2007

Chromatic scheduling

Rajiv Raman
University of Iowa

CHROMATIC SCHEDULING

by

Rajiv Raman

An Abstract

Of a thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Computer Science
in the Graduate College of
The University of Iowa

July 2007

Thesis Supervisor: Associate Professor Sriram Pemmaraju

ABSTRACT

A classical problem in combinatorics and combinatorial optimization is the graph coloring problem, which asks for the smallest number of colors required to color the vertices of a graph so that adjacent vertices receive distinct colors. Graph coloring arises in several applications of scheduling and resource allocation. However, problems arising in these applications are more general than classical coloring. In this thesis, we present several approximation algorithms and complexity results for such generalized coloring problems.

We deal with the class of *perfect graphs* and sub-classes of perfect graphs on which the classical coloring problem can be solved in polynomial time. However, the coloring problems we consider are NP-hard even on very restricted sub-classes of perfect graphs. The contributions of this thesis are :

- The first constant factor approximation algorithms and complexity results for the *max-coloring problem* on bipartite graphs, interval graphs and an any hereditary class of graphs.
- An improved analysis of the well studied *first-fit* coloring algorithm for interval graphs.
- An experimental evaluation of new heuristics for the max-coloring and interval coloring problem on chordal graphs.

Abstract Approved: _____

Thesis Supervisor

_____

Title and Department

_____

Date

CHROMATIC SCHEDULING

by

Rajiv Raman

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Computer Science
in the Graduate College of
The University of Iowa

July 2007

Thesis Supervisor: Associate Professor Sriram Pemmaraju

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Rajiv Raman

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Computer Science at the July 2007 graduation.

Thesis Committee: _____
Sriram Pemmaraju, Thesis Supervisor

_____
Samuel A. Burer

_____
Suely Oliveira

_____
Alberto M. Segre

_____
Kasturi R. Varadarajan

"To" my mother, for all that she has given me.

ACKNOWLEDGEMENTS

This thesis, and all the results herein would not have been possible without the guidance of Professor Sriram Pemmaraju. I first met Sriram when he was at IIT Bombay, and his courses on Discrete Algorithms and Automata Theory were what got me interested in Theoretical Computer Science. I have greatly benefited from Sriram's generous and sage advise over the years, and I am deeply indebted to him for my growth, both as a researcher and as a person. Sriram has always been there, spending countless hours discussing research ideas, honing my technical presentations, proof-reading my manuscripts, guiding me on job applications, and inviting me to his house for some wonderful dinners. Thank you Sriram.

I would also like to thank Kasturi Varadarajan, whose courses in Computational Geometry were some of the best courses I've taken at the University of Iowa. Kasturi was always there with kind words of encouragement, and sharing his excitement and enthusiasm for research. I have also learnt a lot from Bruno Codenotti, who spent a year visiting the University of Iowa and whose course on Game Theory got me interested in Computational Economics. Bruno's enthusiasm, and friendliness were a great source of inspiration.

I would also thank my committee members: Sam Burer, Suely Oliveira, Alberto Segre, and Kasturi Varadarajan. I would especially like to thank Alberto for carefully reading my manuscript and suggesting changes to Chapter 4 which greatly improved the presentation.

I spent two summers and a semester working at Los Alamos National Labs.

I would like to thanks Anders Hansson, Gabriel Istrate and Stephan Eidenbenz, for providing me with the opportunity to work at the labs, and all the students and interns at the lab for providing an exciting research environment.

I met a great bunch of people at Iowa without whom, life would have been a lot less interesting. I would like to thank, in no particular order, Allison, Amit, Jenelle, Imran, John, Swati, Eli, Sandeep, Ben, Jennifer, Shouxi, Chetan, Aditya, Varsha, Soumik, Hari and Shiv. I apologise in advance to those whose names I forget to mention here, but Thank you.

Going back further in time, I would like to thank Shobana, Kurosh and Iris for encouraging me to leave a cushy job to pursue a graduate degree.

Finally, I would like to thank my brother and my mother for their constant love, support and encouragement.

ABSTRACT

A classical problem in combinatorics and combinatorial optimization is the graph coloring problem, which asks for the smallest number of colors required to color the vertices of a graph so that adjacent vertices receive distinct colors. Graph coloring arises in several applications of scheduling and resource allocation. However, problems arising in these applications are more general than classical coloring. In this thesis, we present several approximation algorithms and complexity results for such generalized coloring problems.

We deal with the class of *perfect graphs* and sub-classes of perfect graphs on which the classical coloring problem can be solved in polynomial time. However, the coloring problems we consider are NP-hard even on very restricted sub-classes of perfect graphs. The contributions of this thesis are :

- The first constant factor approximation algorithms and complexity results for the *max-coloring problem* on bipartite graphs, interval graphs and an any hereditary class of graphs.
- An improved analysis of the well studied *first-fit* coloring algorithm for interval graphs.
- An experimental evaluation of new heuristics for the max-coloring and interval coloring problem on chordal graphs.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

CHAPTER 1
INTRODUCTION

In this dissertation, we study several graph coloring problems and present approximation algorithms, online algorithms and hardness results for these problems. In the classical graph coloring problem, we are given a graph $G = (V, E)$ and the objective is to partition the vertex set V of G into the fewest number of independent sets. The general model of the problems we study in this dissertation are as follows: We are given a graph $G = (V, E)$, along with a weight function on the vertices $w : V \rightarrow \mathbf{N}$. We wish to partition the vertex set of this graph into color classes (i.e., independent sets) while minimizing a certain global objective function that depends on the vertex weights. For example, in the *max-coloring problem* the input is a graph $G = (V, E)$, along with vertex weights $w : V \rightarrow \mathbf{N}$. The task is to partition the vertex set V of G into *independent sets* $S_1, \cdots, S_k$, that minimizes the sum of the maximum weight of a vertex in each S_i. In other words, if $S = \{S_1, \cdots, S_k\}$ is a partition of V into color classes, then the cost of this partition, $cost(S)$ is

$$cost(S) = \sum_{i=1}^{k} max_{v \in S_i} w(v)$$

The objective is to find a feasible partition that has minimum cost.

Graph coloring is a fundamental combinatorial optimization problem arising in several applications. The problem has been traditionally motivated by applications in time-tabling, scheduling, resource allocation, etc. but also arises in diverse areas

such as information theory and statistical physics. The problems in this dissertation are motivated by applications in resource allocation and scheduling. For both problems, the vertices of a graph correspond to jobs. Each job requires some non-sharable resource to run, and if two jobs require the same resource, they cannot be run simultaneously and hence are in *conflict*. This pairwise conflict relation is given by the edges of the graph. Since a coloring partitions the jobs into independent sets, the set of jobs in each color class can be run simultaneously. Although the classical graph coloring problem is well studied, and is hopelessly hard to approximate [28], the graphs arising in applications tend to have special structure that makes the coloring problem relatively easy. However, we are not interested in just solving the classical coloring problem that requires the use of fewest number of color classes; we are interested in obtaining a coloring whose cost is a function of vertex weights, as in the case of max-coloring. As we will show in later chapters, such problems are hard even for graph classes for which classical coloring is easy. Our approach to solving these coloring problems are through the development of approximation algorithms, online algorithms and efficient heuristics. Approximation algorithms are algorithms that are guaranteed to produce solutions close to the optimum, while online algorithms produce solutions without knowing future inputs. However, both approximation and online algorithms give bounds on the worst case behavior of the algorithm and say nothing about the performance of the algorithm on real-life instances. In order to gain insight into the practical performance of the algorithms, we empirically study the performance of our algorithms.

In Section 1.1, we present some of the applications that motivate our work. In Section 1.2, we present some of the basic terminology used throughout this dissertation. Finally in Section 1.3 we present the main contributions of this dissertation.

1.1 Applications

1.1.1 Scheduling Conflicting Jobs

In several scheduling applications, jobs that need to be scheduled require some non-sharable resource to run. For example, processes in a distributed operating system that need access to the same file or device cannot be scheduled simultaneously. As described earlier, such a scheduling problem can naturally be modeled as a graph coloring problem. However, not all jobs or processes are the same and each process may have different running times. This gives rise to more general coloring problems where each vertex has weight corresponding to its running time and we are interested in minimizing the average completion time of the jobs or the time by which the last job completes, etc. Further options such as whether jobs can be pre-empted, whether jobs have to be executed in batches, etc. give rise to other variants.

1.1.2 Dynamic Storage Allocation

Static variables in a program are assigned memory during compilation. A simple way to do this is to assign to each variable a separate location. However, this is extremely wasteful since not all variables may be *alive* at the same time. Modern compilers construct a *conflict graph* of the variables and use this graph to assign memory regions to variables such that variables alive at the same time have non-intersecting memory

regions. Fabri [27] and Ershov [25] modeled the problem of compile-time memory allocation as a coloring problem on graphs. Consider a simple program without loops or branching statements. Each variable has a start-time and an end-time, and a certain memory size requested. This can be modeled as an *intersection graph* of intervals on the real line, or an *interval graph* where intervals correspond to variable life times and weights on the vertices correspond to the requested size. Memory is modeled as a linear array. The objective is to assign each vertex a contiguous region in the memory array, whose length is equal to the weight of the vertex, such that adjacent vertices(i.e., variables alive simultaneously) are assigned non-intersecting regions. The objective is to minimize the total memory used to assign space to all the variables.

1.1.3 Buffer Minimization

Programs that run with stringent memory or timing constraints use a dedicated memory manager that provides better performance than the general purpose memory management of the operating system. The *buffer minimization* problem arises in the context of designing memory managers for wireless protocol stacks like GPRS or 3G. With the rapid growth of wireless communication devices, many telecommunication companies now license their wireless protocol stacks to vendors of mobile devices. These protocols stacks have stringent memory requirements as well as soft real-time constraints and a dedicated memory manager is a natural design choice. A dedicated memory manager for these stacks must have deterministic response times and use as little memory as possible. The most commonly used memory manager design for this

purpose is the *segregated buffer pool* [69]. This consists of a fixed set of buffers of various sizes with buffers of the same size linked together in a linked list. As each memory request arrives, it is satisfied by a buffer whose size is large enough.

The assignment of buffers to memory requests can be viewed as an assignment of colors to the requests – all requests that are assigned the same buffer are colored identically. In the on-line version, buffers have to be allocated to requests as they arrive and without knowledge of future requests. The off-line version is also useful because designers of protocol stacks would like to estimate the size of the total memory block needed by extracting large traces of memory requests and running off-line algorithms on these traces. This buffer minimization problem was studied by Pemmaraju, et al. [63], and also by Govindarajan , et al. [36] in the context of Digital Signal Processing applications.

1.2 Background and Notation

1.2.1 Approximation Algorithms

As stated before, most of the problems we consider in this dissertation are NP-hard and we develop approximation algorithms for these problems. An NP-optimization problem Π is either a maximization or a minimization problem. Each valid instance I of Π comes with a non-empty set of feasible solutions, each of which is assigned a non-negative rational number called the objective function value. A feasible solution that achieves the optimal objective function value is called an *optimal solution*, denoted $OPT_{\Pi}(I)$ for instance I. We will shorten this to $OPT(I)$, or OPT, when there is no ambiguity. For example, the valid instances for the graph coloring

problem are all graphs $G = (V, E)$. A feasible solution is a partition of the vertex set V of G into independent sets, and the objective function value is the number of independent sets in our solution. Since graph coloring is a minimization problem, the objective is to minimize the number of independent sets required to partition the vertex set of G. An algorithm is an *approximation algorithm* if it runs in polynomial time, and is guaranteed to produce a solution that is "close" to the optimal objective value. The precise definition is below.

Definition 1.2.1 *An approximation algorithm $\mathcal{A}$ for a minimization problem Π is called an α-approximation algorithm if for any instance I of Π, $\mathcal{A}$ runs in polynomial time, and produces a solution whose value is at most $\alpha \cdot OPT(I)$ for problem instance I, where $\alpha > 1$. For a maximization problem Π, an algorithm $\mathcal{A}$ is called an α-approximation algorithm if on any instance I of Π, $\mathcal{A}$ runs in polynomial time and produces a solution whose value is at least $OPT(I)/\alpha$, where $\alpha > 1$.*

For some NP-optimization problems, even computing an approximate solution can be hard, i.e., computing a sufficiently small approximation to the objective function implies P=NP (or some other well known complexity conjecture believed to be unlikely). More precisely, a minimization problem Π is inapproximable beyond a threshold α if the existence of a β-approximation algorithm, with $\beta < \alpha$ implies P=NP. Combinatorial optimization problems vary widely with respect to approximability. Some problems admit approximations that are *very close* to the optimum objective function value. For example the *Knapsack Problem* can be approximated to a $(1 + \epsilon)$ factor for any $\epsilon > 0$ by an algorithm that runs in time that is polynomial

in both the input size and $1/\epsilon$ [60]. Such approximation algorithms are called a *fully polynomial time approximation scheme*, or FPTAS. Some problems are not so well behaved and admit a $(1+\epsilon)$ factor approximation, for any $\epsilon > 0$, but only with an algorithm that runs in time that is polynomial in the input size and exponential in $1/\epsilon$. For example the problem of machine scheduling to minimize *makespan* [45]. Such problems are called a *polynomial time approximation scheme*, or PTAS.

Other problems, such as *Vertex Cover* have approximation algorithms that guarantee a constant factor approximation, but it is unlikely that they can do any better approximation. For *Vertex Cover*, there is an easy 2-approximation algorithm, while the problem is NP-hard to approximate beyond a threshold of 1.36 [18]. For problems like *Set Cover* however, even a constant factor approximation is unlikely to exist. The best approximation algorithms guarantee a $\lg n$ approximation, and the problem is inapproximable beyond $\lg n$ [29].

Still other problems turn out to be even harder to approximate than Set Cover. Graph Coloring, for example, is inapproximable beyond a threshold of $n^{1-\epsilon}$ for any $\epsilon > 0$ [28]. For further results and definitions on approximation algorithms and inapproximability results, see [3, 68].

1.2.2 Online Algorithms

In many applications, all the data are not available at the start of the algorithm, and the algorithm must make decisions as and when data items arrive. Algorithms that operate without making any assumptions about the future data elements, but make decisions using only the information about the current and past

data items are called online algorithms. Online algorithms are usually required to run in polynomial time. The performance of an online algorithm for an optimization problem is typically measured by its *competitive ratio*. For example, consider the *Online Dynamic Storage Allocation problem*. In this problem, we have a linear size array W in which we wish to allocate space to the variables of a program as and when they arrive. When a variable v arrives, it requests a contiguous space of size $w(v)$ in the array. The algorithm must allocate this space for the variable before satisfying future requests, such that variables alive at the same point in time are never assigned overlapping spaces in the array. Once a variable has been allocated space in W, it cannot be reassigned. The objective of the algorithm is to satisfy all requests while using the smallest size array W [55].

Definition 1.2.2 *An online algorithm $\mathcal{A}$ is said to be c-competitive for a minimization problem Π, if for any instance I of Π,*

$$\mathcal{A}(I) \leq c \cdot OPT(I)$$

Notice that here the performance of the online algorithm is being compared to the optimal "offline" cost. We usually have the restriction that algorithm $\mathcal{A}$ runs in polynomial time. For the Online Dynamic Storage Allocation problem (DSA), the competitive ratio of an algorithm $\mathcal{A}$ is the ratio of the size of the array, W, used by the online algorithm $\mathcal{A}$ to the optimum space required by an offline algorithm that has all the variable requests at the start of the algorithm. As we shall see in Chapter

2, if all the requests are 1, then the problem reduces to the *online interval graph coloring problem*, which has a 3-competitive algorithm. However, Online DSA has only an $O(\log n)$-competitive algorithm. See the book by Borodin and El-Yaniv [9] for more on online algorithms.

1.2.3 Graph Theory

A graph $G = (V, E)$, in this dissertation refers to a simple, and finite graph. We typically use n to denote the number of vertices, and m to denote the number of edges. All graph classes in this dissertation are *hereditary*. A class $\mathcal{G}$ of graphs is hereditary if for any $G \in \mathcal{G}$, and any vertex induced subgraph G' of G, $\mathcal{G}$ contains G'. Given a collection $\mathcal{S}$ of subsets of a ground set S, the *intersection graph* of $\mathcal{S}$ is a graph whose vertex set is $\mathcal{S}$ and two sets $A, B \in \mathcal{S}$ are adjacent if $A \cap B \neq \emptyset$.

A *coloring* of a graph is a map $c : V \to S$, where S is some discrete set such that if $\{u, v\} \in E$, then $c(u) \neq c(v)$. The classical graph coloring problem seeks to find a map c such that $|\{s \in S \mid c(v) = s\}|$ is minimized. Figure 1.1 shows a cycle of size 5, colored with 3 colors. The *chromatic number* of a graph, denoted $\chi(G)$ is the smallest size of the set S such that a feasible coloring exists. i.e., the set of colors used is minimized. A *clique* in a graph is a subset $C \subseteq V$ such that all vertices in C are adjacent to each other. The *clique number* of a graph, denoted $\omega(G)$ is the largest size of any clique $C \subseteq V$. An *independent set* of a graph is a collection of vertices $S \subseteq V$ such that no vertices in S are adjacent to each other. The *maximum independent set* is a set $S \subseteq V$ of maximum cardinality. The *stability number* or *independence number*, denoted $\alpha(G)$ is the cardinality of the maximum independent

set.

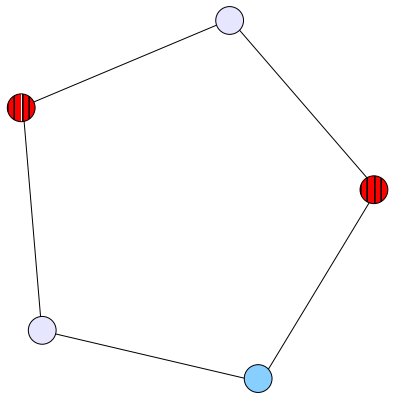

Figure 1.1: The graph C_5 and a vertex coloring. $\chi(C_5) = 3$, although $\omega(C_5) = 2$

It is easy to see that for that for all graphs G, $\omega(G) \leq \chi(G)$ holds, since all the vertices in a clique must be colored with distinct colors. However, $\omega(G)$ in general does not provide a good lower bound for coloring. Indeed, Erdös showed that for every $k \in \mathbf{N}$, there is a graph such that $\omega(G) = 2$ and $\chi(G) > k$ [24].

Algorithms to compute the chromatic number, or a coloring of a graph have been studied by several researchers. Graph Coloring was one of the 21 problems shown NP-complete in the celebrated paper of Karp [48]. Graph Coloring is also hopelessly hard to approximate. Feige and Kilian [28] showed that no approximation algorithm for Graph Coloring exists with approximation ratio better than $n^{1-\epsilon}$, for any $\epsilon > 0$, assuming $NP \not\subseteq ZPP$[1]. The best known approximation algorithm for

[1]ZPP is the class of problems solvable by randomized algorithms that always returns the correct answer, and whose expected running time is polynomial

general graphs is by Karger, Motwani and Sudan [47]. Infact, even for 3-colorable graphs, the best known approximation algorithms only guarantee an approximation factor of $n^{0.2111}$ [2].

1.2.4 Perfect Graphs and Sub-Classes of Perfect Graphs

An important class of graphs on which coloring can be solved in polynomial time is the class of *perfect graphs*. Perfect graphs were introduced by Berge [7] in 1960 in studying a problem related to information theory. Since then, perfect graphs have become an important object of study in mathematics, finding applications in several diverse areas, and are intimately connected to integer programming. There are several characterizations of perfect graphs, among which the one that we will use is the following.

Definition 1.2.3 *A graph G is* perfect *if and only if for all vertex induced subgraphs G' of G, we have $\chi(G') = \omega(G')$.*

For example, the graph in Figure 1.1 is not perfect since $\chi(G) = 3$, but $\omega(G) = 2$. Perfect graphs contain many important and interesting sub-classes of graphs that arise in several applications. Trees and bipartite graphs clearly satisfy the characterization of perfect graphs described above. We now define some other graph classes that are also perfect. Further definitions and properties of these sub-classes of perfect graphs can be found in the book by Golumbic [35].

Definition 1.2.4 *A graph G is an* interval graph *if the vertices of G can be placed in one-to-one correspondence with finite segments on the real line so that two vertices*

are adjacent if and only if the corresponding intervals intersect. Such a representation as an intersection graph of line segments is called the interval representation of the graph.

Figure 1.2 shows an interval graph along with its interval representation. Intervals can be recognized in linear time. i.e., given a graph G as input, we can test whether the graph is an interval graph, and if it is, produce its interval representation [17]. Coloring interval graphs can also be solved in polynomial time by processing the intervals in order of their left end-points and assigning the smallest available color to each interval [35].

A generalization of interval graphs are chordal graphs, which are defined as follows.

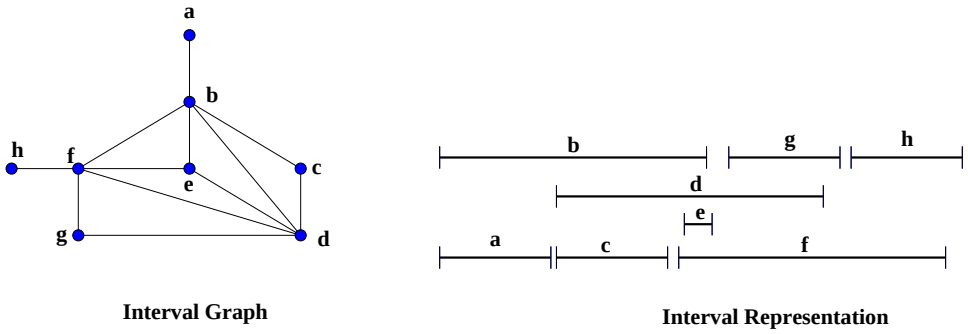

Figure 1.2: An interval graph and an interval representation

Definition 1.2.5 *A graph G is chordal if and only if it does not contain a chordless cycle of length 4 or more as an induced subgraph.*

The fact that interval graphs are a generalization of chordal graphs becomes clear from the following characterization of chordal graphs.

