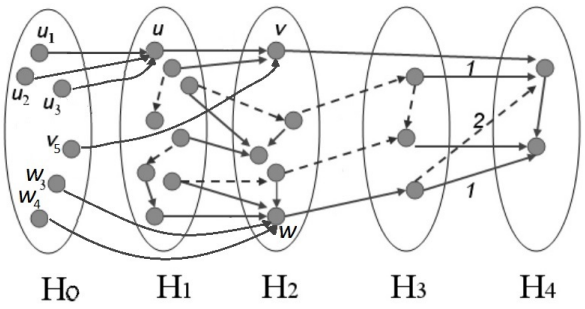


Figure 4.1: An H-partition, where all the auxiliary vertices are in H_0. In the above scenario, assume $F(u) = \{1, 2, 3\}$, $F(v) = \{5\}$ and $F(w) = \{3, 4\}$.

most $A + \frac{1}{1+\lambda} \cdot d - A = \frac{1}{1+\lambda} \cdot d$ neighbours from previous H_j's ($0 \leq j < i$). We will denote this new upper bound on the out-degree by $\hat{A}$. That is, $\hat{A} = \frac{1}{1+\lambda} \cdot d$.

Phase 4 - Running *Procedure H-Mod-Color*

Recall that *Procedure H-Mod-Color* colors a graph using $(1 + \lambda) \cdot A$ colors in $O\left(\sqrt{A} \cdot \log^2 A \cdot \frac{1}{\lambda} \cdot \log n\right)$ time, for an arbitrarily small constant $\lambda > 0$, where A is the out-degree of the H-Partition. In Phase 3, we extended our H-Partition $\mathcal{H}$ to the entire graph $\hat{G}$. Let us look on that extended H-Partition $\hat{\mathcal{H}}$. We will run *Procedure H-Mod-Color* as a blackbox on $\hat{G}$ and the above H-Partition $\hat{\mathcal{H}}$, with the out-degree $\hat{A}$ and size $\hat{l} = l + 1 = \left\lceil \frac{2}{\varepsilon} \cdot \log n \right\rceil + 1$ (because only H_0 was added). This will result in a $((1 + \lambda) \cdot \hat{A})$-coloring ψ of $\hat{G}$, which is also a proper List-Coloring of G.

Denote the above algorithm, constructed by Phases 1 to 4, by *Procedure H-List-Color*. We will discuss its correctness and running time.

First, we would shortly prove that, as was argued before, the degree of $\hat{\mathcal{H}}$ is $\hat{A} = \frac{1}{1+\lambda} \cdot d$.

Lemma 4.7. *The out-degree of the H-Partition of $\hat{G}$ is $\hat{A} = \frac{1}{1+\lambda} \cdot d$, and the number of vertices $\hat{n} \leq n \cdot \frac{1}{1+\lambda} \cdot d$.*

Proof. Let $v \in \hat{V}$. If $v \in \hat{V} \backslash V$, then it has no out-going neighbours since it belongs to H_0. Otherwise – if $v \in V$ – then there exists an index $0 < i < l$ such that $v \in V(H_i)$. Note that v has at least $\frac{\lambda}{1+\lambda} \cdot d + A$ allowed colors in $R(v)$. Since the colors are chosen from the range $\{0, 1, \ldots, d-1\}$, it has at most $d - \left(\frac{\lambda}{1+\lambda} \cdot d + A\right) = \frac{1}{1+\lambda} \cdot d - A$ forbidden colors. Thus, it has at most $\frac{1}{1+\lambda} \cdot d - A$ neighbours from H_0. In addition, it has at most A neighbours from previous H_j's ($0 < j < i$). Thus, it has at most $\frac{1}{1+\lambda} \cdot d$ out-going neighbours. Hence, the out-degree of the H-Partition of $\hat{G}$ is $\hat{A} = \frac{1}{1+\lambda} \cdot d$.