Definition 1.2.6 *A graph G is a chordal graph if it is an intersection graph of a family of subtrees of a tree.*

If the underlying tree were a path, then the corresponding graph is an interval graph. However, a characterization that is more useful to us is the characterization in terms of a *perfect elimination order*.

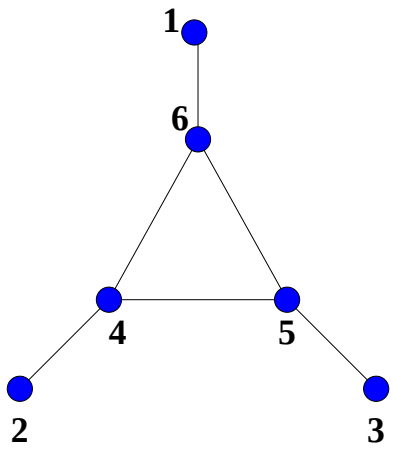

Figure 1.3: A chordal graph with a perfect elimination order

Definition 1.2.7 *A* perfect elimination order *is an ordering of the vertices of G into $(v_1, \cdots, v_n)$ such that for each v_i the vertices adjacent to v_i in $(v_{i+1}, \cdots, v_n)$ form a clique.*

Chordal graphs are exactly the class of graphs that possess a perfect elimination order. Furthermore, since perfect elimination orders can be computed for a

chordal graph, it is easy to recognize chordal graphs. We can compute a coloring of a chordal graph in polynomial time by processing the vertices in reverse perfect elimination order, and assigning to a vertex the smallest available color [35]. Figure 1.3 shows a chordal graph with its vertices numbered in a perfect elimination order. To see that every interval graph G is a chordal graph, note that ordering the vertices of G in "left-to-right" order of the right end-points of the corresponding intervals yields a perfect elimination ordering of G.

For perfect graphs in general, the recognition problem was solved in the breakthrough result of Chudnovsky et al. [15]. Perfect graphs can be colored in polynomial time using the *ellipsoid algorithm*. Grötschel, Lovász and Schrijver [38] showed the first and only polynomial time algorithm to color perfect graphs. However, since the algorithm is based on the ellipsoid method, it is too slow to be used in practice.

Theorem 1.2.8 ([38]) *Given a graph $G = (V, E)$ that is perfect, we can compute a minimum coloring of G in polynomial time.*

1.2.5 Non-perfect Graph Classes

In this section, we define some classes of graphs encountered in this dissertation that are not perfect. However, all the graphs in this section satisfy the *hereditary property*. The classical Graph Coloring problem on the classes of graphs defined in this section are NP-hard, but these classes of graphs enjoy approximation algorithms with a constant approximation ratio.

Similar to the definition of interval graphs that are intersection graphs of intervals on the real line, we can define intersection graphs of arcs of a circle. Such

graphs are called *circular-arc graphs*. Figure 1.4 shows a circular-arc graph and its representation in the form of arcs of a circle.

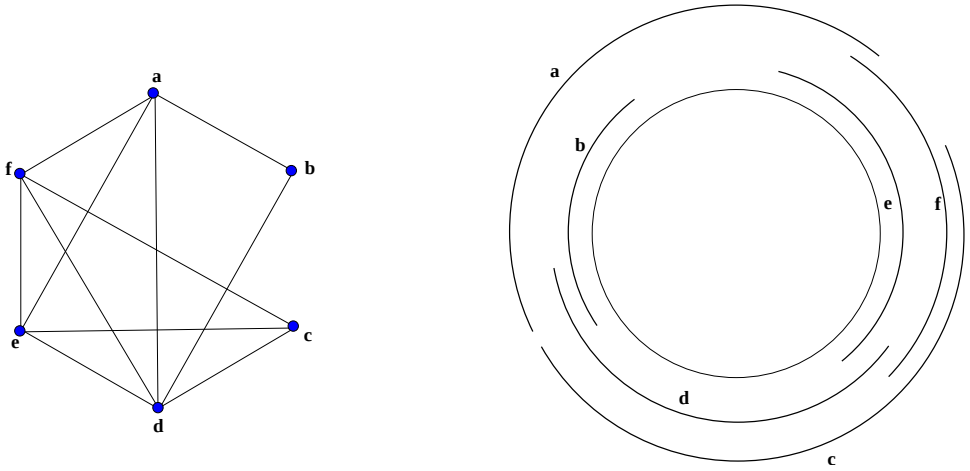

Figure 1.4: An example of a circular-arc graph and its representation as arcs of a circle

Definition 1.2.9 *A graph $G = (V, E)$ is called a* circular-arc graph *if it can be represented as an intersection graph of arcs of a circle.*

It is easy to see that circular-arc graphs are not perfect, since circular-arc graphs contain the odd-cycle graphs C_{2k+1}, $k = 2, \cdots$. Inspite of their apparent simplicity, the classical coloring problem on circular-arc graphs is NP-complete [32]. Circular-arc graphs contain interval graphs, since any interval graph can be represented as arcs of a circle such that there is some point on the circumference of the circle *not* contained in any arc. Hsu and Shih [66] gave a 5/3-approximation algorithm for coloring circular-arc graphs. However, circular-arc graphs can be recognized in linear time [56].

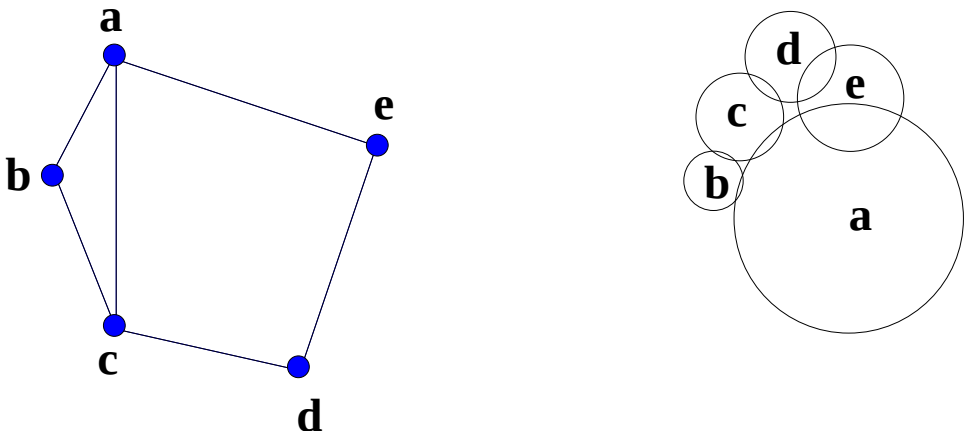

Figure 1.5: An example of a disk graph and its representation as an intersection of disks

A graph is a *disk graph* if can be represented as an intersection graph of closed disks in the plane. The special case, where each disk has the same radius is called a *unit disk graph*. Figure 1.5 shows a graph and its representation as a collection of disks. The classical graph coloring problem remains NP-hard even on unit disk graphs [8]. Gräf [37] gave a 5-approximation algorithm to color disk graphs. However, the recognition problem for disk graphs and unit disk graphs is NP-hard [10, 44].

1.3 Organization and Overview of Contributions

In this section, we describe the main contributions of this dissertation.

1.3.1 First-fit Coloring

In an *online graph coloring problem*, vertices of a graph are presented one by one to the algorithm. When a vertex is presented the edges to the vertices previously presented are also revealed to the algorithm. The algorithm must assign a color to a

vertex, before the next vertex is presented. A color once assigned to a vertex cannot be changed. A natural algorithm to solve the online coloring problem is *first-fit*. The first-fit coloring algorithm is simply to assign the smallest available color to a vertex when it is presented.

The online coloring problem for interval graphs was first studied in the context of the Dynamic Storage Allocation problem. Chrobak and Slusarek [14] introduced an algorithm, called *Buddy Decreasing Size* (BDS) to solve the Dynamic Storage Allocation problem, that used first-fit coloring of interval graphs as a subroutine. They also showed that if first-fit coloring of interval graphs is c-competitive, then BDS is a $2c$-approximation algorithm for DSA. The online coloring problem for interval graphs can be seen as a geometric packing problem as follows. Think of each interval as marking off a segment of the x-axis; furthermore view each interval as a unit height rectangle with height dimension parallel to the y-axis. When a rectangle arrives, the algorithm must assign y-coordinates to the rectangle, while the x-coordinates of the rectangle cannot be altered. In other words, the algorithm can only slide each given interval up or down in the vertical strip defined by the endpoints of the interval. The first-fit algorithm assigns the smallest possible y-coordinate when an interval arrives. The objective is to minimize the maximum y-coordinate assigned to any rectangle. As Figure 2.1 shows, first-fit can be easily forced to use more colors than the chromatic number of the graph.

First-fit is widely used in practice since it is easy to implement and works extremely well on instances arising in practice. However, a tight analysis of the

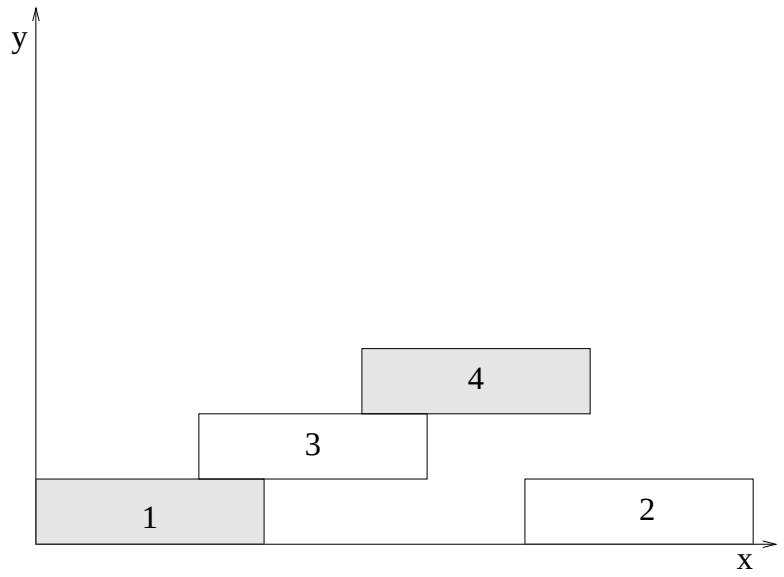

Figure 1.6: An instance of a problem where fist-fit is forced to use more colors than the chromatic number

performance of first-fit has proved to be elusive, despite efforts by several researchers. Woodall [70] in 1974 presented the first upper bounds on the performance of first-fit. He showed that first-fit is $O(\log n)$ competitive. Then, Kierstead [50] in 1988 showed that the competitive ratio of first-fit is a constant. He showed that first-fit is 40-competitive. Kierstead and Qin [52] in 1995, improved the bound to 26. In Chapter 2, we present a new analysis that proves that first-fit is 8-competitive, using a much simpler analysis. We then use this new analysis for a related problem of *online coloring intervals with bandwidth*, introduced by Adamy and Erlebach [1]. The problem of *online coloring of intervals with bandwidths* is motivated by the problem of allocating bandwidth in an all optical WDM (wavelength-division-multiplexing) network, minimizing total duration in a call scheduling problem, and minimizing

the number of machines used in a job scheduling problem in which each bandwidth represents the fraction of a machine that can be used by a job. We present an 30-competitive algorithm, in comparison to the 195-competitive algorithm presented by Adamy and Erlebach [1].

1.3.2 Max-Coloring

The max-coloring problem is a generalization of the classical vertex coloring problem. The input to the max-coloring problem is a graph $G = (V, E)$ along with a weight function on the vertices, $w : V \rightarrow \mathbf{N}$. The task is to compute a proper vertex coloring of G that minimizes the sum of the costs of the color classes, where the cost of a color class is the weight of a maximum weight vertex in the color class. More precisely, if $\mathcal{S}$ is the collection of all valid colorings of the vertex set of G, then the objective is to find a coloring $S = \{S_1, \cdots, S_k\} \in \mathcal{S}$, that minimizes the following objective function

$$cost(S) = \sum_{i=1}^{k} max_{v \in S_i} w(v)$$

Notice that if all vertex weights are 1, then the max-coloring problem reduces to classical coloring. However, if the weights are not all equal, then the max-coloring problem may use more colors than the chromatic number to minimize the cost of the coloring. Figure 1.7 shows such an example. The path clearly admits a coloring with 2 colors, namely $\{\{a, c\}, \{b, d\}\}$. However, this coloring has a cost of 6, while using a coloring with 3 colors, namely $\{\{a, d\}, \{b\}, \{c\}\}$, we can decrease the cost of the

coloring to 5. In fact, there are examples of graphs where $\chi(G) = 2$ and the number of colors used in a minimum cost max-coloring is $n/2$ (See Figure 3.6 in Chapter 3).

Figure 1.7: An example where the optimum max-coloring uses more colors than the chromatic number of the graph. The weights of the vertices are shown above the corresponding vertices. On the left is a coloring with 2 colors that has a max-coloring cost of 6. On the right is the same graph, colored with 3 colors, whose max-coloring cost is 5.

The max-coloring problem was originally motivated by the problem of buffer allocation described in Section 1.1. Recall that the buffer allocation problem is the problem of estimating the smallest size of a *segregated buffer pool* required to allocate all variables of a program. A segregated buffer pool consists of a fixed set of buffers of various sizes with buffers of the same size linked together in a linked list. As each memory request arrives, it is satisfied by a buffer whose size is large enough to satisfy the request.

The assignment of buffers to memory requests can be viewed as an assignment of colors to the requests – all requests that are assigned a buffer are colored identically. Thus the problem of determining whether a given segregated buffer pool suffices for a particular sequence of allocation requests can be formalized as follows. Let $G = (V, E)$ be a graph whose vertices are objects that need memory and whose edges connect pairs of objects that are alive at the same time. Let $w : V \rightarrow N$ be a weight function that

assigns a natural number weight to each vertex in V. For any object v, $w(v)$ denotes the size of memory it needs. Suppose the segregated buffer pool contains k buffers with weights $w_1, w_2, \ldots, w_k$. The problem is to determine if there is a k-coloring of G into color classes $C_1, C_2, \ldots, C_k$ such that for each i, $1 \leq i \leq k$, $\max_{v \in C_i} w(v) \leq w_i$. The optimization version of this problem is the max-coloring problem.

Solving the max-coloring problem and selecting k buffers of sizes $\max_{v \in C_i}\{w(v)\}$, for $i = 1, 2, \ldots, k$, leads to a segregated buffer pool that uses minimum amount of total memory. Note that this is an off-line problem. In the on-line version, buffers have to be allocated to requests as they arrive and without knowledge of future requests. The off-line version is also useful because designers of protocol stacks would like to estimate the size of the total memory block needed by extracting large traces of memory requests and running off-line algorithms on these traces. If we restrict our attention to memory allocation requests from *straight-line programs*, (i.e., programs without loops or branching statements), each memory allocation request is made for a specific duration of time and thus these requests can be viewed as intervals on a real line. Requests which are live at the same time have to be satisfied by different buffers. In this restriction to straight-line programs, the underlying graph is an interval graph.

The max-coloring problem for interval graphs, is related to the *dynamic storage allocation problem* (DSA) described in Section 1.1, or the interval coloring of interval graphs (See Section 1.3.3). An instance of this problem consists of an interval graph $G = (V, E)$ and a weight function $w : V \rightarrow \mathbf{N}$. A feasible solution to this problem in an assignment of an interval $I(v)$ to each vertex v such that $|I(v)| = w(v)$ and

$I(u) \cap I(v) = \emptyset$ if u and v are adjacent vertices. The goal is to minimize $|\cup_{v \in V} I(v)|$.

The similarity between the interval coloring problem and the max-coloring problem can be best understood by casting these problems into a geometric setting as rectangle packing problems. Start with an interval representation $\{I_v \mid v \in V\}$ of the given interval graph $G = (V, E)$. Interpret each weight $w(v)$ as the height of interval I_v. In other words, the instance of the problem consists of axis-parallel rectangles $\{R_v \mid v \in V\}$, such that the projection of R_v on the x-axis is I_v and the height of R_v is $w(v)$. Each rectangle can be slid up or down but not sideways; all rectangles have to occupy the positive quadrant; and the regions of the plane they occupy have to be pairwise disjoint. Given these constraints, the interval coloring problem is equivalent to the problem of packing these rectangles so as to minimize the y-coordinate of the highest point contained in any rectangle. The max-coloring problem seeks a packing of the rectangles into disjoint horizontal strips $S_i = \{(x, y) \mid x \geq 0, \ell_i \leq y \leq u_i\}$, denoted by $[\ell_i, u_i]$. The constraints are that every rectangle is completely contained in some strip and for any two rectangles R_u and R_v in a strip, their projections on the x-axis I_u and I_v are disjoint. Given these constraints, the max-coloring problem seeks a packing of the rectangles into strips so that the total height $\sum(u_i - \ell_i)$ of the strips is minimized. Figure 1.8 shows two rectangle packings of a set of rectangles; the packing on the left is optimal for the interval coloring problem and the packing on the right is optimal for the max-coloring problem.

Stated as rectangle packing problems, interval coloring and max-coloring seem similar. However, as we show in Chapter 3, the weights of optimal solutions for the

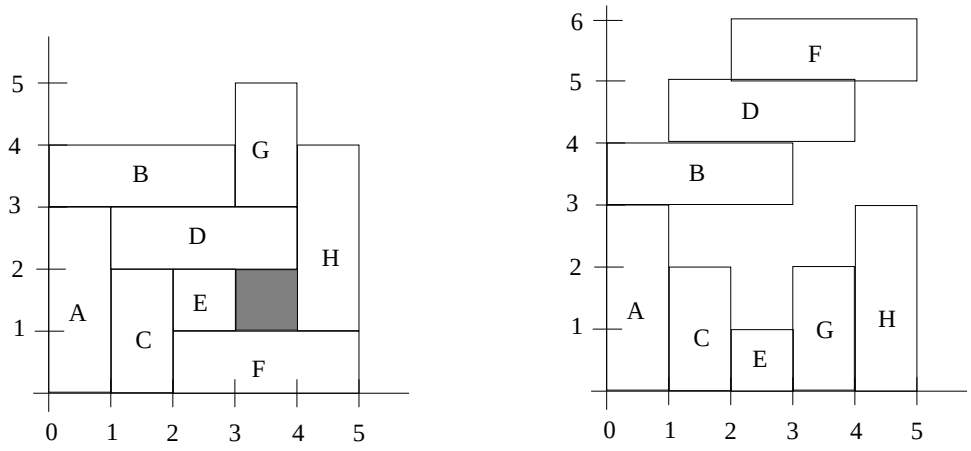

Figure 1.8: On the left is a rectangle packing corresponding to an optimal interval coloring, with weight 5. On the right is a rectangle packing corresponding to an optimal max-coloring. In the packing on the right the rectangles are packed into 4 strips: $S_1 = (0, 3)$, $S_2 = (3, 4)$, $S_3 = (4, 5)$, and $S_4 = (5, 6)$, for a total weight of 6.

two problems on the same input can be quite different. For DSA, Buchsbaum, et. al. [12],et. al. show that the optimum value is at most a factor $(2 + \epsilon)$ times the weight of the maximum weight clique, while for max-coloring it is easy to construct examples (See Figure 3.1) where the optimum cost is a factor $\Omega(\log n)$ away from the weight of the maximum weight clique.

Another motivation for studying the max-coloring problem comes from scheduling theory; namely the problem of scheduling conflicting jobs in batches. Suppose we are given a set $\mathcal{J} = \{J_1, \cdots, J_n\}$ of jobs, their processing times p_j, and a conflict graph $G = (\mathcal{J}, E)$. We want to schedule these jobs on an arbitrary number of machines. For example, the jobs may be processes in a distributed operating system. The conflict relation between the jobs is given as a graph $G = (\mathcal{J}, E)$, where the

edges represent a pair-wise conflict. For example, processes in a distributed system may require access to the same files or devices and hence cannot be run together.

Our task is to compute a batch schedule on this set of jobs to minimize the makespan, or the completion time of all the jobs in the schedule. It follows that each subset of jobs in a batch must be conflict-free, and hence must be an independent set. Thus, a feasible schedule is a coloring of the conflict graph. The time taken to process a batch is the time taken by a job with the longest processing time. The problem of computing a batch schedule to minimize makespan, is precisely the max-coloring problem on the conflict graph of the jobs.

Motivated by the buffer allocation problem, we began our study of the max-coloring problem on interval graphs and showed that the problem is NP-hard. We also developed a 2-approximation algorithm for the max-coloring problem on interval graphs. Prior to our work, the problem was considered by Guan and Zhu [40] who were motivated by a problem of scheduling communication slots for a metropolitan area network. In their paper, they studied the max-coloring problem for trees, and present results on the number of colors required to achieve a minimum max-coloring cost. The max-coloring problem has also been studied by Govindarajan and Rengarajan [36] who, like us, were motivated by the problem of allocating a small buffer, but in the context of digital signal processing applications. We then studied the max-coloring problem on other graph classes. The problem turns out to be surprisingly difficult to solve even on trees and bipartite graphs. For trees, we show a PTAS and a sub-exponential time algorithm, while for bipartite graphs, we show that the max-coloring

problem is inapproximable beyond a threshold of $8/7-\epsilon$ for any $\epsilon > 0$ and also present an approximation algorithm with an approximation ratio of $8/7$. The results for trees and bipartite graphs were also independently obtained by Escoffier, et al [26, 57] For more general graph classes, we show that if for a hereditary class of graphs, the classical coloring problem has a c-approximation algorithm, then the max-coloring problem on this class of graphs admits an $e \cdot c$-approximation algorithm, where e is the base of the natural logarithm. This implies an e-approximation algorithm for the max-coloring problem on perfect graphs. Our results are presented in Table 1.3.2, where we show the best approximation ratios achieved by our algorithms, and the best known lower bounds.