As for the number of vertices, since each vertex has at most $\frac{1}{1+\lambda} \cdot d - A$ forbidden colors and there are n vertices, there are at most $n \cdot \left(\frac{1}{1+\lambda} \cdot d - A\right)$ auxiliary vertices. That is, the amount of vertices in $\hat{G}$ is at most $\hat{n} \leq n \cdot \left(\frac{1}{1+\lambda} \cdot d - A\right) + n \leq n \cdot \frac{1}{1+\lambda} \cdot d$. □

Next, we will prove that the coloring by the end of *Procedure H-List-Color* is a proper list-coloring which makes use of at most d colors.

Theorem 4.8. *Let $\lambda > 0$ be an arbitrarily small constant and let $G = (V, E)$ be a graph where each vertex $v \in V$ has a list of permitted colors $R(v)$ of size at least $\frac{\lambda}{1+\lambda} \cdot d + A$ taken from the range $\{0, 1, \ldots, d-1\}$, where d is an integer parameter such that $d \geq (1+\lambda) \cdot A$, $d \in O(A)$. Then,* Procedure H-List-Color *can be used to produce a proper list-coloring of G (which makes use of at most d colors).*

Proof. By Lemma 4.7, the out-degree of the computed H-Partition in Phase 3 of *Procedure H-List-Color* is at most $\hat{A} = \frac{1}{1+\lambda} \cdot d$. By Corollary 3.7 we know that *Procedure H-Mod-Color* can compute a $((1+\lambda) \cdot \hat{A})$-coloring which extends the partial coloring, for an arbitrarily small constant $\lambda > 0$. We also note that the partial coloring we compute in Phase 2 makes use of at most d colors. Hence, the number of colors will remain d. Thus, *Procedure H-Mod-Color* computes a proper d-coloring of $\hat{G}$. Since it is a proper coloring, for each vertex $v \in V$ and for each one of its forbidden colors $c \in F(v)$: $\psi(v) \neq \psi(v_c) = c$. That is, $\psi(v) \in \{0, 1, \ldots, d-1\}$ and $\psi(v) \notin F(v)$. Thus, $\psi(v) \in \{0, 1, \ldots, d-1\} \setminus F(v) = R(v)$. That is, each vertex $v \in V$ selects a color only from the list of allowed colors $R(v)$. Hence, ψ is also a proper list-coloring of G.

□

We will next analyze the running time of *Procedure H-List-Color*.

Theorem 4.9. *In order to produce a list-coloring,* Procedure H-List-Color *requires $O\left(\sqrt{d} \cdot \log^2 d \cdot \frac{1}{\varepsilon} \cdot \log n\right)$ time, for an arbitrarily small constant $\varepsilon > 0$.*

Proof. Phase 1 requires $O\left(\frac{1}{\varepsilon} \cdot \log n\right)$ rounds, as seen in [6, ch. 5], Theorem 5.5.

Phases 2 and 3 do not not require any communication, as creating the new vertices or assigning the newly created vertices to H_0 does not take any rounds.

Phase 4 is an activation of *Procedure H-Mod-Color* with the parameter $\hat{A} = \frac{1}{1+\lambda} \cdot d$. By Corollary 4.5, its running time is $O\left(\sqrt{\hat{A}} \cdot \log^2 \hat{A} \cdot \frac{1}{\varepsilon} \cdot \log \hat{n}\right)$ rounds. Recall that by Lemma 4.7, we know that $\hat{A} = \frac{1}{1+\lambda} \cdot d$ and $\hat{n} \leq n \cdot \frac{1}{1+\lambda} \cdot d$. Hence, $\log \hat{n} \in O(\log n)$. Thus, the running time of Phase 4 is $O\left(\sqrt{d} \cdot \log^2 d \cdot \frac{1}{\varepsilon} \cdot \log n\right)$.

Since Phase 4 requires most of the running time, the running time of *Procedure H-List-Color* is $O\left(\sqrt{d} \cdot \log^2 d \cdot \frac{1}{\varepsilon} \cdot \log n\right)$ rounds. Recall that $d \in O(a)$. Thus, the running time is $O\left(\sqrt{a} \cdot \log^2 a \cdot \frac{1}{\varepsilon} \cdot \log n\right)$.