Our Results for Max-Coloring		
Graph	Lower Bound.	Upper Bound
Interval Graph	NP-Hard	2
Trees	–	$(1+\epsilon)$, for any $\epsilon > 0$
Bipartite Graphs	$\frac{8}{7} - \epsilon$, $\forall \epsilon > 0$	$\frac{8}{7}$
Perfect Graphs	$\frac{8}{7} - \epsilon$, $\forall \epsilon > 0$	e
Hereditary Graphs	$\frac{8}{7} - \epsilon$, $\forall \epsilon > 0$	$e \cdot c$

Table 1.1: Summary of results for the Max-Coloring problem

1.3.3 Interval Coloring

In the interval coloring problem, we are given a graph $G = (V, E)$ and a weight function $w : V \rightarrow \mathbf{N}$. The problem is to assign to each vertex an interval $I(v)$ on

the real line, such that $|I(v)| = w(v)$ and such that adjacent vertices receive disjoint intervals. i.e, $|I(u) \cap I(v)| = \emptyset$, for each $\{u, v\} \in E$. The objective is to minimize the span $|\cup_{v \in V} I(v)|$ of the intervals.

The interval coloring problem, restricted to interval graphs is the dynamic storage allocation problem defined in Section 1.1, or as the *shipbuilding problem* [35]. The interval coloring problem also models the scheduling of conflicting jobs that need to be scheduled non-preemptively to minimize makespan.

1.3.4 Experimental Evaluation

Since the max-coloring and interval coloring problems are motivated by real-life applications, we also studied the max-coloring and interval coloring problems empirically. We evaluated the performance of our approximation algorithms and heuristics for both the max-coloring and interval coloring problems on chordal graphs. We focus on the class of chordal graphs for several reasons. One of our motivations is to determine if the constant-factor algorithms for interval coloring interval graphs can be extended to chordal graphs. Another motivation for considering chordal graphs is that the way certain kinds of compilers such as algebraic compilers process source code, the interference graph of source objects ends up being a chordal graph [65]. We generated two kinds of chordal graphs with widely different structure and evaluated the performance of two approximation algorithms, *WtPartition* and *Geomfit*, described in Chapter 3, as well as two simple heuristics, namely *first-fit* and *best-fit*. We ran two kinds of experiments. In one, we assigned weights for the random graphs generated so that the optimum max-coloring cost can be easily computed. In an-

other set of experiments, we assign random weights to the vertices and compared the performance of our algorithms against the weight of the maximum weight clique.

We then study the interval coloring problem on the chordal graphs generated. We use four heuristics similar to the ones for max-coloring. However, the algorithms for interval coloring do not enjoy a constant factor guarantee. We show in Chapter 4 that for the max-coloring problem, the algorithm $Geomfit$, described in Chapter 3 performs better than all algorithms. For the interval coloring problem, again the algorithm similar to $Geomfit$ has the best performance guarantee consistently.

CHAPTER 2
FIRST-FIT COLORING

2.1 Introduction

This chapter[1] deals with algorithms and analysis for two *online coloring problems*. Recall from Chapter 1 that an online algorithm for a problem must process the data elements when they arrive without any knowledge of the future arrivals of data elements. The performance of an online algorithm is measured by the *competitive ratio*, defined in Chapter 1.

In *online coloring*, the vertices of a graph are presented one by one to an algorithm. When a vertex is presented, the adjacency to the vertices previously presented is also revealed to the algorithm. The algorithm must assign a color to the vertex before the next vertex is presented. Once a vertex is assigned a color, it cannot change. It is easy to see that any online graph coloring algorithm can be forced to use more colors than the chromatic number even when the off-line graph coloring problem can be solved optimally in polynomial time.

2.1.1 First-Fit Coloring of Interval Graphs

A natural and simple heuristic for online coloring interval graphs is First-fit, which simply assigns the smallest available color to each vertex. The first-fit algorithm is both simple to implement and provides excellent performance in practice. However, as the example in Figure 2.1 shows, the first-fit algorithm can be forced to use more

---

[1]The results of this chapter are joint work with Sriram Pemmaraju and Kasturi Varadarajan, and appeared in [63] and [61]

colors than the chromatic number of a graph. Suppose the vertices are presented to the algorithm in the order of the number inscribed in each rectangle. Then, first-fit places vertices 1 and 2 in the first color class. Now when vertex 3 arrives, it must be placed in the second color class. However, when vertex 4 arrives, since it is adjacent to a vertex in both the first and second color classes, First-fit is forced to assign it a third color. It is clear that the chromatic number of the graph is 2.

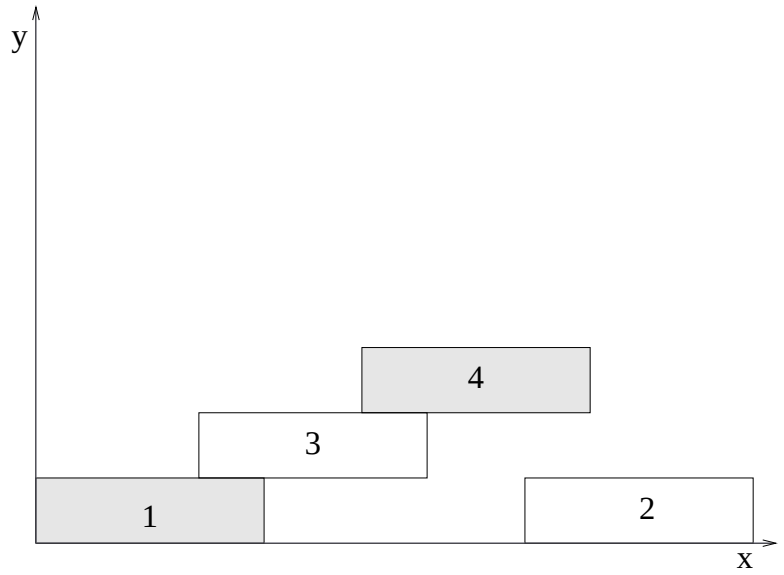

Figure 2.1: An instance of a problem where fist-fit is forced to use more colors than the chromatic number

A natural question that arises is bounds on the lower and upper bounds on the first-fit coloring of interval graphs. However, this problem has turned out to be notoriously hard. The problem first arose in connection to the *dynamic storage allocation problem* (DSA) described in Chapter 1. In [14], Chrobak and Slusarek made

the connection between dynamic storage allocation and first-fit coloring in a quest for finding approximation algorithms for DSA. They presented an algorithm, called BUDDY DECREASING SIZE (BDS), which uses the first-fit coloring of interval graphs as a sub-routine. BDS has the property that if first-fit is c-competitive, then BDS is a $2c$-approximation algorithm. Woodall [70] in 1974, presented the first upper bounds on the number of colors required by first-fit. He showed that for any interval graph G with chromatic number $\chi(G)$, first-fit uses at most $\chi(G)\log\chi(G)$ colors. Kierstead was the first to show that the number of colors used by first-fit is linearly bounded by the chromatic number. In [50], he showed that first-fit uses at most $40 \cdot \chi(G)$ colors. In a later paper, Kierstead and Qin [52] improved this bound and showed that first-fit is at most 25.72-competitive. In [63] we presented a new analysis that improved the bounds to 20/9. Kierstead, Trotter and Brightwell [11], and independently Narayanswamy and Babu [58] presented a slight modification of this analysis to show that first fit is infact 8-competitive. In this chapter we present this improved analysis of Narayanswamy and Babu. For the special case of *proper interval graphs*, i.e., interval graphs where no interval is properly contained in any other interval, Chrobak and Slusarek [14] showed that first-fit is $2 \cdot \chi(G)$-competitive, and this analysis is tight.

The best known lower bounds for the first-fit coloring problem were presented by Chrobak and Slusarek [14], who showed that first-fit uses at least $4.4\chi(G)$ colors in the worst case. In recent work, Kierstead and Trotter [49] have improved this for interval graphs with *large enough* chromatic number. They show that first-fit uses at

least $(5-\epsilon)\chi(G)$ colors, where $\epsilon \to 0$ as $\chi(G) \to \infty$. Further, they also show that all *tree-like* interval graphs used in the analysis cannot give a lower bound better than 5.

The first-fit heuristic has also been studied for more general graphs. Irani [46] proved that for d-inductive graphs, first-fit uses at most d colors. The number of colors used by the first-fit algorithm has also been studied as a graph parameter, called the *Graph Grundy Number*, denoted $\chi_{FF}(G)$. This is defined as the *maximum* number of colors used by first-fit over all permutations of the vertices of a graph. The graph grundy number was first studied by Grundy in [39], where he used them in the study of *kernels of directed graphs*. Christen and Selkov [13] were the first to study the graph grundy number as a graph parameter. In [4], Balogh, et. al. study the first-fit chromatic number of planar graphs and Erdös Rènyi random graphs with $p = 1/2$.

For interval graphs, Chrobak and Slusarek [14] proved that there is no algorithm that can give a competitive ratio better than 3. Kierstead and Trotter [53] presented an algorithm that is a slight modification of first-fit that achieves this bound.

2.1.2 Online Coloring of Intervals with Bandwidth

In an instance of the problem of *online coloring of intervals with bandwidth*, intervals are presented one at a time, with each interval i being associated with a bandwidth $b(i) \in (0, 1]$. Each interval must be assigned a color immediately after it has been presented (and before the next interval is presented) and a color assigned to an interval cannot be changed later. An algorithm for this problem assigns colors to

intervals in the manner described above, so that for any color c and any real r, the sum of bandwidths of intervals containing r and colored c is at most 1. The goal is to minimize the number of colors used. This problem was introduced by Adamy and Erlebach in [1]. Their study was motivated by problems in resource allocation such as channel allocation in an all optical WDM (wavelength-division-multiplexing) network, minimizing total duration in a call scheduling problem, and minimizing the number of machines used in a job scheduling problem in which each bandwidth represents the fraction of a machine that can be used by a job.

In [22], Epstein and Levy showed a lower bound of 3.2609 for interval coloring with bandwidths, slightly improving the lower bound of 3 that follows from online coloring interval graphs. They studied resource augmented versions of the interval coloring problem with and without bandwidths. Specifically, they presented an online algorithm with competitive ratio 3 when each color class has a capacity of 2, instead of 1. They also studied the problem of online coloring with bandwidths for unit interval graphs, where they presented an algorithm with competitive ratio 7/2 and give a lower bound of 2. In [23], Epstein, et. al. continued their study on online coloring with bandwidths, where they presented an improved lower bound of 24/7 and studied several variants of online coloring with bandwidths. In [20], Epstein, et. al. studied the variant where each color class has variable capacity. When the algorithm opens a color class, it chooses a capacity and cannot change it once chosen. They studied the bounded model where each capacity is in the range $(0, 1]$, and the unbounded case, where the algorithm may use any capacity for the color class. They

gave upper and lower bounds of 14 and 4.59 for the bounded model, and matching upper and lower bounds of 4 for the unbounded model. They also studied the off-line problem and proved that the bounded model is polynomial-time solvable, while the unbounded model is NP-complete. They presented a 3.6-approximation algorithm for the bounded case.

An instance of this problem is shown in Figure 2.2. For any real r, define density$(r) = \sum_{i:r\in i} b(i)$. Clearly, $\lceil \max_r \text{density}(r) \rceil$ is a lower bound on the number of colors needed. In the instance shown in Figure 2.2, $\max_r \text{density}(r) = 1\frac{3}{8}$ and therefore at least 2 colors are needed for this instance. It is easy to check that $\{A, E, D\}, \{C, B\}$ is a feasible coloring for this instance and is therefore optimal.

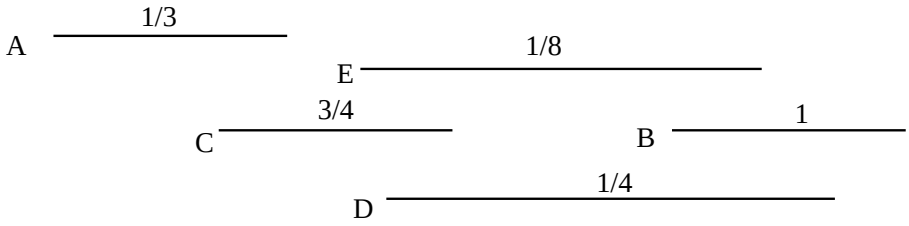

Figure 2.2: An instance of the problem of online interval coloring with bandwidths. The numbers shown above the intervals correspond to their bandwidths.

Note that if all the bandwidths are 1, then we have the well-studied problem of the online coloring of interval graphs.

As mentioned by Adamy and Erlebach [1] , the problem of online coloring

of intervals with bandwidth is a simultaneous generalization of online coloring of intervals and online bin packing. The First-Fit algorithm for online coloring of intervals without bandwidths naturally extends to the current problem, with bandwidth constraints. Given a coloring of a subset $S \subseteq \mathcal{I}$ of the intervals, we define

$$\text{density}_c(r) = \sum_{x \in S : r \in x, color(x) = c} b(x)$$

for each color c and real r. Then the First-Fit algorithm for the online interval coloring problem with bandwidths uses the following first-fit rule: for each interval x that arrives, color x with the smallest color c such that $b(x) + \text{density}_c(r) \leq 1$ for all $r \in x$. If intervals in the instance shown in Figure 2.2 arrive in the order A, B, C, D, E, then the First-Fit algorithm uses 3 colors because it colors A and B with color 1, C and D with color 2, and E with color 3.

The performance of First-Fit algorithm for the problem of online coloring of intervals with bandwidths differs in a fundamental way from the First-Fit algorithm for the usual problem of online coloring of intervals . We have noted the existence of an upper bound $8 \cdot \chi(\mathcal{I})$ on the number of colors used by First-Fit for the usual problem. However, as pointed out in [1], for the problem with bandwidths the First-Fit algorithm can be forced to do arbitrarily poorly. We present a bad example in Figure 2.3. Pick an integer $k > 1$ and a real $\varepsilon \in (0, 1)$. Corresponding to each pair (k, ε) of values we construct an instance that contains intervals with bandwidth ε and intervals with bandwidth 1. Figure 2.3 shows an instance with $k = 5$. The long (and

thin) intervals all have bandwidth $\varepsilon > 0$ and the short (and thick) intervals all have bandwidth 1. A k-coloring of this instance produced by the First-Fit algorithm is shown, where the intervals in color 1 are provided first to the algorithm, followed by all the intervals in color 2, and so on. Notice that for each interval of bandwidth ε, there is an interval with bandwidth 1 in each of the previous color classes. Similarly, for each interval of bandwidth 1, there is an interval of bandwidth ε in each of the previous color classes. This justifies the first-fit coloring shown in the figure. Also notice that lower bound on the optimal number of colors needed is $\max_r \text{density}(r) = 1+(k-1)\cdot\varepsilon$. Choosing $\varepsilon = 1/(k-1)$, we get a lower bound of 2. A 2-coloring of this instance can be obtained by using color 1 for all intervals with bandwidth 1 and for the interval with bandwidth ε with the rightmost right endpoint. It is easy to see that the remaining intervals, all of which have bandwidths ε, can be colored with color 2.

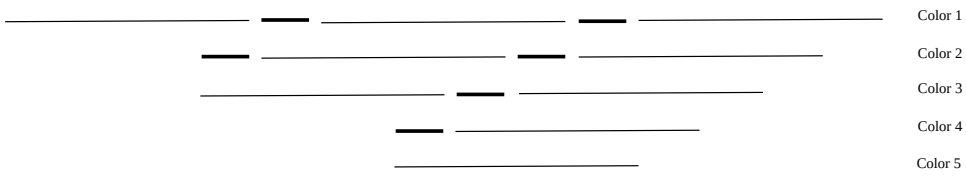

Figure 2.3: An instance of intervals with bandwidth showing that First-Fit can perform arbitrarily badly.

In our main result for online coloring with bandwidths, we show that such bad examples can exist only in the "limiting case" where bandwidths are arbitrarily close to 1. As long as the maximum bandwidth of every interval in the input is bounded

above by a constant strictly smaller than 1, the First-Fit algorithm uses a constant times the optimal number of colors.

We consider the problem of online coloring of interval graphs with bandwidths. In Section 2.3.1 we show that the Adamy-Erlebach algorithm is 30-competitive. This requires an extension of the analysis of First-fit in Section 2.2 to the case of intervals with bandwidths. The extended analysis is presented in Section 2.3.2. We conclude with final remarks and open questions in Section 2.4.

2.2 Column Construction

For interval graphs, it is convenient to work with a representation of G in terms of a set of intervals $\mathcal{S}$. We assume without loss of generality that each interval in the set $\mathcal{S}$ of input intervals is of the form $[i, j]$, where $0 \leq i < j \leq N$ are integers, and N is a sufficiently large positive integer.

We denote by $\mathcal{E}$ the set of *elementary* intervals $\{[i-1, i] | 1 \leq i \leq N\}$. We will refer to the ordering of the intervals in $\mathcal{E}$ according to increasing order of the left endpoints as their *natural* ordering. Thus each input interval is a union of consecutive elementary intervals; two input intervals intersect if they both contain a common elementary interval. The *leftmost* (resp. *rightmost*) elementary interval of an interval $I \in \mathcal{S}$ is the first (resp. last) elementary interval in the natural ordering that is contained in I.

We briefly review the first-fit algorithm for coloring intervals in $\mathcal{S}$. The intervals are presented to the algorithm in some arbitrary order. The first interval is assigned to color class 1. Each subsequent interval is then assigned to the lowest

color class that will accept it. That is, assume that the algorithm has seen intervals $\mathcal{S}' \subseteq \mathcal{S}$ and assigned them to color classes $1, \ldots, j$. When a new interval I arrives, the first-fit algorithm finds the smallest value of i, for $1 \leq i \leq j+1$, such that I does not intersect any interval in $\mathcal{S}'$ that was assigned to the i'th color class. Such an i must exist because no interval in $\mathcal{S}'$ is assigned to the $(j+1)$'th color class. The algorithm then assigns I to the i'th color class.

Clearly, the first-fit algorithm yields a proper coloring of the intervals in $\mathcal{S}$ (more precisely, of the interval graph corresponding to $\mathcal{S}$). Suppose that first-fit uses colors $1, \ldots, m$. We will now argue that there is an elementary interval that is contained in at least $m/8$ input intervals. Note that this corresponds to a clique of size at least $m/8$ in the corresponding interval graph, which means $m/8$ is a lower bound on the size of any proper coloring of the interval graph. A useful way to visualize the coloring generated by first-fit is to imagine an interval $[l, r]$ that is assigned to color class k as a rectangle $\{(x, y) | l \leq x \leq r, k-1 \leq y \leq k\}$ of height one.

2.2.1 Column Construction

The key property of the first-fit coloring that will be needed in the proof is that if an interval $I \in \mathcal{S}$ is assigned to color class k, then for each $1 \leq i \leq k-1$ there is an interval I' assigned to color class i such that I intersects I'. The proof is based on a construction of a set of "columns" corresponding to the first-fit coloring. A column corresponds to a unique elementary interval e, and with some abuse of notation is referred to as column e. There may be elementary intervals that have no corresponding columns. A column has a positive integral *height* associated with

it. If a column has height t, we say that it is active at heights $1, \ldots, t$ and inactive at heights $t+1, \ldots, \infty$. A column of height t is labeled, at each height i between 1 and t, with one symbol which is either "R", "$", or "F". A column e of height t is labeled "R" at some height $1 \leq i \leq t$ if and only if some interval $I \in \mathcal{S}$ that is assigned to the i'th color class contains the elementary interval e. However, it could be the case that an elementary interval e is contained in an interval that is assigned to the i'th color class and there is either no column corresponding to e or the column e is inactive at height i. It is useful to visualize a column e of height t as a rectangle $\{(x,y)|l(e) \leq x \leq r(e), 1 \leq y \leq t\}$ of width 1, where $l(e)$ and $r(e)$ are the left and right endpoints of elementary interval e; the box $\{(x,y)|l(e) \leq x \leq r(e), i-1 \leq y \leq i\}$ contains the label of the column at height i, for $1 \leq i \leq t$. See Figure 2.4 for an example. It is worth pointing out that the goal of the column construction procedure is to find a column with at least $m/8$ "R" labels in it.

The column construction procedure works by starting with a set of columns that are active at height 1 and, for $i \geq 2$, choosing a subset of the active columns at height $i-1$ to be the active columns at height i. If the set of active columns at height i is empty, the procedure stops. For any set of columns, there is a natural ordering that is induced by the natural ordering of the corresponding elementary intervals. Let C_i denote the set of columns active at height i. For any $e \in C_i$, the left (resp. right) neighbor of e in C_i is the column in C_i immediately preceding (resp. succeeding) e in the natural ordering; if no such column exists, the left (resp. right) neighbor is undefined.

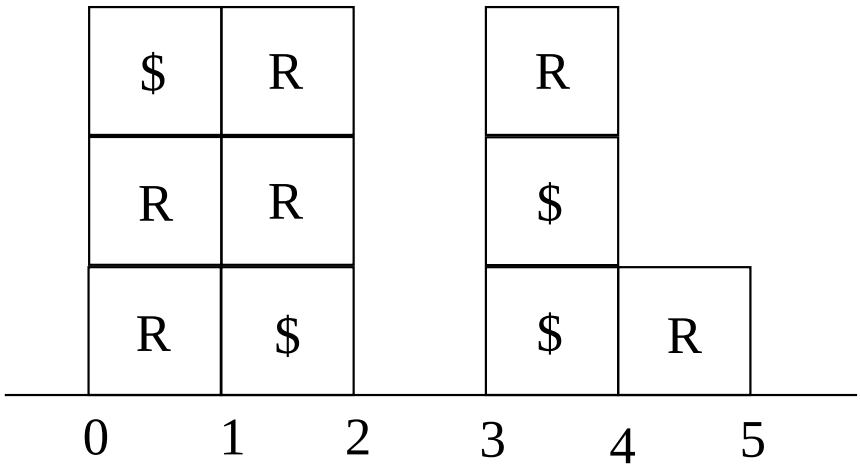

Figure 2.4: A possible result of the column construction procedure. The columns correspond to elementary intervals $[0,1],[1,2],[3,4]$, and $[4,5]$. Column $[1,2]$ has has height 3 and has labels \$ "R", and "R" associated with heights 1, 2, and 3 respectively. Note that $C_1 = \{[0,1],[1,2],[3,4],[4,5]\}$, $C_2 = \{[0,1],[1,2],[3,4]\}$, and $C_3 = \{[0,1],[1,2],[3,4]\}$.

We are now ready to describe the column construction procedure. If an elementary interval e is contained in an interval assigned to the first color class, then e is added to C_1 and is assigned the label "R" at height 1. These are the only columns in C_1. For $i \geq 2$, the following rules specify which columns from C_{i-1} are picked in C_i.

1. For each $e \in C_{i-1}$, if e is contained in some interval assigned to color class i, then e is added to C_i and is assigned a label "R".

2. For each remaining $e \in C_{i-1}$, if e is the left neighbor in C_{i-1} of some column e' added to C_i by Rule 1, then e is added to C_i with a label "\$" at height i. For accounting purposes, we say that e' contributes a \$ to e at height i. Note that e is the left-neighbor of e' in C_i. For each remaining $e \in C_{i-1}$, if e is the right neighbor in C_{i-1} of some column e' added to C_i by Rule 1, then e is added to C_i with a label "\$" at height i. For accounting purposes, we say that e' contributes a \$ to e at height i. Note that e is the right neighbor of e' in C_i.

3. For each remaining $e \in C_{i-1}$, let e' be the left-neighbor of e in C_{i-1}. If undefined, we proceed to inspect the right neighbor of e in C_{i-1}. Suppose that e' is the left neighbor of e in $C_j, \ldots, C_{i-1}$, and is not the left neighbor of e in $C_1, \ldots, C_{j-1}$; note that such a j does exist. If the number of "R" labels of e at heights $j, \ldots, i-1$ is greater than $(i-j)/4$, then e is added to C_i with a label "F" at height i. If not, let e'' be the right neighbor of e in C_{i-1}. If undefined, e is not added to C_i. Otherwise, suppose that e'' is the right neighbor of e in

$C_k, \ldots, C_{i-1}$, and is not the left neighbor of e in $C_1, \ldots, C_{k-1}$. If the number of "R" labels of e at heights $k, \ldots, i-1$ is greater than $(i-k)/4$, then e is added to C_i with a label "F" at height i. If not, e is not added to C_i. See figure 2.5 for an example.

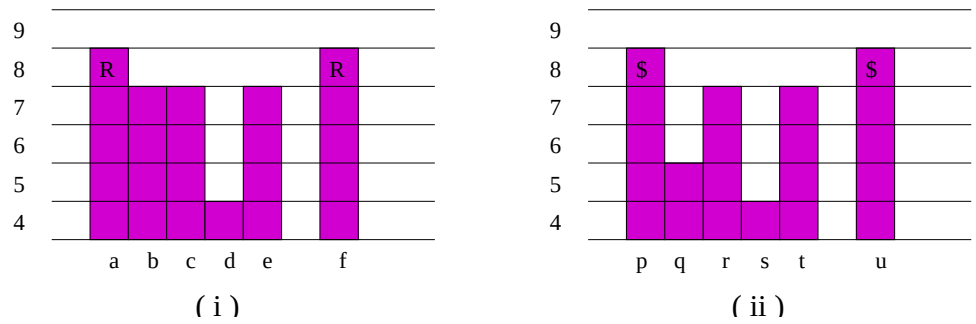

Figure 2.5: (i) A snapshot in the construction of C_8 after Rule 1 has been applied. Columns b and e will get added to C_8 with \$ labels at height 8 due to Rule 2. (ii) Snapshot after Rule 2 has been applied. Column p is the left neighbor of r in C_6, C_7 and t is the right neighbor of r in C_5, C_6, C_7. r is added to C_8 (with a label F at height 8) iff the number of its R labels at heights $6, 7$ is greater than $2 \cdot 1/4$ or the number of its R labels at heights $5, 6, 7$ is greater than $3 \cdot 1/4$.

The construction proceeds until round m', after which *no* column is active. i.e, $C_{m'+1} = \emptyset$. Note that if $m' > m$, then at heights $m+1, \cdots, m'$, only Rule 3 applies and the only symbols added are F's. The construction procedure maintains the following important invariant. Abusing notation, we say that interval I intersects a column e if it contains elementary interval e.

Lemma 2.2.1 *Let $1 \leq i < j \leq m$, for any interval $I \in \mathcal{S}$ that is assigned to color class j, there is a column $e \in C_i$ such that I intersects e.*

Proof: By induction on i. The lemma holds for $i = 1$ because of the key property of the first-fit coloring. For the inductive step, assume $i \geq 2$ and the lemma holds for $i - 1$. Let I be an interval that is assigned to color class $j > i$. By the key property, there is an $I' \in \mathcal{S}$ that is assigned to the i'th color class such that I' intersects I. By the induction hypothesis, I' (resp. I) intersects a non-empty set C' (resp. C) of consecutive columns (natural ordering) in C_{i-1}. If $C \cap C' \neq \emptyset$, we are done since all columns in C' are added to C_i by Rule 1. If some column $e \in C_{i-1}$ lies between the columns in C' and the columns in C, then I and I' cannot intersect because the elementary interval e lies between I and I'. So it must be that some $f \in C$ is either the left neighbor or right neighbor in C_{i-1} of some column in C'. The column f is added to C_i by Rule 2 if it is not already added by Rule 1, completing the proof. □

The invariant implies that $C_1, \ldots, C_m$ are non-empty, and thus $m' \geq m$. For a column e with height at least j and $1 \leq i \leq j$, let $\rho_e(i,j)$, $\delta_e(i,j)$, and $\phi_e(i,j)$ denote, respectively, the number of R, \$, and F labels of e between heights i and j (inclusive). Let $\rho_e(j) = \rho_e(1,j)$, $\delta_e(j) = \delta_e(1,j)$, and $\phi_e(j) = \phi_e(1,j)$. Define $\rho_e(0) = \delta_e(0) = \phi_e(0) = 0$.

Lemma 2.2.2 *For any column $e \in C_i$, and any integer $1 \leq i \leq m'$, $\rho_e(i) \geq \frac{1}{4}(\rho_e(i) + \phi_e(i))$.*

Proof: The proof is by induction on i. The base cases $i = 0, 1$ are easily verified. Suppose $i \geq 2$ and the lemma is true for all $0 \leq i' < i$. If e is not labeled with F at height i, then the induction step goes through easily. So let us assume

that e is labeled F at height i. So e was added to C_i by Rule 3 and so there exists a $1 \leq j \leq i-1$ such that $\rho_e(j, i-1) > (i-j)/4$. This implies that $\rho_e(j, i) \geq (i-j+1)/4$. From this and the induction hypothesis, it follows that

$$\begin{aligned}
\rho_e(i) &= \rho_e(j,i) + \rho_e(j-1) \\
&\geq \frac{1}{4}(i-j+1) + \frac{1}{4}(\rho_e(j-1) + \phi_e(j-1)) \\
&\geq \frac{1}{4}(\rho_e(j,i) + \phi_e(j,i)) + \frac{1}{4}(\rho_e(j-1) + \phi_e(j-1)) \\
&= \frac{1}{4}(\rho_e(i) + \phi_e(i))
\end{aligned}$$

□

We are now ready to obtain our main result.

Theorem 2.2.3 *Let m denote the number of colors used by first-fit to color the set of intervals $\mathcal{S}$. There is a clique of size at least $m/8$ in the corresponding interval graph.*

Proof: We will show that there is a column $e \in C_{m'}$ such that $\rho_e(m') \geq m/8$. Let e be any column in $C_{m'}$, and assume that e has both a left and right neighbor in $C_{m'}$. The case when either neighbor is absent is infact easier. Let $f_1, \cdots, f_a$ be the left neighbors of e, where f_i is the left neighbor of e in $C_{t_{i-1}+1}, \cdots, C_{t_i}$, $i = 1, \cdots, a$, where $t_0 = 0 < t_1 < \cdots < t_a = m'$. Similarly, let $g_1, \cdots, g_b$ be the right neighbors of e, with g_i being a right neighbor of e at $C_{n_{i-1}+1}, \cdots, C_{n_i}$ for $i = 1, \cdots, b$. $n_0 = 0 <$

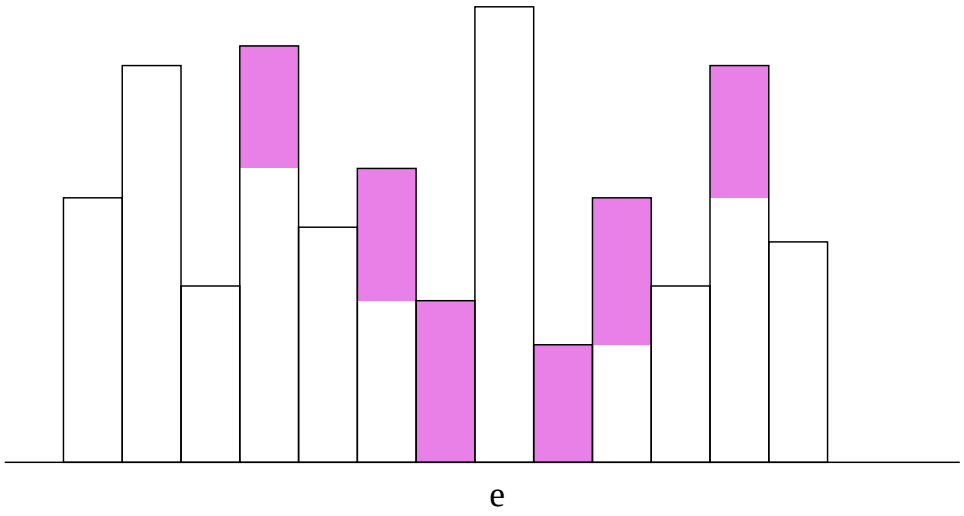

Figure 2.6: The columns at the end of the construction procedure. The number of \$ labels in column e is bounded by the number of "R" labels in all the shaded rectangles.

$n_1 < \cdots < n_b = m'$. Now we claim that

$$\delta_e(m') \leq \sum_1^a \rho_{f_i}(t_{i-1}+1, t_i) + \sum_1^b \rho_{g_i}(n_{i-1}+1, n_i).$$

The claim follows from the observation that $C_e(i)$ is a "\$" if and only if it is supported by either of its neighbors. i.e., a "\$" is added to e at level i by Rule 2. See figure 2.6. Since each f_i and g_i become inactive beyond a height of t_i and n_i respectively, by Rule 3 we have

$$\rho_{f_i}(t_{i-1}+1, t_i) \leq \frac{1}{4}(t_i - t_{i-1}).$$

By an identical argument, we have that

$$\rho_{g_i}(n_{i-1}+1, n_i) \leq \frac{1}{4}(n_i - n_{i-1}).$$

Hence,

$$\delta_e(m') \leq \sum_1^a \frac{1}{4}(t_i - t_{i-1}) + \sum_1^b \frac{1}{4}(n_i - n_{i-1}) \leq \frac{1}{4}(t_a + n_b) = \frac{m'}{2}$$

Since

$$\rho_e(m') + \phi_e(m') + \delta_e(m') = m',$$

and

$$\rho_e(m') + \phi_e(m') = m' - \delta_e(m') \geq m' - \frac{m'}{2} = \frac{m'}{2},$$

we can apply Lemma 2.2.2 to get

$$4\rho_e(m') \geq \frac{m'}{2}$$

or

$$\rho_e(m') \geq \frac{m'}{8}$$

Since no "R" labels are added after level m, $\rho_e(m) = \rho_e(m')$. Hence,

$$\rho_e(m) = \rho_e(m') \geq \frac{m'}{8} \geq \frac{m}{8}$$

By a similar argument, we can show that $\rho_e(m) \geq m/8$ in the cases where e has no left-neighbor or right-neighbor. □

From Theorem 2.2.3 we directly obtain our main result.

Theorem 2.2.4 *For any interval graph G, the first-fit strategy for on-line coloring of G uses at most $8 \cdot \chi(G)$ colors.*

2.3 Interval Coloring with Bandwidths

In this section, we consider the problem of online coloring interval graphs with bandwidth. We start with a description of the algorithm due to Adamy and Erlebach in [1], and give an improved analysis. Narayanswamy [59] presented an algorithm that is 10-competitive for this problem, which is the best bound known.

For any instance $\mathcal{I}$ of the interval coloring problem with bandwidths, let $OPT(\mathcal{I})$ denote the fewest number of colors needed to color the intervals in $\mathcal{I}$ such that the bandwidth constraints are satisfied. Whenever $\mathcal{I}$ is obvious from the context, we simply use OPT. Adamy and Erlebach [1] present an online algorithm for solving the interval coloring problem with bandwidths that uses at most $195 \cdot OPT$ colors. The Adamy-Erlebach algorithm is a combination of the First-Fit algorithm and the Kierstead-Trotter algorithm. When an interval x arrives, if $b(x) \leq 1/2$, use the First-Fit rule to assign a color to x from a set C_1 of colors. Otherwise, if $b(x) > 1/2$, ignore the bandwidth of x and use the Kierstead-Trotter algorithm to assign a color to x from a set C_2 of colors. Here C_1 and C_2 are disjoint.

Adamy and Erlebach derive their approximation factor of 195 as follows. Let $\mathcal{I}_1 = \{x \in \mathcal{I} \mid b(x) \leq 1/2\}$ and $\mathcal{I}_2 = \{x \in I \mid b(x) > 1/2\}$. No two intersecting

intervals in $\mathcal{I}_2$ can be assigned the same color. Therefore, $\mathrm{O}PT(\mathcal{I}_2) = \chi(\mathcal{I}_2)$. As mentioned above, the Kierstead-Trotter algorithm will use at most $3 \cdot \chi(\mathcal{I}_2)$ colors and therefore the number of colors used, $|C_2| \leq 3 \cdot \mathrm{O}PT(\mathcal{I}_2) \leq 3 \cdot \mathrm{O}PT(\mathcal{I})$. Most of the Adamy-Erlebach paper is devoted to showing that the First-Fit algorithm uses at most $192 \cdot \mathrm{O}PT(\mathcal{I}_1)$ colors. The fact that $|C_1| \leq 192 \cdot \mathrm{O}PT(\mathcal{I})$ implies that $|C_1| + |C_2| \leq 195 \cdot \mathrm{O}PT(\mathcal{I})$.

2.3.1 Improved Analysis

In the following, we analyze the family of algorithms obtained by using a threshold $\alpha \in (0, 1)$ instead of $1/2$ in the Adamy-Erlebach algorithm. For each $\alpha \in (0, 1)$ we call the corresponding instance of the algorithm, the α-Adamy-Erlebach algorithm. (The α-Adamy-Erlebach algorithm colors an interval with the First-Fit algorithm if its bandwidth is at most α.) Our main result, proved in Section 2.3.2, is the following.

Theorem 2.3.1 *For any set $\mathcal{I}$ of intervals whose maximum bandwidth is $\alpha \in (0, 1)$, the First-Fit algorithm uses at most*

$$\frac{16}{(1-\alpha)} \cdot OPT(\mathcal{I}) + 1$$

number of colors.

Using this result we show the following.

Theorem 2.3.2 *The α-Adamy-Erlebach algorithm uses at most*

$$\left(\frac{16}{(1-\alpha)} + 3(\lceil\frac{1}{\alpha}\rceil - 1)\right) \cdot OPT(\mathcal{I}) + 1$$

colors.

Proof: Let $I_1 = \{x \in I \mid b(x) \leq \alpha\}$ and $I_2 = \{x \in I \mid b(x) > \alpha\}$. For intervals in I_2 the α-Adamy-Erlebach algorithm ignores bandwidths and produces a coloring by using the Kierstead-Trotter algorithm. Hence, $|C_2| \leq 3 \cdot \chi(I_2)$. Since $b(x) > \alpha$ for all $x \in \mathcal{I}_2$, in any coloring of $\mathcal{I}_2$ that satisfies the bandwidth constraint, a clique of size at most $\lceil 1/\alpha \rceil - 1$ can be colored using one color. Since $\mathcal{I}_2$ contains a clique of size $\chi(I_2)$, any coloring of I_2 that satisfies the bandwidth constraints, uses at least $\chi(I_2)/(\lceil 1/\alpha \rceil - 1)$ colors. Therefore, we have

$$\chi(I_2) \leq (\lceil 1/\alpha \rceil - 1) \cdot \mathrm{O}PT(I_2)$$

and therefore

$$|C_2| \leq 3 \cdot (\lceil 1/\alpha \rceil - 1) \cdot \mathrm{O}PT(I_2).$$

□

Substituting $\alpha = 1/2$ in the above expression we get an upper bound of $35 \cdot \mathrm{O}PT + 1$ on the number of colors used by the Adamy-Erlebach algorithm.

Corollary 2.3.3 *The Adamy-Erlebach algorithm uses at most $35 \cdot OPT + 1$ colors.*

The function $\left(\frac{16}{(1-\alpha)} + 3(\lceil \frac{1}{\alpha} \rceil - 1)\right)$ has a minimum at roughly $\alpha = 1/3$ in the range $(0, 1)$. Substituting $\alpha = 1/3$ in this expression we get an upper bound of $30 \cdot \mathrm{O}PT + 1$ on the number of colors used by the 1/3-Adamy-Erlebach algorithm.

Corollary 2.3.4 *The* 1/3*-Adamy-Erlebach algorithm uses at most* $30 \cdot OPT + 1$ *colors.*

2.3.2 Column Construction for Intervals with Bandwidth

In this section, we present a proof of Theorem 2.3.1 by showing that the column construction procedure of section 2.2 and the analysis can be extended for use in the case of intervals with bandwidths as well. This parallels the work Adamy and Erlebach, whose analysis of the First-Fit algorithm for coloring intervals with bandwidth at most 1/2 is a modification of the "centrality approach"used by Kierstead [50, 52].

Without loss of generality, we assume that there is a positive integer N such that each interval in $\mathcal{I}$ is $[x, y]$ for some integers x and y, $0 \leq x, y \leq N$. As before, we denote by $\mathcal{E}$ the set of *elementary* intervals $\{[x-1, x] \mid 1 \leq i \leq N\}$. Suppose the First-fit algorithm uses colors $1, 2, \ldots, m$ to color the intervals in $\mathcal{I}$. In the following we will show that there is an elementary interval e and a set of colors $C \subseteq \{1, 2, \ldots, m\}$ such that $|C| \geq (m-1)/8$ and for each color $c \in C$, $\text{density}_c(e) \geq (1-\alpha)/2$, where $\text{density}_c(e)$ is the sum of the bandwidths of intervals in $\mathcal{I}$ that contain e and are colored c. This implies that $\text{density}(e) = \sum_{c \in C} \text{density}_c(e) \geq (m-1)(1-\alpha)/16$. The key property of the first-fit coloring that we will repeatedly use in the analysis is:

First-Fit Property

If an interval x is colored $j \geq 1$, then for any color i, $1 \leq i < j$, there is

an elementary interval e contained in x such that $\text{density}_i(e) \geq (1-\alpha)$.

Now we describe the *column construction* procedure for intervals with bandwidth. The construction procedure is similar to the one described in Section 2.2, with a key difference. Rules 2 and 3 are identical to the corresponding rules in Section 2.2, while Rule 1 is modified to take into consideration the density of the intervals. The new Rule 1 is as follows.

For notational convenience, let $C_0 = \mathcal{E}$.

Rule 1 For each $e \in C_{i-1}$, if $\text{density}_i(e) \geq (1-\alpha)/2$, then e is added to C_i with a label "R" at height i.

The rest of the analysis consists of two parts. In the first part we show that there is at least one column that grows to a height of $(m-1)$. In the second part, we show that among columns of height $(m-1)$, there is one, say e, that contains at least $(m-1)/8$ "R" labels. Each "R" corresponds to a distinct color i such that $\text{density}_i(e) \geq (1-\alpha)/2$, implying that $\text{density}(e) \geq (m-1)(1-\alpha)/8$. This part of the proof is similar to the corresponding part in Section 2.2 and is only sketched in what follows.

2.3.3 Some Columns are Tall

We first establish the equivalent of Lemma 2.2.1 in the current setting.

Lemma 2.3.5 *For $0 \leq i < j \leq m$, and any interval $I \in \mathcal{S}$ that is assigned color j, there is a column $e \in C_i$ that I intersects.*

Proof: By induction on i. The base case $i = 0$ is straightforward. For the induction step, suppose $k \geq 1$, and the lemma holds for $i \leq k-1$. Consider an interval I that is colored $j > k$. By the First-Fit property, I contains an elementary interval e such that $\text{density}_k(e) \geq 1 - \alpha$. The induction hypothesis implies that C_{k-1} is non-empty. If $e \in C_{k-1}$, then $e \in C_k$ by Rule 1 and the induction step is complete. Suppose therefore that $e \notin C_{k-1}$, and that a and b are the left and right neighbors of e in C_{k-1}. (The case where one of the neighbors is absent is handled similarly.) By the induction hypothesis, each interval that is colored k and contains e must contain either a or b. Thus, either $\text{density}_k(a) \geq (1-\alpha)/2$, or $\text{density}_k(b) \geq (1-\alpha)/2$. Suppose the former holds. Then $a \in C_k$ by Rule 1 and $b \in C_k$ either by Rule 1 or Rule 2. Now the induction hypothesis (applied to $i = k-1$ and j) tells us that I contains either a or b. Thus the induction step is complete. □

Lemma 2.3.5 implies that $C_{m-1} \neq \emptyset$.

2.3.4 Some Tall Columns are Dense

In this section we sketch an argument that shows that among the columns of height $m-1$, there is a column that contains at least $(m-1)/8$ "R" labels. The argument is similar to the argument in Section 2.2. We can show that no column contains too many "$" and "F" labels. The upper bound on the number of "F" labels follows from the fact that if column e is labeled "F" at height i, there must be a j, $1 \leq j < i$, such that the number of "R" labels in column e at heights $j, j+1, \ldots, i$ is at least $(i-j)/4$. Hence, lemma 2.2.2 holds. From lemma 2.2.2 it follows that the number of "F" labels is at most 3 times the number of "R" labels in any column. The

rest of the analysis is identical to that in Section 2.2, and this proves Theorem 2.3.1.

2.4 Future Work

The biggest open problem is a tight analysis of the first-fit algorithm. In general it would be interesting to improve the lower bounds for the problem of online coloring with bandwidths. The current lower bounds use small variations of the Chrobak-Slusarek lower bound technique, but it may be interesting to look for other techniques.

Another question of interest is tight bounds for the online coloring problem for unit interval graphs. First-fit is $(2 \cdot \chi(G) - 1)$-competitive, but the best lower bound is only 3/2 by Epstein and Levy [22].