□

We will summarize our results:

Corollary 4.10. *A graph $G = (V, E)$ with arboricity $a = a(G)$, where each vertex $v \in V$ has a list of permitted colors $R(v)$ of size at least $\frac{\lambda}{1+\lambda} \cdot d + A$ taken from the range $\{0, 1, \ldots, d-1\}$, where:*

- $A = (2 + \varepsilon) \cdot a$,
- *and* $d \in O(a)$, $d \geq (1 + \lambda) \cdot A$,

can be properly list-colored (by using d colors) in $O\left(\sqrt{a} \cdot \log^2 a \cdot \frac{1}{\varepsilon} \cdot \log n\right)$ time, for an arbitrarily small constant $\varepsilon > 0$.

Chapter 5

Conclusions and Open Problems

In this thesis we discussed several variants of the Vertex Coloring Problem, and we devised new algorithms to solve these problems. The first problem we faced was achieving an $O(a)$-coloring of an input graph $G = (V, E)$. Specifically, as seen in Chapter 3, we devised a procedure called *Procedure H-Color*, which achieves a $((2+\varepsilon)\cdot a)$-coloring in $O\left(\sqrt{a}\cdot\log^2 a\cdot\frac{1}{\varepsilon}\cdot\log n\right)$ time, where $\varepsilon > 0$ is an arbitrarily small constant. Hence, we improved the previously best known result which achieved a $((2+\varepsilon)\cdot a)$-coloring in $O\left(a\cdot\log n\right)$ time.

In Chapter 4, we faced the second and third problems, which are the *Partially Colored Graphs Problem* and *List-Coloring Problem*, respectively. Similarly to [2], we first showed *Procedure H-Mod-Color*, which is an adaption of *Procedure H-Color* for the problem of Partially-Colored Graphs and then we used it in order to devise an algorithm for the List-Coloring Problem. For this scenario, we assumed each vertex $v \in V$ has a list of permitted colors $R(v)$ of size at least $\frac{\lambda}{1+\lambda}\cdot d + A$ taken from the range $\{0, 1, \ldots, d-1\}$, where:

- $A = (2+\varepsilon)\cdot a$,
- $d \geq (1+\lambda)\cdot A$, $d \in O(A)$,
- and $\varepsilon, \lambda > 0$ are arbitrarily small constants.

We then showed an algorithm to compute a proper list-coloring in $O\left(\sqrt{a}\cdot\log^2 a\cdot\frac{1}{\varepsilon}\cdot\log n\right)$ time.

In [5], the authors presented an algorithm to compute a $((2^i+\varepsilon)\cdot a)$-coloring in $O\left(\frac{1}{\varepsilon}\cdot\log n\cdot a^{\frac{1}{i}}\right)$ time, where $i \in \mathbb{N}$ is a given parameter. Their idea is based upon the previously best known result of computing a $((2+\varepsilon)\cdot a)$-coloring in $O\left(a\cdot\log n\right)$ time. As was seen in Chapter 3, we improved the running time for

this problem, hence making it possible for a new algorithm to appear – an algorithm which will compute a $((2^i + \varepsilon) \cdot a)$-coloring in a more efficient running time.

Yet there might still even be room for improvement in our original problem of computing a $((2 + \varepsilon) \cdot a)$-coloring. As we have seen, our work was mostly based upon [2], which showed how to compute a $(\Delta + 1)$-coloring in sublinear (in Δ) time. Another interesting approach would be to use the result of [11], which showed an even faster algorithm for $(\Delta + 1)$-coloring. While the result of [11] does improve the running time of [2] for $(\Delta + 1)$-coloring, it does not improve the running time for $(\Delta + o(\Delta))$-coloring, which is the result *Procedure H-Color* uses. However, the result of [11] is more general in the sense that it solves the problem of *Conflict Coloring* (which can be seen as a generalization of List-Coloring). One may find it interesting to extend our List-Coloring result from Chapter 4 to a more general scenario of Conflict Coloring, by using the algorithm of [11].