CHAPTER 3
APPROXIMATION ALGORITHMS FOR MAX-COLORING

3.1 Introduction

In this chapter[1], we present the first approximation algorithms and inapproximability results for the max-coloring problem on various classes of graphs. As described in the introduction, the max-coloring problem arises in two distinct applications. In the first application, which originally motivated our work, the problem models the allocation of buffers to variables in a straight-line program. The second motivation for studying this problem comes from scheduling theory; specifically, the problem of scheduling a set of conflicting jobs in batches. We start this section with a more detailed description of the motivation. In Section 3.2, we present previous work on the max-coloring problem. In Sections 3.3, 3.4, and 3.5, we present our results on the max-coloring problem. We conclude with some open questions and future work in Section 3.6.

3.1.1 Buffer Minimization

Programs that run with stringent memory or timing constraints use a dedicated memory manager that provides better performance than the general purpose memory management of the operating system. An example of such programs are protocol stacks like GPRS and 3G that run on mobile devices. These protocol stacks have stringent memory requirements as well as soft real-time constraints and a ded-

[1]This is joint work with Sriram Pemmaraju and Kasturi Varadarajan, and appeared in [63] and [62]

icated memory manager is a natural design choice. A dedicated memory manager for these stacks must have deterministic response times and use as little memory as possible. The most commonly used memory manager design for this purpose is the *segregated buffer pool*. This consists of a fixed set of buffers of various sizes with buffers of the same size linked together in a linked list. As each memory request arrives, it is satisfied by a buffer whose size is large enough.

The assignment of buffers to memory requests can be viewed as an assignment of colors to the requests – all requests that are assigned a buffer are colored identically. Thus the problem of determining whether a given segregated buffer pool suffices for a particular sequence of allocation requests can be formalized as follows. Let $G = (V, E)$ be a graph whose vertices are objects that need memory and whose edges connect pairs of objects that are alive at the same time. Let $w : V \rightarrow N$ be a weight function that assigns a natural number weight to each vertex in V. For any object v, $w(v)$ denotes the size of memory it needs. Suppose the segregated buffer pool contains k buffers with weights $w_1, w_2, \ldots, w_k$. The problem is to determine if there is a k-coloring of G into color classes $C_1, C_2, \ldots, C_k$ such that for each i, $1 \leq i \leq k$, $\max_{v \in C_i} w(v) \leq w_i$. The optimization version of this problem is the *max-coloring problem*. Given a graph $G = (V, E)$ and a weight function $w : V \rightarrow \mathbf{N}$ the problem is to find a proper vertex coloring $C_1, C_2, \ldots, C_k$ of G that minimizes $\sum_{i=1}^{k} \max_{v \in C_i}\{w(v)\}$. Note that the special case of this problem in which $w(v) = 1$ for all $v \in V$ is simply the problem of coloring a graph with fewest colors. Solving the max-coloring problem and selecting k buffers of sizes $\max_{v \in C_i}\{w(v)\}$, for $i = 1, 2, \ldots, k$, leads to a segregated buffer pool

that uses minimum amount of total memory. Note that this is an off-line problem. In the on-line version, buffers have to be allocated to requests as they arrive and without knowledge of future requests. The off-line version is also useful because designers of protocol stacks would like to estimate the size of the total memory block needed by extracting large traces of memory requests and running off-line algorithms on these traces.

If we restrict our attention to memory allocation requests from *straight-line programs*, (i.e., programs without loops or branching statements), each memory allocation request is made for a specific duration of time and thus these requests can be viewed as intervals on a real line. Requests which are live at the same time have to be satisfied by different buffers. In this restriction to straight-line programs, the underlying graph is an interval graph (See Chapter 1 for a definition of interval graphs).

3.1.2 Batch scheduling

In several scenarios involving the scheduling of a set of jobs on a set of machines, the jobs are partitioned into *batches* and jobs in a batch are processed simultaneously. Processing of the next batch of jobs starts only after all the jobs in the current batch complete. Batched scheduling is used, especially in manufacturing processes where a machine is capable of handling several types of jobs, but requires a different set up for each job. In this case, all jobs of the same type can be scheduled on the machine, before changing the set up for a different type of job. For example, materials in a factory may have to be processed in a kiln, where all materials that need to be processed at the same temperature can be processed simultaneously, before

starting a next batch with a different temperature. Batching thus leads to a higher utilization of the machine and hence more efficient schedules. Another application where batch scheduling is used is in scheduling jobs on a cluster machine, where jobs that need to communicate to accomplish a task are scheduled together. A recent paper by Potts and Kovalyov [64] surveys recent results in batch scheduling. Batch scheduling problems with conflicting jobs can be modeled as a max-coloring problem as follows. The input to a batch scheduling problem is a set of jobs $\mathcal{J} = \{J_1, \cdots, J_n\}$ and a processing time $w : \mathcal{J} \rightarrow \mathbf{N}$. We are also given a graph G, whose vertex set is the set of jobs and and edge between a pair of jobs implies the two jobs are not compatible and hence cannot be scheduled simultaneously. A proper vertex coloring of this graph corresponds to a schedule where each color class constitutes a batch. The processing time for a batch is the maximum processing time of any job in this batch, and the processing time for the entire set of jobs is the sum of processing times of the batches. Thus, if $S_1, \cdots, S_k$ are the independent sets in a coloring, the *weight* of an independent set $w(S_i) = \max_{v \in S_i} w(v)$, and the cost of this coloring is the sum of the weights of the independent sets. i.e., $\sum_{i=1}^{k} w(S_i)$. The problem of minimizing the makespan of this batched schedule is precisely the max-coloring problem.

3.2 Related Work

The max-coloring problem was first studied by Gaun and Zhu [40], who motivate the problem by an application that involves the transmission of wireless messages in a metropolitan network. They study the maximum number of colors needed by an optimal max-coloring for various special classes of graphs, and also show that

the max-coloring problem can be solved in polynomial time for graphs of bounded path-width. The max-coloring problem has also been studied by Govindarajan and Rengarajan [36] who, like us, are motivated by the problem of allocating a small buffer, but in the context of digital signal processing applications. The authors experimentally evaluate a first-fit strategy for max-coloring on circular arc graphs; in their experiments the first-fit strategy produces a solution with weight within 2.1% of the optimal weight. We point out later that using our algorithm for interval graphs, a 3-approximation for circular arc graphs is easily obtained. The max-coloring problem was also investigated by Demange, et al. [57], where they study the complexity of max-coloring and present approximation algorithms for various classes of graphs. They analyze their algorithms both under the standard approximation ratio described in Chapter 1, and the *differential approximation ratio*[2]. Specifically, in [57], the authors present polynomial time algorithms for co-graphs, bipartite graphs with weights $\{1, t\}$, and line graphs of bipartite graphs with weights $\{1, t\}$. They show NP-hardness of max-coloring for bipartite graphs, split graphs and line graphs of k-regular bipartite graphs. For bipartite graphs, they present a $4r_w/(3r_w + 2)$-approximation algorithm, where r_w is the ratio of the maximum to minimum weights of the vertices. In [26], the authors study the max-coloring problem on planar, bipartite and split graphs.

---

[2]The differential approximation ratio measures how the value of an approximation is placed in the interval between the worst and optimal solutions. More precisely, for a minimization problem Π, and instance I of Π, let $worst(I), cost_A(I), OPT(I)$ denote the worst possible solution for the instance I, the cost of the solution produced by algorithm A and the optimum solution for the instance. Then the differential approximation ratio is defined as $\frac{|worst(I)-cost_A(I)|}{|worst(I)-OPT(I)|}$

They show that the max-coloring problem is NP-hard on triangle-free planar graphs. Further, they also present a $7/6 - \epsilon$ hardness of approximation for the edge-coloring version on bipartite graphs.

The problem of scheduling a set of conflicting jobs in batches has also been studied by Epstein, et al. [19], where they study the problem with the objective of minimizing the *average completion time* of the jobs. They present a 4-approximation algorithm for perfect graphs.

3.3 Interval Graphs

In this section, we show that the max-coloring problem on interval graphs is NP-Complete and present a 2-approximation algorithm. The max-coloring problem on interval graphs is related to the well-studied *dynamic storage allocation problem*(DSA), described in Chapter 1. Recall that an instance of DSA consists of an interval graph $G = (V, E)$ and a weight function $w : V \rightarrow \mathbf{N}$. A feasible solution to this problem in an assignment of an interval $I(v)$ to each vertex v such that $|I(v)| = w(v)$ and $I(u) \cap I(v) = \emptyset$ if u and v are adjacent vertices. The goal is to minimize $|\cup_{v \in V} I(v)|$. Stockmeyer proved this problem NP-complete in 1976 (see problem SR2 in Garey and Johnson [31]) and Kierstead [50] presented the first constant-factor approximation algorithm in 1988. This was an 80-approximation algorithm that used a first-fit strategy to perform on-line coloring of unweighted interval graphs. Kierstead [51] subsequently improved this to a 6-approximation algorithm, which was then improved by Gergov [33, 34] to a 5-approximation and then a 3-approximation algorithm. Recently, Buchsbaum et al. [12] presented a $(2+\varepsilon)$-approximation for this

problem.

In Chapter 1 we showed how the max-coloring problem and interval coloring problems are similar when stated as rectangle packing problems. However, as we show below, the weights of optimal solutions for the two problems on the same input can be quite different. Let OPT_I denote the weight of an optimal interval coloring and let OPT_M denote the weight of an optimal max-coloring for a given instance. Since any feasible solution of the max-coloring problem is also a feasible solution to the interval coloring problem, it follows that $OPT_I \leq OPT_M$. For any clique Q in the given graph, every vertex in the clique needs to have a distinct color and therefore $\sum_{v \in Q} w(v)$ is a lower bound on both OPT_I and OPT_M. Let $LOAD$ denote the maximum over all cliques Q in G of $\sum_{v \in Q} w(v)$. Equivalently, in the context of rectangle packings, $LOAD$ is the maximum sum of heights of rectangles that intersect any vertical line. Clearly, $LOAD \leq OPT_I \leq OPT_M$ and Gergov [34] shows that $OPT_I \leq 3 \cdot LOAD$. Buchsbaum et al. [12] further investigate the relationship between $LOAD$ and OPT_I and show that $OPT_I \leq (2 + \epsilon)LOAD$.

It is easy to construct an instance for which $LOAD$ and OPT_I are poor lower bounds on OPT_M. Consider the weighted intervals shown in Figure 3.1. These form n disjoint cliques, $Q_1, Q_2, \ldots, Q_n$, where clique Q_i contains i intervals each with weight $\lceil W/i \rceil$, where $W \geq n$ is an integer. Letting $w(Q_i)$ denote $\sum_{v \in Q_i} w(v)$ we see that $w(Q_i) = i \cdot \lceil W/i \rceil \leq W + (i-1)$. From this it follows that $LOAD \leq W + (n-1)$. Also note that for this instance $LOAD = OPT_I$. It can be verified that the optimal solution for max-coloring is an n-coloring $C_1, C_2, \ldots, C_n$, where C_i contains exactly one interval

each from $Q_n, Q_{n-1}, \ldots, Q_i$. Letting $w(C_i)$ denote $\max_{v \in C_i} w(v)$ we see that $w(C_i) = \lceil W/i \rceil$. This implies that $OPT_M = \sum_{i=1}^{n} \lceil W/i \rceil \geq W \cdot H_n$, where H_n is the nth harmonic number. The upper bound on $LOAD$ along with the above lower bound on OPT_M together imply that for this family of instances $OPT_M = \Omega(LOAD \cdot \log n)$. Despite the fact that the obvious lower bound on OPT_M, namely $LOAD$ can be rather loose, we are able to develop several $O(1)$-approximation algorithms for the max-coloring problem.

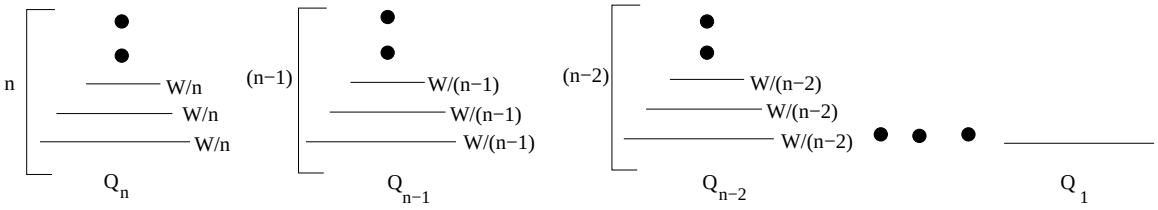

Figure 3.1: An example of a weighted interval graph for which $OPT_M = \Omega(LOAD \cdot \log n)$.

3.3.1 A 2-approximation algorithm

To develop the 2-approximation algorithm, we use a reduction to *online coloring*. Recall that in an instance of the *on-line graph coloring problem*, vertices of a graph are presented one at a time and when a vertex is presented, all edges connecting that vertex to previously presented vertices are also revealed. Each vertex must be assigned a color immediately after it has been presented (and before the next vertex is presented), and a color assigned to a vertex cannot be changed later. An algorithm for the on-line graph coloring problem assigns colors to vertices in the

manner described above, so as to construct a proper vertex coloring of the graph. We say that an algorithm A for the on-line graph coloring problem k-colors a graph G, if no matter which order the vertices of G are presented A uses at most k colors to color G.

Let A be an algorithm for the on-line graph coloring problem. We use A as a "black-box" to devise a simple algorithm for the max-coloring problem. The algorithm, called `MCA` (short for max-coloring algorithm) is given below.

Algorithm 1 MCA(G,w)

1:Sort vertices of G in non-increasing order of weights.
(Let $(v_1, \cdots, v_n)$ be this ordering).
2:Present the vertices in this order to A.
3:Return the coloring produced by A.

We will now make a connection between the number of colors used by A and the weight of the coloring produced by `MCA`. This connection, along with known results on on-line coloring of interval graphs will lead to constant factor approximation algorithms for max-coloring for interval graphs.

Theorem 3.3.1 *Let $\mathcal{C}$ be a hereditary class of graphs and let A be an algorithm for on-line graph coloring such that for some integer constant $c > 0$ and for any graph $G \in \mathcal{C}$, A k-colors G for some $k \leq c \cdot \chi(G)$. Then, for any $G \in \mathcal{C}$ and for any weight function $w : V(G) \rightarrow \mathbf{N}$, `MCA` produces a coloring for G whose weight is at most $c \cdot OPT_M(G)$.*

Proof: Let $C_1, C_2, \ldots, C_k$ be a coloring of G that is optimal for the max-coloring problem. Let $w_i = \max_{v \in C_i} w(v)$ and without loss of generality assume that $w_1 \geq w_2 \geq \cdots \geq w_k$. Now note that $k \geq \chi(G)$ and $OPT_M(G) = \sum_{i=1}^{k} w_i$. Let $A_1, A_2, \ldots, A_t$ be the coloring of G produced by `MCA`. Let $a_i = \max_{v \in A_i} w(v)$ and without loss of generality assume that $a_1 \geq a_2 \geq \cdots \geq a_t$. From our hypothesis it follows that $t \leq c \cdot \chi(G) \leq c \cdot k$. For notational convenience, define sets $A_{t+1} = A_{t+2} = \ldots = A_{c \cdot \chi(G)} = \emptyset$ and let $a_i = 0$ for i, $t < i \leq c \cdot \chi(G)$. We will now claim that for each i, $1 \leq i \leq \chi(G)$, and each j, $c(i-1) < j \leq c \cdot i$, we have $w_i \geq a_j$. Showing this would imply the result we seek because the coloring produced by `MCA` has weight

$$\begin{aligned}
\sum_{\ell=1}^{c \cdot \chi(G)} a_\ell &= \sum_{i=1}^{\chi(G)} \sum_{j=c(i-1)+1}^{c \cdot i} a_j \\
&\leq \sum_{i=1}^{\chi(G)} c \cdot w_i \\
&\leq c \cdot OPT_M(G).
\end{aligned}$$

Since w_1 is the maximum weight of any vertex in G, the claim is trivially true for $i = 1$. For any $i \geq 2$, let $V_i \subseteq V$ be defined as $V_i = \{v \mid w(v) > w_i\}$. The coloring $C_1, C_2, \ldots, C_{i-1}$ is an $i-1$ coloring of $G[V_i]$.

Because of the order in which vertices are presented to A, all vertices in V_i are presented to A before any vertex with weight w_i. Therefore, by our hypothesis, algorithm A colors $G[V_i]$ with no more than $c \cdot (i-1)$ colors. Therefore, the weight of the heaviest vertex in color classes A_j for j, $c(i-1) < j \leq c \cdot i - 1$ is at most w_i.

See Figure 3.2 □

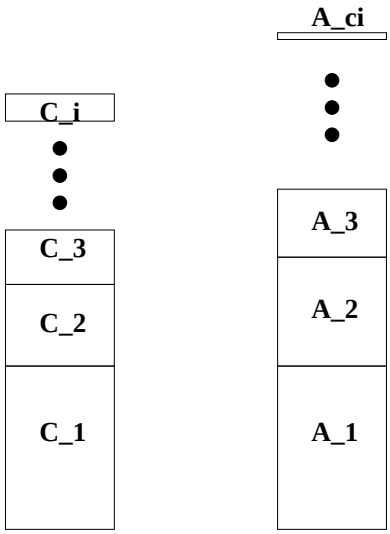

Figure 3.2: The figure shows the color classes of OPT_M on the left, and the color classes of MCA on the right. We show that the weight of the $(c \cdot i)^{th}$ color class produced by MCA is at most the weight of the i^{th} color class of OPT_M.

From the results in Chapter 2, it follows from Theorem 3.3.1 that if we use the first-fit algorithm for on-line coloring the interval graph, then max-coloring has an 8-approximation. However, as noted in Chapter 2, Kierstead and Trotter [53] presented a simple algorithm for on-line coloring of interval graphs that is $3k - 2$-competitive, where $\chi(G) = k$. From this result, and Theorem 3.3.1, a 3-approximation algorithm for max-coloring interval graphs follows, after noting that interval graphs are a hereditary class of graphs.

Theorem 3.3.2 *There is a 3-approximation algorithm for solving the max-coloring*

problem on interval graphs.

Rather than use the Kierstead-Trotter algorithm as a black box, if we make a simple modification to one of the steps in the Kierstead-Trotter algorithm, we can reduce the approximation factor from 3 to 2. The Kierstead-Trotter algorithm maintains sets $S_1, S_2, \ldots$ such that when a vertex u is presented, it finds the smallest i such that $S_1 \cup S_2 \cup \ldots \cup (S_i \cup \{u\})$ does not contain an $(i+1)$-clique. The vertex u is then inserted into S_i. Kierstead and Trotter show that each induced subgraph $G[S_i]$ is the union of disjoint paths. Therefore $G[S_i]$ can be 3-colored using the "first-fit" on-line algorithm that assigns to each presented vertex the smallest available color. The new algorithm for max-coloring is given below.

Algorithm 2 BETTER-MCA(G,w)

1:Sort the vertices of G in non-increasing order of weights.
(Let $(v_1, \cdots, v_n)$ be this ordering of the vertices of G).
2:Let $k \leftarrow 1$; $S_k \leftarrow \emptyset$.
3:**for** $j \leftarrow 1$ **to** n **do**
  4:Insert v_j into set S_i
  5:where $i \leq k$ is the smallest value such that
  6:$S_1 \cup S_2 \cup \ldots \cup (S_i \cup \{u\})$ does not contain an $(i+1)$-clique.
  7:If no such i exists,
  8:set $k \leftarrow k+1$ and insert v_j into S_k
9:**end for**
10:Use color 1 to color vertices in S_1
11:**for** $i \leftarrow 2$ **to** k **do**
  12:Use colors $2i-2$ and $2i-1$ to 2-color the vertices in S_i
13:**end for**
14:Return the coloring.

Steps (2) and (3) come from the Kierstead-Trotter algorithm. The modifica-

tion we make to the Kierstead-Trotter algorithm is that instead of coloring S_i on-line with 3 colors, we just color S_i off-line using two colors. This is possible because, as mentioned above, $G[S_i]$ is a union of disjoint paths [53].

Theorem 3.3.3 `BETTER-MCA` *is a 2-approximation algorithm for the max-coloring problem on interval graphs.*

Proof: Let $C_1, C_2, \ldots, C_k$ be a coloring of G that is optimal for the max-coloring problem. Let $w_i = \max_{v \in C_i} w(v)$ and without loss of generality assume that $w_1 \geq w_2 \geq \cdots \geq w_k$. Now note that $k \geq \chi(G)$ and $OPT_M(G) = \sum_{i=1}^{k} w_i$.

Suppose that at the end of Step (3) in `BETTER-MCA`, we have sets $S_1, S_2, \ldots, S_t$. An element is inserted into S_t only because $S_1 \cup S_2 \cup \ldots \cup (S_{t-1} \cup \{u\})$ has a t-clique. Therefore, $\chi(G) \geq t$ and it follows that $t \leq k$. Let $s_i = \max_{v \in S_i} w(v)$. We now claim that for each i, $1 \leq i \leq t$, $s_i \leq w_i$.

It is clear that $s_1 = w_1$. To obtain a contradiction, suppose that for some i, $s_i > w_i$ and let i be the smallest such value. This implies any vertex x with weight s_i or larger is in an earlier color class, C_j, for some $j < i$ in the optimal coloring. Therefore, vertices with weight s_i or larger are $(i-1)$-colorable, so they induce a clique of size at most $i-1$ and therefore in Step (3) no vertex with weight s_i will get inserted into S_i - a contradiction.

In Steps (4) and (5) we convert the vertex partition $S_1, S_2, \ldots, S_t$ into a coloring whose weight is at most $s_1 + 2 \cdot \sum_{i=2}^{t} s_i$. The following inequalities

$$s_1 + 2 \cdot \sum_{i=2}^{t} s_i \leq w_1 + 2 \cdot \sum_{i=2}^{t} w_i \leq 2 \cdot OPT_M(G)$$

give the result we seek. □

3.3.2 NP-hardness

In this section, we show that the problem of max-coloring interval graphs is NP-hard. Our proof is considerably simpler than the one presented in the preliminary version [63]. We show that the decision problem for max-coloring interval graphs is NP-complete. The decision problem is defined as follows.

MAX-COLORING INTERVAL GRAPHS

INPUT: An interval graph $G = (V, E)$ with weight function $w : V \rightarrow \mathbf{N}$, and an integer k

QUESTION : Does G have a max-coloring of cost at most k

The reduction is from coloring *circular-arc graphs*. Recall from Chapter 1 that a graph $G = (V, E)$ is a circular-arc graph if its vertices can be placed in one-to-one correspondence with open arcs on the circumference of a circle such that two arcs intersect if and only if the corresponding vertices are adjacent. The decision version of the problem, whether there exists a k-coloring for a given circular-arc graph $G = (V, E)$ was proved NP-complete by Garey, et al. in [32]. We now describe the problem and the reduction.

CIRCULAR-ARC GRAPH COLORING

INPUT : A circular-arc graph $G = (V, E)$ and an integer k.

QUESTION : Does G have a coloring with $\leq k$ colors ?

We may assume that a circular arc representation of G is given to us, since

there exist algorithms for recognizing circular-arc graphs, that produce a circular-arc representation of the graph [56]. Without loss of generality, we can assume that there exists a point on the circle that is contained in precisely k circular arcs. If not, consider a sector that does not contain any end-points of any of the arcs. Assume that this sector is contained in $l < k$ arcs. We can introduce $k - l$ arcs that are contained only in this sector. The modified graph is k-colorable if and only if the original graph is k-colorable. This is because any coloring of the original graph that uses at least k colors can be extended to the modified graph as the newly added arcs have at least $k - l$ available colors. (Note that if the sector was contained in at least $k + 1$ arcs, then G is not k-colorable, and the reduction can be trivially completed.)

Theorem 3.3.4 *MAX-COLORING INTERVAL GRAPHS is NP-complete.*

Proof: Given a circular-arc graph, $G = (V, E)$, consider a ray r from the center of the circle that intersects precisely k circular arcs. For each such arc v_i, $i = 1, \cdots, k$, partition v_i into two arcs l_i and r_i, where l_i has the same left end-point as v_i (in a clockwise direction), and has as right end-point, the point of intersection of r and v_i. r_i has left end-point the intersection point of r and v_i, and it's right end-point is that of v_i. This gives an interval graph $G' = (V', E')$. The weights of the vertices V' are as follows. For each l_i, r_i, let $w(l_i) = w(r_i) = i$. For all other vertices v, let $w(v) = 1$. See Figures 3.3 and 3.4.

If the circular-arc graph G has a k-coloring then, assigning $color_{G'}(v) = color_G(v)$, for all $v \notin \{l_i, r_i,\ i = 1, \cdots, k\}$, and $color_{G'}(l_i) = color_{G'}(r_i) = color_G(v_i)$, $i = 1, \cdots, k$ gives a max-coloring of G' of cost $k(k + 1)/2$.

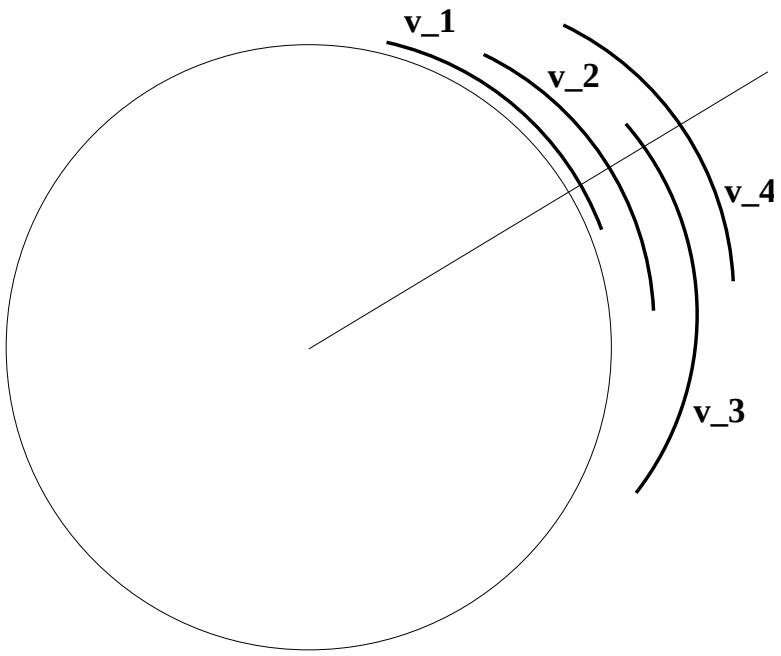

Figure 3.3: A circular-arc graph with $k = 4$ arcs crossing the ray from the center

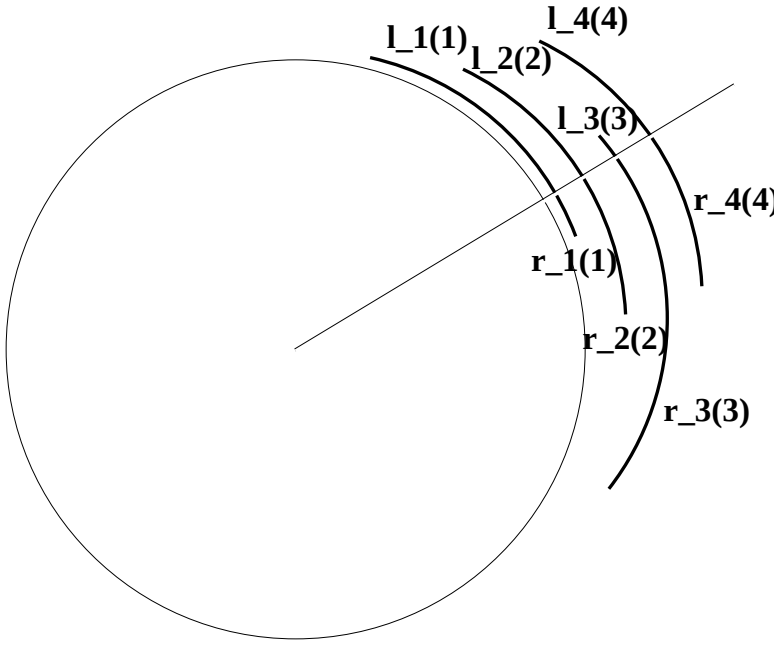

Figure 3.4: The 4 arcs crossing the ray are split into rays l_i and r_i for $i = 1, \cdots, 4$. The i^{th} arc is assigned a cost of i. The weight of new arcs are shown in brackets.

Note that the intervals l_i and the intervals r_i each form a clique of weight $k(k+1)/2$. This is a lower bound on the max-coloring cost of G'. If G' has a max-coloring of cost $k(k+1)/2$, then we must have (1) $color_{G'}(l_i) = color_{G'}(r_i)$, for $i = 1, \cdots k$, and (2) the number of colors used in the max-coloring is k. Thus by setting $color_G(v_i) = color_{G'}(l_i) = color_{G'}(r_i)$, and $color_G(v) = color_{G'}(v)$ for all other vertices v, we get a k-coloring of G. □

3.4 Max-Coloring Trees and Bipartite Graphs

3.4.1 Max-Coloring Trees

In this section, we present results on the max-coloring problem for trees and bipartite graphs. The results on bipartite graphs were also obtained independently by de Werra, et al. [57]. The max-coloring problem has turned out to be surprisingly hard even for trees. For trees, we don't know if the max-coloring problem can be solved in polynomial time. We present (i) a sub-exponential exact algorithm and (ii) a PTAS. We start with an observation on the distribution of weights of color classes in an optimal max-coloring of bipartite graphs.

Lemma 3.4.1 *Let G be a bipartite graph. In any optimal max-coloring $\{C_1, C_2, \ldots, C_k\}$ of G with $w_i = weight(C_i)$ and $w_1 \geq w_2 \geq \cdots \geq w_k$, we have that $w_i \geq \sum_{j=i+1}^{k} w_j$, $i = 1, \cdots, k-1$.*

Proof: If $w_i < \sum_{j=i+1}^{k} w_j$, then the subgraph induced by vertices in $\cup_{j=i}^{k} C_j$ can be colored with two colors with weight at most $2w_i$. This coloring has weight less than

the weight of $\{C_1, C_2, \ldots, C_k\}$, a contradiction. □

Corollary 3.4.2 *Let G be a bipartite graph. In any optimal max-coloring $\{C_1, C_2, \ldots, C_k\}$ of G with $w_i = weight(C_i)$ and $w_1 \geq w_2 \geq \cdots \geq w_k$, we have that $\frac{w_i}{2} \geq w_{i+2}$, for $i = 1, \cdots, k-2$, and hence, $w_1 \geq 2^{\lfloor (i-1)/2 \rfloor} \cdot w_i$.*

Since the weights of the color classes decrease rapidly, we can expect that the max-color number of a tree may not be too high. We now state three upper bounds on χ_{mc} of trees, the first of which applies to arbitrary graphs as well.

Lemma 3.4.3 *Let G be a vertex-weighted graph with maximum vertex degree Δ. Then $\chi_{mc}(G) \leq \Delta + 1$.*

Proof: Let $k = \chi_{mc}(G)$ and let $C_1, C_2, \ldots, C_k$ be an optimal max-coloring of G. Let $w_i = weight(C_i)$ and without loss of generality assume that $w_1 \geq w_2 \geq \cdots \geq w_k$. If $k > \Delta + 1$, then for each vertex $v \in C_k$, there is a color class C_i, $i < k$, such that v has no neighbors in C_i. Note that since $w_i \geq w_k$, when v is moved into C_i, the weight of the coloring does not increase. Furthermore, when C_k becomes empty, the weight of the coloring decreases, contradicting the optimality of $C_1, C_2, \ldots, C_k$. □

Lemma 3.4.4 *Let T be a vertex-weighted tree on n vertices. Then, $\chi_{mc}(T) \leq \lfloor \log_2 n \rfloor + 1$.*

Proof: Let $k = \chi_{mc}(G)$ and let $C_1, C_2, \ldots, C_k$ be an optimal max-coloring of G. Let $w_i = weight(C_i)$ and without loss of generality assume that $w_1 \geq w_2 \geq \cdots \geq w_k$. For each $i > 1$, we can assume without loss of generality that every vertex $v \in C_i$ has a neighbor in C_j, for every $j < i$.

For each vertex $v \in C_1$, let $T(v)$ denote the rooted tree with one vertex, namely v. For each $v \in C_i$, $i > 1$, define $T(v)$ as the tree rooted at v, such that (i) the children of v in $T(v)$ are exactly the neighbors of v in T belonging to color classes $C_1, C_2, \ldots, C_{i-1}$, and (ii) for each child u of v, the subtree of $T(v)$ rooted at u is simply $T(u)$. For each i, $1 \leq i \leq k$, let $S_i = \min\{|T(v)| \mid v \in C_i\}$. In other words, S_i is the size of a smallest tree $T(v)$ rooted at a vertex v in C_i. Then,

$$\begin{aligned} S_1 &= 1 \\ S_i &\geq \sum_{j=1}^{i-1} S_j + 1, \text{ for each } i > 1 \end{aligned}$$

This implies that $S_i \geq 2^{i-1}$, $1 \leq i \leq k$. Using the fact that $S_k \leq n$, we get $\chi_{mc} = k \leq \lfloor \log_2 n \rfloor + 1$. □

Lemma 3.4.5 *Let T be a vertex-weighted tree on n vertices, and let W be the ratio of the weight of heaviest vertex to the weight of the least heavy vertex. Then, $\chi_{mc}(T) \leq \lceil \log_2 W \rceil + 1$.*

Proof: Let $k = \chi_{mc}(T)$ and let $C_1, C_2, \ldots, C_k$ be an optimal max-coloring of T. For $1 \leq i \leq k$, let $w_i = weight(C_i)$ and without loss of generality assume that $w_1 \geq w_2 \geq \cdots \geq w_k$. Thus w_1 is the weight of the heaviest vertex in the tree. Let $\ell = \min\{t \in \mathbf{N} \mid \text{ for all } v \in V(T), w(v) \geq w_1/2^t\}$. Therefore, $\ell = \lceil \log_2 W \rceil$.

Consider the collection of disjoint intervals $\mathcal{I} = \{I_0, I_1, \ldots, I_{\ell-1}\}$, where $I_i = [\frac{w_1}{2^{i+1}}, \frac{w_1}{2^i})$, for $i = 1, \ldots, \ell - 1$ and let $I_0 = [\frac{w_1}{2}, w_1]$. Because of the choice of ℓ, for each vertex $v \in V(T)$, $w(v)$ belongs to exactly one interval I_j. Let $V_j = \{v \in V(T) \mid$

$w(v) \in I_j\}$, $j = 0, 1, \ldots, \ell - 1$. We say that a vertex v *contributes* to a color class C_i if $v \in C_i$, and $w(v) = \max\{w(u) \mid u \in C_i\}$. The *contribution* of an interval I_j is the maximum number of vertices in V_j that contribute to distinct color classes.

Corollary 3.4.2 tells us that $w_i \geq 2 \cdot w_{i+2}$ for $i = 1, \cdots k-2$. This immediately implies that no interval I_j, $j = 1, 2, \ldots, \ell - 1$ has a contribution of more than two. If the contribution of I_0 is three or more, then it must be the case that we can construct a 2-coloring with the same or smaller weight, compared to $\{C_1, C_2, \ldots, C_k\}$. This contradicts the fact that $k = \chi_{mc}(T)$ and $C_1, C_2, \ldots, C_k$ is an optimal max-coloring of T.

Now suppose that intervals $I_{i_1}, I_{i_2}, \ldots, I_{i_t}$, $0 \leq i_1 < i_2 < \cdots < i_t \leq \ell - 1$, is the sequence of all intervals in $\mathcal{I}$, each of whose contribution is two. We now prove the following claim:

> **Claim:** For any pair of consecutive intervals I_p, $p = i_j$ and I_q, $q = i_{j+1}$, where $j < t$, it is the case that there is an interval in $\{I_{p+1}, I_{p+2}, \ldots, I_{q-1}\}$ with contribution zero.

If we can show this claim, then we can charge the "extra" contribution of each I_{i_j} to an interval between I_{i_j} and $I_{i_{j+1}}$, whose contribution is zero. This implies that the contributing vertices in all intervals except I_t can be reassigned to a distinct interval. Since there are ℓ intervals and since the contribution of I_t is at most two, there is a total contribution of at most $\ell + 1$, implying that there are at most $\ell + 1$ color classes.

We prove the above claim by contradiction, assuming that the contribution of every interval in $\{I_{p+1}, I_{p+2}, \ldots, I_{q-1}\}$ is one. Let $\{x_p, x_{p+1}, \ldots, x_q\} \cup \{y_p, y_q\}$ be

vertices such that (i) for each $j = p, p+1, \ldots, q$, $x_j \in V_j$ and x_j contributes to some color class and (ii) for each $j \in \{p, q\}$, $y_j \in V_j$ and x_j and y_j contribute to distinct color classes. Since $x_j \in V_j$, $w(x_j) \geq \frac{w_1}{2^{j+1}}$, $j = p, p+1, \ldots, q$. Also, since $y_q \in V_q$, $w(y_q) \geq \frac{w_1}{2^{q+1}}$. Therefore,

$$\begin{aligned}
\sum_{j=p}^{q} w(x_j) + w(y_q) &\geq \sum_{j=p}^{q} \frac{w_1}{2^{j+1}} + \frac{w_1}{2^{q+1}} \\
&= w_1 \frac{2^{q-p+1} - 1}{2^{q+1}} + \frac{w_1}{2^{q+1}} \\
&= \frac{w_1}{2^p} > w(y_p)
\end{aligned}$$

This contradicts Lemma 3.4.1 and proves the claim. □

The upper bounds in the three preceding lemmas are all tight, as the following example will show. Let $T_0, T_1, T_2, \ldots$ be a sequence of trees where T_0 is a single vertex, with weight 1, and T_i, $i > 0$, is constructed from T_{i-1} as follows. Let $V(T_{i-1}) = \{u_1, u_2, \ldots, u_k\}$. To construct T_i, start with T_{i-1} and add a set of new vertices $\{v_1, v_2, \ldots, v_k\}$, each with weight 2^i, and edges $\{u_i, v_i\}$ for all $i = 1, 2, \ldots, k$. Thus the leaves of T_i are $\{v_1, v_2, \ldots, v_k\}$ and every other vertex in T_i has a neighbor v_j for some j. Now consider a tree T_n in this sequence. Clearly, $|V(T_n)| = 2^n$ and the maximum vertex degree of T_n, $\Delta(T_n) = n - 1$. See Figure 4.1 for T_0, T_1, T_2, and T_3. Now consider the coloring of T_n defined by $C_i = \{$ leaves of $T_{n+1-i}\}$, $1 \leq i \leq n+1$. Note that $weight(C_i) = 2^{n+1-i}$ and therefore the weight of the coloring is $2^{n+1} - 1$. It is not hard to see that any coloring using fewer than $n+1$ colors has weight at least

2^{n+1}. Therefore,

$$\chi_{mc} = n + 1 = \log_2 |V(T_n)| + 1 = \Delta + 1 = \log_2 \left(\frac{2^n}{1} \right) + 1.$$

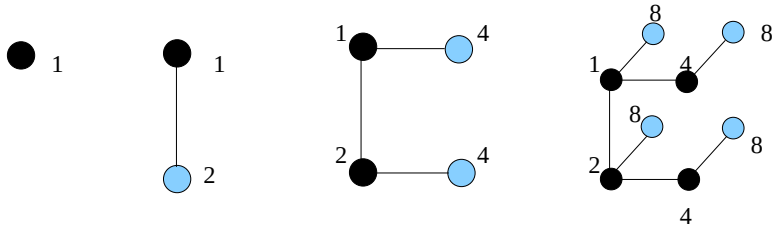

Figure 3.5: Sequence of trees which show that the upper bounds of Lemmas 3.4.3, 3.4.4 and 3.4.5 are all tight. The figure above shows the trees $T_0, \cdots, T_3$.

Now we show that if the tree has a constant number of distinct weights, we can find an optimal max-coloring in polynomial time. Later this will turn out to be critical for our PTAS. We deal with the case of constant number of distinct weights via the solution to a problem called `FEASIBLE` k-`COLORING`.

`FEASIBLE` k-`COLORING`

INPUT: A tree T with weight function $w : V \to \mathbf{N}$, and a sequence $(W_1, W_2, \cdots, W_k)$ of positive integers, satisfying $W_1 \geq W_2 \geq \cdots \geq W_k$.

OUTPUT: Either a coloring of the tree into color classes $A_1, \cdots, A_k$, such that for all $v \in A_i$, $w(v) \leq W_i$ or if such a coloring does not exist, a report that no such

feasible coloring exists.

Here is a simple dynamic programming algorithm for solving `FEASIBLE` k-`COLORING` on trees in $O(nk)$ time. Let T be rooted at an arbitrary vertex r. Let $ch(v)$ denote the set of children of v, and let $parent(v)$ denote the parent of v. For a vertex v, let $T(v)$ denote the sub-tree of T rooted at v. Let $[k]$ denote the set of colors $\{1, \cdots, k\}$. For a vertex v, let $F(v) = \{j \mid w(v) > W_j\}$ be the set of forbidden colors, and let $S(v) = \{i \mid$ there exists a feasible coloring of $T(v)$ with $color(v) = i\}$. The algorithm to compute a feasible coloring, if one exists is as follows. The correctness of the algorithm and the running time are easy to established and this is summarized in the lemma below.

Algorithm 3 FKC

```
1:Let S(v) = φ, ∀ v ∈ T
2:for Each vertex v in a post-order traversal of T do
  3:for each color i ∈ [k] − F(v) do
    4:if S(u) − {i} ≠ φ ,∀u ∈ ch(v) then
      5:Set S(v) = S(v) ∪ {i}.
    6:end if
  7:end for
8:end for
9:if S(r) = φ, return NULL then
  10:Pick an arbitrary i ∈ S(r) and set color(r) = i
11:end if
12:Pick an arbitrary i ∈ S(r) and set color(r) = i.
13:for each v in a pre-order traversal of T do
  14:Pick an arbitrary j ∈ S(v) − color(parent(v)) and set color(v) = j
15:end for
```

Lemma 3.4.6 *Algorithm FKC solves the* `FEASIBLE` k-`COLORING` *problem in* $O(nk)$

time.

The main idea underlying our PTAS is the reduction of the number of distinct weights of the vertices down to a constant. We then pick candidates for the weights of the color classes and for each such choice, using the algorithm for `FEASIBLE` k-`COLORING`, we test if there is a legal coloring of the tree with the chosen weights for the color classes.

We are given a tree T, with weight function $w : V \rightarrow \mathbf{N}$ and an $\epsilon > 0$. Let $c > 0$ be an integer such that $(2 \log c + 3)/c \leq \epsilon$, and let $\alpha = (w_1 - 1)/c$. Let $I_1, I_2, \cdots, I_c$ be a partition of the range $[1, w_1)$, where $I_i = [1 + (i-1)\alpha, 1 + i \cdot \alpha)$, $1 \leq i \leq c$. Let T' be a tree that is identical to T, except in its vertex weights. The tree T' has vertex weights $w' : V \rightarrow \mathbf{N}$ defined by the rule: for any $v \in V$, if $w(v) \in I_j$ then $w'(v) = 1 + (j-1) \cdot \alpha$ and if $w(v) = w_1$, then $w'(v) = w_1$. In other words, except for vertices with maximum weight w_1, all other vertices have their weights "rounded" down. As a result T' has $c + 1$ distinct vertex weights. Now let OPT' denote the weight of an optimal max-coloring of T' and let $\mathcal{C}' = C'_1, C'_2, \ldots, C'_k$ be the color classes corresponding to OPT'_M. Since the weights of vertices have fallen in going from T to T', clearly $OPT'_M \leq OPT_M$. If we use the coloring $\mathcal{C}'$ for T, we get a coloring whose weight is at most $OPT'_M + k\alpha$. Substituting $(w_1 - 1)/c$ for α and noting that $w_1 \leq OPT'_M$, we obtain that weight of $\mathcal{C}'$ used as a coloring for T' is at most $(1 + \frac{k}{c})OPT'_M$ We now show that given the distribution of vertex weights of T', $k = O(\log c)$. To see this first observe that the weights of last three color classes C'_k, C'_{k-1}, and C'_{k-2} cannot all be identical, by Lemma 3.4.1. Also, observe that the

possible vertex weights of T' are $1, 1+\alpha, 1+2\alpha, \ldots$. Therefore, $weight(C'_{k-2}) \geq 1+\alpha$. From Corollary 3.4.2, we obtain

$$1 + \alpha \leq w(C_{k-2}) \leq \frac{w_1}{2^{\lfloor (k-3)/2 \rfloor}}.$$

Solving this for k yields $k \leq 2\log_2(c) + 3$. Therefore, by our choice of c, we have

$$\frac{k}{c} \leq \frac{2\log_2(c) + 3}{c} \leq \epsilon.$$

Thus $(1+\epsilon)OPT'_M$ is an upper bound on the weight of $\mathcal{C}'$ used as a coloring for T. Since $OPT'_M \leq OPT_M$, we see that the weight of OPT'_M used as a coloring for T is at most $(1+\epsilon)OPT_M$.