Another interesting question is to devise an algorithm for $((2 + \varepsilon) \cdot a)$-coloring that would be more efficient than *Procedure H-Color* from Chapter 3. Such an algorithm does not have to be based upon a $(\Delta + 1)$-coloring algorithm, and it might, in turn, give rise to a more efficient algorithm for $((2^i + \varepsilon) \cdot a)$-coloring, for $i \geq 2$. In addition, it can also be used for the problems of Partially-Colored Graphs and List-Coloring, in similar ways to those that were seen in Chapter 4.

Bibliography

[1] H. Attiya and J. Welch. *"Distributed Computing: Fundamentals, Simulations, and Advanced Topics"*. WILEY, 2004.

[2] L. Barenboim. Deterministic $\Delta + 1$-Coloring in Sublinear (in Δ) Time in Static, Dynamic and Faulty Networks. *Proc. of the 2015 ACM Symp. on Principles of Distributed Computing*, pages 345–354, 2015.

[3] L. Barenboim and M. Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decompositions. *Proc. of the 27th ACM Symp. on Principles of Distributed Computing*, pages 25–34, 2008.

[4] L. Barenboim and M. Elkin. Distributed $\Delta + 1$-Coloring in Linear (in Δ) Time. *Proc. of the 41th ACM Symp. on Theory of Computing*, pages 111–120, 2009.

[5] L. Barenboim and M. Elkin. Deterministic distributed vertex coloring in polylogarithmic time. *Proc. 29th ACM Symp. on Principles of Distributed Computing*, pages 410–419, 2010.

[6] L. Barenboim and M. Elkin. *"Distributed Graph Coloring: Fundamentals and Recent Developments"*. MORGAN & CLAYPOOL PUBLISHERS, 2012.

[7] B. Bollobas. *"Random Graphs"*. Cambridge University Press, 2nd ed., 2001.

[8] R. Cole and U. Vishkin. *"Deterministic coin tossing with applications to optimal parallel list ranking"*. 1986.

[9] L. Cowen, R. Cowen, and D. Woodall. Defective colorings of graphs in surfaces: partitions intosubgraphs of bounded valence. *Journal of Graph Theory*, 10:187–195, 1986.

[10] L. Cowen, W. Goddard, and C. Jesurum. Coloring with defect. *Proc. of the 8th ACM-SIAM Symp. on Discrete Algorithms*, pages 548–557, 1997.

[11] P. Fraigniaud, M. Heinrich, and A. Kosowski. Local Conflict Coloring. *2016 IEEE 57th Annual Symposium on Foundations of Computer Science*, pages 625–634, 2016.

[12] F. Kuhn. Weak Graph Colorings: Distributed Algorithms and Applications. *Proc. of the 21st ACM Symposium on Parallel Algorithms and Architectures*, pages 138–144, 2009.

[13] N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[14] N. Lynch. *"Distributed Algorithms"*. MORGAN KAUFMANN, 1996.

[15] C. Nash-Williams. Decompositions of finite graphs into forests. *J. London Math*, 39:12, 1964.

[16] D. Peleg. *"Distributed Computing: A Locality-Sensitive Approach"*. SIAM, 2000.

[17] N. Santoro. *"Design and Analysis of Distributed Algorithms"*. WILEY, 2000.