To construct OPT'_M in polynomial time, for each $k = 1, \ldots, 2\log c + 3$, we generate all $O(c^k)$ possible sequences of weights and call algorithm `FEASIBLE` k`-COLORING` for each subsequence and pick the coloring with the minimum weight. This gives OPT'_M. Each solution to `FEASIBLE` k`-COLORING` takes $O(nk)$ time, and we have $O(c^k)$ sequences, for $k = 1, \ldots, 2\log c + 3$. Using the fact that $(2\log_2 c + 3)/c \leq \epsilon$, a little bit of algebra yields a running time that is linear in n and exponential in $1/\epsilon$.

3.4.2 Max-Coloring Bipartite Graphs.

This section presents an $\frac{8}{7}$-approximation algorithm for the max-coloring problem on bipartite graphs, followed by a hardness of approximation result that shows that for any $\epsilon > 0$, there is no $(\frac{8}{7} - \epsilon)$-approximation algorithm unless $P = NP$. Thus

our approximation algorithm produces an optimal approximation ratio.

One feature of our approximation algorithm is that it uses at most 4 colors, even though though an optimal max-coloring of a bipartite graph may need an unbounded number of colors. Our PTAS for the max-coloring problem on trees relied on the fact that the `FEASIBLE` k`-COLORING` problem on trees can be solved in polynomial time for any k. However, `FEASIBLE` k`-COLORING` is NP-complete for bipartite graphs for $k \geq 3$ [54]. This has forced us to use a different approach for bipartite graphs. Another difference is that in contrast to the $O(\log n)$ upper bound on the number of colors used by an optimal max-coloring for an n-vertex tree, there are simple examples of n-vertex bipartite graphs G with $\chi_{mc}(G) \geq n/2$. One such example is shown in Figure 3.6.

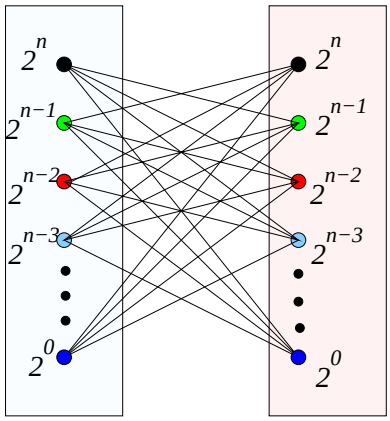

Figure 3.6: An instance of max-coloring of a bipartite graph on $2n$ vertices that requires n colors in an optimal max-coloring. The weights of the vertices are powers of 2 and are shown next to the vertices. A k-coloring for any $k < n$ has weight at least 2^n, while a coloring with n colors has weight $2^n - 1$.

First note that since bipartite graphs are 2-colorable, Lemma 3.4.1 holds and hence if an optimal max-coloring of a bipartite graph uses a large number of colors, the contribution of all but the first few color classes must be quite small. We can use this to our advantage and develop an algorithm that tries to find a *good* approximation to the weights of the first few color classes. We run three algorithms, A_2, A_3, and A_4, that use 2, 3 and 4 colors respectively. The color classes produced by algorithm A_i, $2 \leq i \leq 4$, are denoted $\{A_1^i, A_2^i, \cdots\}$, and the weights of the corresponding color classes are denoted $\{a_1^i, a_2^i, \cdots\}$. We start with a description of algorithm A_2.

Algorithm 4 $A_2(G, w)$

1:**for** each connected component G_i of G **do**
2:Color G_i with colors 1 and 2, such that a vertex with maximum weight is colored 1.
3:**end for**

The fact that A_2 is a 2-approximation immediately follows from the fact that $weight(A_2) \leq 2w_1$, and $w_1 \leq OPT_M$. We encode this result in the following lemma.

Lemma 3.4.7 $weight(A_2) \leq 2w_1$

In an optimum coloring, the weight of the first color class, w_1 is fixed. By using more colors, OPT_M may gain an advantage because it can then push *heavy* vertices into *lower* color classes. We now introduce algorithm A_3 which constructs a 3-coloring of G such that the weight of the second color class is minimized.

Algorithm 5 $A_3(G, w)$

1:Let S be a *maximal* independent set of G picked
by examining vertices in non-increasing weight order.
2:Use Algorithm A_2 to color $G \setminus S$.
3:Rename colors 1 and 2 (used by Algorithm A_3), as colors 2 and 3 respectively.
4:Color S with color 1.

Lemma 3.4.8 *$weight(A_3) \leq w_1 + 2w_2$. If $w_2 \leq \frac{1}{2}w_1$, then $weight(A_3) \leq \frac{3}{2}w_1 + w_2$.*

Proof: In algorithm A_3, $a_1^3 = w_1$. Since S is a maximal independent set selected in non-increasing weight order, the weight of the second color class of OPT_M, w_2 cannot be smaller than the weight of any vertex in $G \setminus S$. Hence, $w_2 \geq a_2^3$. Since $a_3^3 \leq a_2^3$, it follows that $weight(A_3) = a_1^3 + a_2^3 + a_3^3 \leq w_1 + w_2 + w_2 = w_1 + 2w_2$. If $w_2 \leq \frac{1}{2}w_1$, then by plugging this inequality into the above upper bound for $weight(A_3)$, we get the second inequality. □

As a warm-up to our main result, we now show that running A_2 and A_3 together and selecting a coloring with smaller weight gives a 4/3-approximation.

Theorem 3.4.9 *Let A be the algorithm that runs A_2 and A_3 and returns, from among the two colorings produced, a coloring with smaller weight. Algorithm A is a 4/3-approximation algorithm for max-coloring bipartite graphs.*

Proof: There are two cases depending on the relative values of $weight(A_2)$ and $w_1 + 2w_2$. If $weight(A_2) \leq w_1 + 2w_2$, then combining this with the bound from Lemma 3.4.7 we get that $weight(A_2) \leq min\{2w_1, w_1 + 2w_2\}$. Thus,

$$\begin{aligned} weight(A_2) &\leq 2 \cdot w_1, \text{ and} \\ weight(A_2) &\leq w_1 + 2 \cdot w_2 \end{aligned}$$

Adding the first inequality to twice the second, we get,

$$3 \cdot weight(A_2) \leq 4 \cdot w_1 + 4 \cdot w_2$$

Since $OPT_M \geq w_1 + w_2$, we get

$$3 \cdot weight(A_2) \leq 4 \cdot OPT_M$$

If $weight(A_2) > w_1 + 2w_2$, then using Lemma 3.4.8, we derive the inequality $weight(A_3) \leq w_1 + 2w_2 < weight(A_2) \leq 2w_1$. Thus $w_2 \leq w_1/2$. Hence from Lemma 3.4.8 we get $weight(A_3) \leq min\{w_1 + 2w_2, \frac{3}{2}w_1 + w_2\}$.

$$\begin{aligned} weight(A_3) &\leq w_1 + 2 \cdot w_2 \\ weight(A_3) &\leq \frac{3}{2}w_1 + w_2 \end{aligned}$$

Adding the first inequality to twice the second gives

$$3 \cdot weight(A_3) \leq 4 \cdot w_1 + 4 \cdot w_2$$

Using the lower bound of $w_1 + w_2$ on OPT_M, we get

$$3 \cdot weight(A_3) \leq 4 \cdot OPT_M$$

□

We can improve this ratio, by using a fourth color. The greedy strategy employed by algorithm A_3 in selecting the first color class causes a_2^3 to be no larger than w_2. However, it might cause a_3^3 to be significantly larger than w_3. We rectify this situation by introducing algorithm A_4 that uses four colors to color G.

Algorithm 6 $A_4(G, w)$

1:**for** all w^* such that there is a $u \in V$, with $w(u) = w^*$ **do**
2:Partition the vertices of G into two parts
3:$P_1 = \{v \in V \mid w(v) > w^*\}$, and
4:$P_2 = \{v \in V \mid w(v) \leq w^*\}$.
5:Use algorithm A_2 to color P_2
6:Rename colors 1 and 2 as 3 and 4 respectively.
7:Use algorithm A_2 to color P_1.
8:**end for**
9:Return the coloring with minimum weight, over all choices of w^*.

Lemma 3.4.10 $weight(A_4) < w_1 + w_2 + 2w_3$

Proof: Since the weight of every vertex in G is used for the threshold w^*, in some iteration of A_4, $w^* = w_3$. At this point, A_4 partitions the vertex set such that $P_1 = \{v \mid w(v) > w_3\}$ and $P_2 = \{v \mid w(v) \leq w_3\}$. In this iteration, A_4 colors P_1 with weight at most $w_1 + w_2$, and colors P_2 with weight at most $2w_3$. Since A_4

returns the coloring with minimum weight, over all choices of w^*, it follows that $weight(A_4) \leq w_1 + w_2 + 2w_3$. □

The final algorithm, which we call `Bipartite Max-Color` runs A_2, A_3, A_4, and returns the minimum weight coloring.

Algorithm 7 Bipartite Max-Color(G, w)

1:Run algorithms A_2, A_3, A_4
2:Return the coloring of minimum weight.

Theorem 3.4.11 *Algorithm* `Bipartite Max-Color` *is a* $\frac{8}{7}$*-approximation for the max-coloring problem on bipartite graphs.*

Proof: Let $w(B)$ denote the weight of the coloring produced by algorithm Bipartite Max-Color. From Lemmas 3.4.10, 3.4.8, and 3.4.7, we get the following upper bounds on B.

$$w(B) \leq 2w_1 \tag{3.1}$$

$$w(B) \leq w_1 + 2w_2 \tag{3.2}$$

$$w(B) \leq w_1 + w_2 + 2w_3 \tag{3.3}$$

Now, multiplying the inequality (3.3) by 4, inequality (3.3) by 2, and adding all three, we get

$$7 \cdot w(B) \leq 8 \cdot (w_1 + w_2 + w_3) \leq 8 \cdot OPT_M$$

□

3.4.3 An $(\frac{8}{7} - \epsilon)$-hardness reduction

We now show that the 8/7-approximation produced by the above algorithm is optimal. We do this by showing a matching hardness result via a reduction from the PRE-COLORING EXTENSION problem on bipartite graphs. The PRE-COLORING EXTENSION problem for general graphs is defined below.

PRE-COLORING EXTENSION

Input: A graph $G = (V, E)$, with chromatic number $\chi(G) = r$, a subset $P \subseteq V$, and a proper assignment $c : P \rightarrow \{1, \cdots, r\}$ of colors to vertices in P.

Question: Is there an extension of the proper vertex coloring of P to a proper vertex coloring of G, using colors from $\{1, \cdots, r\}$?

In [54], Kratochvil proved that PRE-COLORING EXTENSION is NP-complete for planar bipartite graphs even when the color bound $r = 3$. We now show a simple gap introducing reduction from PRE-COLORING EXTENSION on bipartite graphs with $r = 3$ to max-coloring on bipartite graphs.

Theorem 3.4.12 *For any $\epsilon > 0$, there is no $(8/7 - \epsilon)$-approximation algorithm for max-coloring on bipartite graphs, unless P=NP.*

Proof: The reduction is from PRE-COLORING EXTENSION on bipartite graphs. Let

the given instance of PRE-COLORING EXTENSION consist of a bipartite graph $G = (V_1, V_2, E)$, a subset $P \subseteq V_1 \cup V_2$, and a proper assignment $c : P \to \{1, 2, 3\}$ of colors to vertices in P. We transform G into a vertex-weighted bipartite graph $G' = (V_1', V_2', E')$ as follows. Add four new vertices, x_1, x_2, y_1, and y_2 to G. Let $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$, $V_1' = V_1 \cup X$, and $V_2' = V_2 \cup Y$. To each vertex $v \in P$, assign a weight $w(v)$ using the rule: $w(v) = 2^{3-i}$ if $c(v) = i$, for each $i \in \{1, 2, 3\}$. If $v \in (V_1 \cup V_2) - P$, set $w(v) = 1$. The new vertices are assigned weights as follows: $w(x_1) = w(y_1) = 4$ and $w(x_2) = w(y_2) = 2$. The edge set of G' contains some additional edges between the new vertices and the old.

$$\begin{aligned} E' \;=\; & E \cup \{\{x_i, y\} | y \in P \cap V_2', \text{ and } w(y) < w(x_i)\} \cup \\ & \{\{y_i, x\} | x \in P \cap V_1' \text{ and } w(x) < w(y_i)\} \cup \{\{x_1, y_2\}\} \cup \{\{x_2, y_1\}\}. \end{aligned}$$

This completes the description of G'. Figure 3.7 illustrates this construction.

Now suppose that the coloring of P can be extended to a proper 3-coloring $c : V_1 \cup V_2 \to \{1, 2, 3\}$ of G. Start with the coloring c and extend this to a proper vertex coloring of G' by assigning colors to the new vertices as follows: $c(x_1) = c(y_1) = 1$ and $c(x_2) = x(y_2) = 2$. To see that this coloring of G' is proper, observe that all neighbors of x_1 have weights 1 or 2 and are in $P \cup \{y_2\}$. By our construction of G' from G, all such neighbors, with the exception of y_2, were colored in the given pre-coloring of P with some color distinct from 1. Furthermore, $c(y_2) = 2 \neq c(x_1)$. A similar argument shows that the colors assigned to y_1, y_2, and x_2 are all proper.

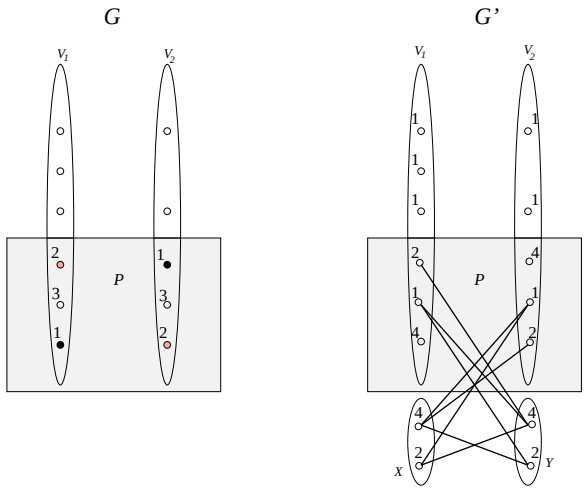

Figure 3.7: On the left is an instance G of PRE-COLORING EXTENSION for bipartite graphs, with $r = 3$. Vertices in the set P are "pre-colored" with colors from $\{1, 2, 3\}$. On the right is the bipartite instance G' of max-coloring, constructed from G. The new vertices in $X \cup Y$, the assignment of weights to vertices, and the edges from the new vertices to the old vertices, are all shown.

Now we show that the coloring c has weight at most 7. Since the maximum weight of any vertex in G' is 4, the weight of color class 1 is at most 4. Also, no vertex with weight 4 is in color class 2. This is because our construction was such that any vertex in P assigned weight 4 has a pre-coloring of 1. The only other vertices with weight 4 are x_1 and y_1. These have been explicitly colored 1. Therefore, the maximum weight of a vertex in color class 2 is 2. A similar argument shows that no vertex with weight 2 or more is in color class 3, thereby showing that the weight of color class 3 is at most 1. This shows that the coloring c has weight at most 7.

Now suppose that G does not have a pre-coloring extension. We show by contradiction that in this case G' does not have a proper vertex coloring of weight

less than 8. So suppose that there is a proper vertex coloring $c' : V_1' \cup V_2' \rightarrow \{1, 2, \ldots\}$ of weight less than 8. Without loss of generality, assume that in this coloring, the color classes are labeled in non-increasing order of their weight. Therefore, all vertices of weight 4 are in color class 1. This includes vertices x_1 and y_1 and this forces all vertices of weight 2 to be excluded from color class 1. Since color class 1 has weight 4, to prevent the total weight of the coloring from reaching 8, all vertices of weight 2 have to be included in color class 2. This includes vertices x_2 and y_2, and so this color class is also non-empty. Therefore the total weight of color classes 1 and 2 is 6. Since c' is a coloring of G' of weight less than 8, it must be the case that color class k, for each $k \geq 4$, is empty. This means that c' is a 3-coloring of G'. Furthermore, it is a 3-coloring of G that respects the pre-coloring of P. This contradicts the assumption that G has no pre-coloring extension and therefore we have that any proper vertex coloring of G' has weight at least 8.

If for some $\epsilon > 0$, there were an $(\frac{8}{7} - \epsilon)$-approximation algorithm for max-coloring bipartite graphs, then using the above polynomial time transformation from G to G', we could distinguish between positive and negative instances of `PRE-COLORING EXTENSION`. This is not possible unless $P = NP$. □

Note that `PRE-COLORING EXTENSION` was proved NP-complete for bipartite, *planar* graphs for $r = 3$. We can modify the above reduction to maintain planarity of the input bipartite graph, by introducing a pair of vertices, one of weight 4 and one of weight 2, to connect to each vertex in P. This shows that max-coloring is impossible to approximate to a factor better than 8/7, even on *planar* bipartite graphs.

3.5 Max-Coloring on Hereditary Graphs

Let $\mathcal{G}$ be a hereditary class of graphs for which the minimum vertex coloring problem has a c-approximation. In other words, there is a polynomial time algorithm A that takes a graph $G \in \mathcal{G}$ as input and returns a proper vertex coloring of G using at most $c \cdot \chi(G)$ colors. In this section, we present two constant factor approximation algorithms for the max-coloring problem. Both algorithms achieve an approximation ratio of $4c$. Both algorithms are based on partitioning the vertex set into disjoint subsets, and solving the ordinary coloring problem in each subset separately. The first algorithm is based on partitioning the vertices into sets of nearly equal weight and solving the ordinary coloring problem on this set. We call this algorithm $WtPartition$. We then present a randomized version of $WtPartition$, called $RandWtPartition$, using which we can improve the approximation ratio to e, the base of the natural logarithm. The second algorithm, *Geomfit* is based on iteratively selecting maximal k-colorable subgraphs, with geometrically increasing k, and coloring each subgraph optimally using the fewest number of colors. Next, we show how we can introduce randomization into *GeomFit* to improve the approximation ratio from 4 to 3. We call this randomized algorithm *RandGeomFit*. The randomized algorithm can be derandomized to give a deterministic algorithm with approximation ratio 3. In both the algorithms in this section, we consider subgraphs of the input graph and color each subgraph optimally. If the underlying class of graphs is hereditary then a coloring algorithm for the class of graphs can be used for vertex induced subgraphs of the input graph, and hence we require the input graph to belong to a hereditary class.

Recall from Chapter 1 that perfect graphs can be optimally colored in polynomial time via the ellipsoid algorithm of Grötschel, Lovász and Schirjver [38]. Therefore, for the class of perfect graphs, both *WtPartition* and *GeomFit* provide a 4-approximation algorithm for max-coloring. These are the first known constant-factor approximation algorithm for perfect graphs. The class of perfect graphs includes many well-known subclasses such as bipartite graphs, interval graphs, chordal graphs, and permutation graphs. For bipartite graphs and interval graphs, we have presented a better approximation algorithm earlier in this chapter, but for chordal graphs and permutations graphs these are the first constant-factor approximation algorithm for max-coloring. In Section 3.3.1, we showed that max-coloring on interval graphs is NP-complete and from this, it follows that max-coloring on perfect graphs is NP-complete as well. In addition, there are well-known classes of graphs that are not perfect, but have constant-factor approximation algorithms for solving the minimum vertex coloring problem. These classes include *circle graphs*, *circular arc graphs*, and *unit disk graphs*. Thus *WtPartition* and *GeomFit* provide an $O(1)$-approximation for all of these classes of graphs.

3.5.1 The Weight Partitioning Algorithm

In this section, we present a simple 4-approximation algorithm for max-coloring. We then show how randomization can help in reducing the approximation factor to e, the base of the natural logarithm. In the next section, we present a second algorithm, whose approximation ratio is also 4. We then show how this approximation ratio can be reduced to 3 using randomization. We state our results for the case where the

classical coloring problem can be solved optimally. The extension to the case where the coloring problem has only a c-approximation algorithm is immediate.

First note that if for a given graph $G = (V, E)$, if we round up the weights of the vertices to the nearest power of 2 and solve the max-coloring problem on this rounded instance, we lose at most a factor of 2, since rounding up the weights of an optimal max-coloring solution is feasible for this rounded instance. Let OPT_R denote the optimum of this rounded instance.

Lemma 3.5.1 $OPT_R \leq 2 \cdot OPT_M$

We solve the max-coloring problem on this rounded instance as follows. Let the weight of the maximum weight vertex in G be denoted W_{max}. Since we are dealing with an instance whose weights are all integral powers of 2, let $W_{max} = 2^k$, for some $k \geq 1$. Partition the vertices into subsets of disjoint weights $R_1, \cdots, R_{k+1}$, where $R_i = \{v \in V |\ w(v) = W_{max}/2^{i-1}\}$. Color each R_i with the fewest number of colors, using a fresh palette for each R_i, and return the union of these color classes as a solution to the max-coloring problem. This is encoded in the algorithm WtPartition(G,w). Let *WtPartition* denote the solution returned by the algorithm.