תוכן העניינים

1. הקדמה 2

1.1. מבנה העבודה 3

1.2. הגדרות ומונחים 4

1.2.1. זמן הריצה במודל המבוזר 4

1.2.2. מונחי צביעה בסיסיים 4

1.2.3. צביעה-פגומה 4

1.2.4. אוריינטציות 4

1.2.5. חלוקה ליערות 5

1.2.6. יעריות 5

1.2.7. צביעה פגומת-יעריות 6

1.2.8. חלוקת-H 6

1.2.9. צביעה ברשימות 6

1.3. סקירת התוצאות 7

1.4. שיטות וטכניקות 8

2. תוצאות עזר 9

2.1. האלגוריתם התת-לינארי של ברנבוים לְ-$\Delta + 1$-צביעה 9

2.1. בניית חלוקת-H 15

2.1. האלגוריתם לצביעה ברשימות של ברנבוים 16

3. צביעה באמצעות האלגוריתם של ברנבוים-$O(a)$ 18

4. צביעת גרפים צבועים-חלקית וצביעה ברשימות 25

5. מסקנות ובעיות פתוחות 34

תקציר

נתבונן בגרף בעל n קדקודים ודרגה מֵרבית Δ, בו אנו מניחים שכל קדקוד מְיַצֵּג מעבד עצמאי. מעבד של קדקוד u רשאי לתקשר עם מעבד של קדקוד v אם ורק אם $(u,v) \in E$. התקשורת היא סינכרונית, כלומר, מתבצעת לאורך סבבי תקשורת עוקבים. את זמן הריצה נמדוד כמספר סבבי התקשורת שבוצעו.

ב*בעיית צביעת הקדקודים המבוזרת*, מטרתנו היא לצבוע את G באמצעות $\Delta + 1$ (או מעט יותר) צבעים, בזמן ריצה יעיל ככל הניתן. גישה מענינת נוספת לבעיית צביעת הקדקודים המבוזרת היא לחפש צביעה אשר במקום להיות תלויה בדרגה המֵרבית, תהיה תלויה בפרמטר אחר, למשל היעריות (מסומנת כְּ: $a = a(G)$). במקרה זה, אנו נתענין למצוא $O(a)$-צביעה של G. ספציפית, נתור אחר $(2+\varepsilon)\cdot a$-צביעה, באשר $\varepsilon > 0$ הוא קבוע שרירותי קטן. כמו-כן, אנו גם נדון ונסוב לגבי גרסאות כאלו ואחרות של בעית הצביעה בקדקודים, בהן מספר הצבעים יהיה תלוי ביעריות הגרף.

סוגיה שהיתה פתוחה בעבר היתה לגבי קיום או אי-קיום אלגוריתם תת-לינארי (בְּ-Δ) עבור $\Delta + 1$-צביעה. הן ברנבוים [2] והן פראיגניוד, הייריך וקוסובסקי [11] הצליחו להשיב בחיוב לסוגיה זו. תוצאות אלו הובילו אותנו לתהות לגבי אלגוריתמים מהירים יותר עבור צביעה התלויה ביעריות במקום בדרגה המֵרבית. מטרתנו היא להביא תוצאות אלו לידי יישום (ספציפית, את התוצאה של ברנבוים) בכדי למצוא אלגוריתמים יעילים יותר עבור צביעה התלויה ביעריות: הן צביעת קדקודים רגילה והן גרסאות נוספות של בעית הצביעה, אשר נרחיב עליהן בהמשך.

אוניברסיטת בן גוריון בנגב

הפקולטה למדעי הטבע

המחלקה למדעי המחשב

צביעת גרפים מבוזרת

חיבור זה מהווה חלק מהדרישות לקבלת התואר "מוסמך למדעי הטבע (M.Sc)"

מאת אריאל ספיר

תחת ההנחיה של פרופ' מיכאל אלקין

חתימת המחבר: ..... תאריך: 12/03/2019

חתימת המנחה: ..... תאריך: 14/03/2019

חתימת יו"ר ועדת מוסמכים מחלקתית: .....

תאריך: 20/3/19

אוניברסיטת בן גוריון בנגב

הפקולטה למדעי הטבע

המחלקה למדעי המחשב

צביעת גרפים מבוזרת

חיבור לשם קבלת התואר "מגיסטר" בפקולטה למדעי הטבע

מאת אריאל ספיר

תחת ההנחיה של פרופ׳ מיכאל אלקין

כ"ו בכסלו ה׳תשע"ט

04/12/2018