Algorithm 8 WtPartition(G, w)

1:Round up the vertex weights to the nearest power of 2.
2:**for** $i = 1, \cdots, k+1$ **do**
  3:Let $R_i = \{v \in V | w(v) = W_{max}/2^{i-1}\}$
4:**end for**
5:Color each R_i with the fewest number of colors.
6:Return the union of the colors.

Theorem 3.5.2 WtPartition $\leq 4 \cdot OPT_M$

Proof: Let r_i denote the minimum number of colors required to color the set R_i. Since $WtPartition$ colors each set using the fewest number of colors, and each vertex in R_i has weight $W_{max}/2^{i-1}$, the cost of the solution produced can be written as

$$WtPartition = \sum_{i=1}^{k+1} r_i \cdot W_{max}/2^{i-1} \tag{3.4}$$

Each color class of OPT_R has weight $W_{max}/2^{i-1}$ for some $i = 1, \cdots, k+1$. Let S_i denote the color classes of OPT_R of weight $W_{max}/2^{i-1}$ and let $s_i = |S_i|$, for each $i = 1, \cdots, k+1$. Then, the cost of the optimum solution to the rounded instance can be written as

$$OPT_R = \sum_{i=1}^{k+1} s_i \cdot \frac{W_{max}}{2^{i-1}} \tag{3.5}$$

Since the vertices of weight $W_{max}/2^{i-1}$ only occupy color classes $S_1, \cdots, S_i$ of OPT_R, $R_i \subseteq \cup_{j=1}^{i} S_j$ for each $i = 1, \cdots, k+1$. Since $WtPartition$ colors each set R_i using the fewest number of colors, it follows that

$$r_i \leq \sum_{j=1}^{i} s_j,\ i = 1, \cdots, k+1. \tag{3.6}$$

Applying the inequality 3.6 in Equation 3.4, we get

$$\begin{aligned}
WtPartition &= \sum_{i=1}^{k+1} r_i \cdot \frac{W_{max}}{2^{i-1}} \\
&\leq \sum_{i=1}^{k+1} \Big(\sum_{j=1}^{i} s_i \Big) \cdot \frac{W_{max}}{2^{i-1}} \text{ Changing the order of summation,} \\
&= \sum_{j=1}^{k+1} s_j \cdot \Big(\sum_{i=j}^{k+1} \frac{W_{max}}{2^{i-1}} \Big) \\
&\leq \sum_{j=1}^{k+1} s_j \cdot \frac{W_{max}}{2^{j-1}} \cdot \Big(\sum_{i=1}^{k+2-j} \frac{1}{2^{i-1}} \Big) \text{ From Inequality 3.5,} \\
&\leq 2 \cdot OPT_R \text{ Now, applying Lemma 3.5.1,} \\
&\leq 4 \cdot OPT_M
\end{aligned}$$

□

If the class of graphs we consider have only a c-approximation algorithm for coloring, then $WtPartition$ in Line 5 uses the c-approximate coloring algorithm. We can then claim that for each i, $WtPartition$ uses at most c times the number of colors used by OPT_R to color the sets $\cup_{j=1}^{i} S_i$, to color the set R_i. Hence, inequality 3.6 is

replaced by the following inequality

$$r_i \leq c \cdot \sum_{j=1}^{i} s_i$$

The rest of the analysis goes through, leaving us with a multiplier $4 \cdot c$ instead of 4. Hence,

Theorem 3.5.3 *For a hereditary class of graphs $\mathcal{G}$ if the classical coloring problem has a c-approximation algorithm, then on this class of graphs, max-coloring has a $4 \cdot c$-approximation algorithm.*

Randomized Weight Partition

We can see the algorithm $WtPartition$ in another light. Consider any weight class C of OPT, the original instance. Let $w(C)$ denote the weight of the color class C. Partition the range $[1, w(C)]$ into disjoint ranges

$$\{[w(C), w(C)/2), [w(C)/2, w(C)/4), \cdots, [w(C)/2^k, 1]\}$$

We can now group vertices of C into those that lie in the same weight range and place each such group in a different color class. The cost of this new coloring is at most $2 \cdot w(C)$, since the newly created color classes have total weight at most $\sum_{i=1}^{k} w(C)/2^{i-1}$. However, we lose another factor of 2 in rounding the weights to the nearest power of 2, since we don't know the weight $w(C)$ of each color class of OPT. The worst case is when the weights of each color class are of the form $2^i + 1$ for some

i and we end up paying a factor of 2 for each color class. However, we can search for the right partition by randomization and hence decrease the approximation ratio in expectation.

We now show how we can randomize the above *WtPartition* algorithm to reduce the approximation ratio to e. The randomization technique and approximation ratio essentially follows the work of Halldórsson, et al. [21].

Algorithm 9 RandWtPartition(G, w)

1:Choose α uniformly at random from $[0, 1)$.
2:**for** $i = 0, 1, \cdots, l$ **do**
  3:Let $R_i = \{v |\ W_{max}/q^{i-1+\alpha} \geq w(v) > q^{i+\alpha}\}$.
  4:Color each R_i optimally using fewest colors.
5:**end for**
6:Return the union of the colors.

In the algorithm above $l = \lceil \log_q W_{max} \rceil$, where W_{max} is the maximum weight of any vertex in G.

Theorem 3.5.4 *Algorithm* RandWtPartition *is an e-approximation algorithm for max-coloring perfect graphs.*

Proof: Let $C_1, \cdots, C_k$ denote the color classes of OPT_M. Then, $OPT_M = \sum_{i=1}^{k} w(C_i)$. For a fixed color class C of OPT_M, let $D_i = C \cap R_i$, $i = 0, 1, \cdots, l$. Let v be the maximum weight vertex in color class C, and let $w(v) = W_{max}/q^x$ for some

real number x. Suppose v is in R_i, then

$$\begin{aligned} W_{max}/q^{i-1+\alpha} &\geq w(v) > W_{max}/q^{i+\alpha}, \text{ or,} \\ W_{max}/q^{i-1+\alpha} &\geq W_{max}/q^{x} > W_{max}/q^{i+\alpha} \end{aligned}$$

Hence, $i-1 \leq x-\alpha < i$, and for all $j \leq \lfloor x-\alpha \rfloor$, $C \cap R_j = \emptyset$. Let $i_m = \lfloor x-\alpha \rfloor$.

We now partition C into color classes $D_{i_m+1}, \cdots, D_l$. The sum of weights of the color classes $D_{i_m+1}, \cdots, D_l$ is

$$\begin{aligned} \sum_{i=i_m+1}^{l} weight(D_i) &\leq \sum_{i=i_m+1}^{l} W_{max}/q^{i-1+\alpha} \\ &< \frac{W_{max}}{q^{\alpha-1}}\Big(\frac{1}{q^{i_m+1}} - \frac{1}{q^{l+1}}\Big) \\ &< \frac{q}{q-1}\frac{W_{max}}{q^x} \cdot (q^{x-\alpha-i_m}) \\ &= \frac{q}{q-1} weight(C) \cdot q^{(x-\alpha)-\lfloor x-\alpha \rfloor} \end{aligned}$$

Since α is chosen uniformly in $U[0,1)$, $y = (x-\alpha) - \lfloor x-\alpha \rfloor$ is also distributed uniformly in $[0,1)$. Thus, taking expectation,

$$\begin{aligned} E[\sum_{i=i_m+1}^{l} weight(D_i)] &< \frac{q}{q-1} \cdot weight(C) \cdot \int_0^1 q^y dy \\ &= \frac{q}{q-1} \cdot weight(C) \cdot \frac{q-1}{\ln q} \\ &= \frac{q}{\ln q} \cdot weight(C) \end{aligned}$$

The function $q/\ln q$ is minimized at $q = e$. Hence,

$$E[\sum_{i=0}^{i_{max}} weight(D_i)] < e \cdot weight(C)$$

Thus $E[OPT(\alpha)] \leq e \cdot OPT_M$. □

For any hereditary class of graphs for which we have a c-approximation algorithm, it is easy to see that we get an $e \cdot c$-approximation algorithm. Thus,

Theorem 3.5.5 *Algorithm* RandWtPartition *is an $e \cdot c$-approximation algorithm for max-coloring any hereditary class of graphs $\mathcal{G}$ for which classical coloring has a c-approximation algorithm.*

We next present a second 4-approximation algorithm, but this time we select maximally k-colorable subgraphs of the graph iteratively and color each set optimally.

3.5.2 Chromatic Partitioning

For ease of exposition, below we first describe *GeomFit* assuming that $c = 1$. In other words, we assume that a minimum vertex coloring of the input graph can be computed in polynomial time. To obtain a $4c$-approximation for arbitrary $c > 1$, *GeomFit* needs to be modified very slightly and the analysis that shows the $4c$ approximation factor is quite similar to the analysis in the $c = 1$ case. The modified `GeomFit` and the modified analysis are presented subsequently.

Algorithm 10 GeomFit(G, w)

1:Let $i = 0$, $l_i = 0$
2:**while** $G \neq \phi$ **do**
  3:Set $c_i = 2^i$
  4:Let $G_i = mkc(G, c_i)$
  5:Color G_i optimally using colors $l_i + 1, \cdots, l_i + c_i$
  6:Set $l_{i+1} = l_i + c_i$, $i = i + 1$.
  7:Set $G = G \setminus G_i$.
8:**end while**

A round of the algorithm corresponds to an iteration of the while loop. Suppose that each round is labeled with the value of i at the beginning of that round. For some integer $t > 0$, suppose that the algorithm executes rounds $0, 1, \cdots, t-1$, after which the graph is entirely colored. In each round i, $0 \leq i < t$, the algorithm calls the subroutine $\texttt{mkc}(G, c_i)$, that returns a maximal c_i-colorable subgraph of G, obtained by examining vertices in non-increasing order of weight. Here G is the subgraph of the input graph induced by the not yet colored vertices and $c_i = 2^i$. When called, the subroutine $\texttt{mkc}(G, c_i)$ starts with an empty set S and processes each vertex v of G, in non-increasing order of weight. The subroutine tests if $G[S \cup \{v\}]$ is c_i-colorable or not and if it is, it adds v to S, and proceeds to the next vertex in G. To perform this test, $\texttt{mkc}(G, c_i)$ calls the algorithm A that returns a minimum vertex coloring of G. Assuming that A runs in polynomial time, each call to the subroutine $\texttt{mkc}(G, c_i)$ also runs in polynomial time. Step (5) of the above algorithm is also executed in polynomial time by calling the algorithm A. Since the number of rounds $t = O(\log(n))$, the entire algorithm runs in polynomial time. We start our analysis

of with a simple observation.

Lemma 3.5.6 *If* GeomFit *uses t rounds to color G, then $\chi(G) > c_{t-2}$.*

Proof: In round $t-2$, the algorithm picks a maximal c_{t-2} colorable subgraph of G. If G were c_{t-2}-colorable, then all of it would have been picked up in round $t-2$ or earlier. Since we used one more round to color G, it must mean that $\chi(G) > c_{t-2}$.

□

Without loss of generality, suppose that OPT uses numbers $1, 2, \ldots$ for colors such that color classes are numbered in non-increasing order of weight. Now observe that color classes created in round i by *GeomFit* are all heavier than color classes created in round $i+1$. Without loss of generality, assume that the color classes created in each round of *GeomFit* are numbered in non-increasing order of weight. Let $color_{OPT}(v)$ denote the color assigned to vertex v in OPT, Now using the color classes of OPT we define a pairwise disjoint collection of vertex subsets of G, $\{V_0, \cdots, V_{t-1}\}$, where $V_i = \{v \in G | c_{i-1} < color_{OPT}(v) \leq c_i\}$, $i = 0, \cdots, t-1$. For the definition to make sense, we assume that $c_{-1} = 0$. Since V_{t-1} contains vertices colored $c_{t-2} + 1, c_{t-2}+2, \ldots, c_{t-1}$ by OPT, from Lemma 3.5.6, it follows that $V_{t-1} \neq \phi$. Now we state and prove a critical observation that follows from the greedy choice of a subgraph in each round of *GeomFit*. Let W_i denote the weight of color class $c_{i-1}+1$ in OPT. Note that color class $c_{i-1}+1$ is a subset of V_i and by our labeling convention, it is a heaviest color class in V_i. Similarly, let R_i denote the weight of color class $l_i + 1$ created by *GeomFit*. Note that this is a heaviest color class created in round i be *GeomFit*. Also note that $l_i = \sum_{j=0}^{i-1} c_j = c_i - 1$ and therefore color class $l_i + 1$ is simply

color class c_i.

Lemma 3.5.7 *$R_i \leq W_i$, for $i = 0, 1, \cdots, t-1$.*

Proof: Since R_0 and W_0 are equal to the maximum weight vertex in G, the lemma holds for $i = 0$. By the greedy choice employed in selecting G_0, we ensure that for any other independent set S of G, the maximum weight of a vertex in $G \setminus S$ is at least as large as the maximum weight vertex in $G \setminus G_0$. This ensures that $R_1 \leq W_1$. By the same reasoning, since in round $i-1$, we greedily select a maximal c_{i-1} colorable subgraph of OPT, and $V_1 \cup V_2 \cup \cdots V_{i-1}$ is c_{i-1} colorable, it follows that $R_i \leq W_i$.

□

Theorem 3.5.8 *Let $\mathcal{G}$ be a hereditary class of graphs on which the minimum vertex coloring problem can be solved in polynomial time. Algorithm* GeomFit *is a 4-approximation algorithm for the max-coloring problem on $\mathcal{G}$.*

Proof: The weight of the max-coloring produced by *GeomFit* is bounded above by

$$wt(GeomFit) \;\leq\; \sum_{i=0}^{t-1} c_i \cdot R_i \leq \sum_{i=0}^{t-1} c_i \cdot W_i$$

The first inequality follows from the fact that in each round i, `GeomFit` uses at most c_i colors and a heaviest color class in round i has weight R_i. The second inequality follows from Lemma 3.5.7.

We obtain a lower bound on OPT as follows. The set V_0 contains one color class and this has weight W_0. Now consider a set V_i, $1 \leq i \leq t-2$. It contains

one color class of weight W_i and the remaining color classes have weight at least W_{i+1}. Recall that V_i has color classes labeled $c_{i-1}+1, c_{i-1}+2, \ldots, c_i$ and therefore $weight(V_i) \geq W_i + (c_{i-1}-1)W_{i+1}$.

$$\begin{aligned} OPT \geq \sum_{i=0}^{t-1} weight(V_i) &\geq W_0 + \sum_{i=1}^{t-2}(W_i + (c_{i-1}-1)W_{i+1}) + W_{t-1} \\ &= W_0 + W_1 + \sum_{i=0}^{t-3} c_i W_{i+2}. \end{aligned}$$

Therefore,

$$4 \cdot OPT \geq 4W_0 + 4W_1 + \sum_{i=0}^{t-3} 4c_i W_{i+2} = 4W_0 + 4W_1 + \sum_{i=2}^{t-1} c_i W_i.$$

This lower bound on $4 \cdot OPT$ is larger than the upper bound on $weight(GeomFit)$ above. Therefore, $weight(GeomFit) \leq 4 \cdot OPT$. □

It is worth pointing out a slight strengthening of the above analysis. In the above proof, the upper bound on $weight(GeomFit)$ and the lower bound on $4 \cdot OPT$, can be combined to yield

$$4 \cdot OPT \geq 3W_0 + 2W_1 + weight(GeomFit).$$

Let k denote the chromatic number of the input graph G. Then $k \cdot W_0 \geq OPT$ and therefore we have $weight(GeomFit) \leq (4 - \frac{3}{k})OPT$.

We now assume that $\mathcal{G}$ is a hereditary class of graphs that has a c-approximation algorithm A for the minimum vertex color problem. We modify *GeomFit* so that in

round i, in Step (4), the algorithm computes a maximal $\lfloor c \cdot c_i \rfloor$-colorable subgraph. Correspondingly, in Step (5), G_i is colored using colors $l_i + 1, l_i + 2, \ldots, l_i + \lfloor c \cdot c_i \rfloor$.

The analysis proceeds in a manner similar to the analysis for the $c = 1$ case. Suppose that *GeomFit* finishes coloring G in t rounds, $0, 1, \ldots, t-1$. In round $(t-2)$, *GeomFit* finds a maximal $\lfloor c \cdot c_{t-2} \rfloor$-colorable subgraph and there is at least one uncolored vertex left over for round $t-1$. This implies that algorithm A needs at least $\lfloor c \cdot c_{t-2} \rfloor + 1$ colors for the input graph G. Since A is a c-approximation for the minimum vertex coloring problem, $\chi(G) > c_{t-2}$. So OPT has to use more than c_{t-2} colors for G. Partition the vertex set V of G according to the coloring used by OPT, exactly as before. For $i = 0, 1, \ldots$, $V_i = \{v \mid c_{i-1} < color_{OPT}(v) \leq c_i\}$. Then since $\chi(G) > c_{t-2}$, $V_{t-1} \neq \emptyset$. As before, let W_i be the weight of color class $c_{i-1} + 1$ in OPT. Recall our assumption that OPT numbers color classes $1, 2, \ldots$ in non-increasing order of weight. Similarly, let R_i denote the weight of a heaviest color class created in round i, by `GeomFit`. Using the same reasoning as in Lemma 3.5.7, we obtain that $R_i \leq W_i$ for all $i = 0, 1, \ldots, t-1$. As before, a lower bound on OPT is

$$OPT \geq W_0 + W_1 + \sum_{i=0}^{t-3} c_i \cdot W_{i+2}.$$

As upper bound on $weight($`GeomFit`$)$ is

$$weight(GeomFit) \leq \sum_{i=0}^{t-1} \lfloor 2^i c \rfloor R_i.$$

It follows that $weight(GeomFit) \leq 4c \cdot OPT$ and we obtain the following theorem.

Theorem 3.5.9 *Let $\mathcal{G}$ be a hereditary class of graphs on which the minimum vertex coloring problem has a c-approximation algorithm. Algorithm* GeomFit *is a 4c-approximation algorithm for the max-coloring problem on $\mathcal{G}$.*

Randomized GeomFit

We now show how we can use randomization to improve the approximation ratio from 4 to 3. The only change is now to use a value q as the multiplier. Pick a value of α uniformly at random from $[1, q)$, and multiply this by α to get a new value for c_i. The algorithm is presented below.

Algorithm 11 RandGeomFit(G, w, α)

1:Pick α uniformly at random from $[1, q)$.
2:Let $i = 0$, $l_i = 0$
3:**while** $G \neq \phi$ **do**
  4:Set $c_i = \lfloor \alpha \cdot q^i \rfloor$
  5:Let $G_i = mkc(G, c \cdot c_i)$
  6:Color G_i optimally using colors $l_i + 1, \cdots, l_i + c \cdot c_i$
  7:Set $l_{i+1} = l_i + c_i$, $i = i + 1$.
  8:Set $G = G \setminus G_i$.
9:**end while**

The algorithm initially picks a value for α uniformly at random from $[1, q)$. Then, the algorithm proceeds in rounds, in each round setting $c_i = \lfloor \alpha \cdot q^i \rfloor$. Here, $q > 1$ is an integer that will be found later from the analysis. In each round, the algorithm calls $mkc(G, c \cdot c_i)$, which returns a maximal $c \cdot c_i$ colorable subgraph of G in non-increasing weight order. For perfect graphs, $c = 1$, and $mkc(G, c_i)$ can

be implemented to run in polynomial time as follows. Order the vertices in non-increasing weight order, and start with a set S, initially empty. When a vertex v is encountered, test whether the graph induced by $v \cup S$ has a $c_i + 1$ clique. If there is a $c_i + 1$ clique, don't add v to S, else add v to S. For perfect graphs, since $\chi(G) = \omega(G)$, this ensures that at the end of this iteration, S is c_i-colorable. We make at most n calls to a maximum clique computation algorithm, and hence mkc runs in polynomial time. For graphs that are not perfect, but for which we have a c-approximate coloring algorithm, we compute such a coloring for each $v \cup S$ instead of finding a maximum clique (If the algorithm is a c approximation algorithm, we know that the chromatic number of the graph returned by $mkc(G, c \cdot c_i)$ is at least c_i/c). We again make at most n calls to a polynomial time coloring algorithm and $mkc(G, c \cdot c_i)$ runs in polynomial time. Finally, since c_i increases geometrically, the number of rounds is at most $O(\log n)$. Hence, *RandGeomFit*(G, w, α) runs in polynomial time.

For a fixed $\alpha \in [1, q)$, let $t(\alpha)$ be the number of rounds used by *RandGeomFit* to color the graph. Let the rounds be numbered $0, 1, \cdots, t(\alpha) - 1$. The analysis proceeds by partitioning the color classes of OPT corresponding to the rounds of *RandGeomFit* and bounding the costs of each such partition. The following lemma follows just as in the earlier section.

Lemma 3.5.10 *If RandGeomFit uses $t(\alpha)$ rounds to color a graph G, then $\chi(G) > c_{t(\alpha)-2}$.*

Proof: In the penultimate round, namely round $t(\alpha) - 2$, mkc is given $c \cdot c_{t(\alpha)-2}$ colors to color G. If G cannot be completely colored with $c \cdot c_{t(\alpha)-2}$ colors, and since

we have a c-approximation for coloring G, it follows that $c \cdot c_{t(\alpha)-2} < c \cdot \chi(G)$, or $\chi(G) > c_{t(\alpha)-2}$. $\square$

Now, we partition the vertices of OPT to correspond to the color classes produced by *RandGeomFit* in each round. Let OPT use p colors, and let the color classes be labeled $C_1, C_2, \cdots, C_p$. Also assume that the color classes are labeled in non-increasing order of their weights. Thus $w(C_1) \geq w(C_2) \geq \cdots \geq w(C_p)$. For $1 \leq i \leq p$, let $X_i = w(C_i)$. Also, let $color(v)$ denote the color assigned to vertex v by OPT. Define

$$V_i(\alpha) = \{v \in V | c_{i-1}(\alpha) < color(v) \leq c_i(\alpha)\}, \text{ for each } i = 0, 1, \cdots, t(\alpha) - 1$$

For the above definition, we also assume that $c_{-1}(\alpha) = 0$. Let $W_i(\alpha) = max\{w(v)|v \in V_i(\alpha)\}$. Note that $W_i(\alpha) = X_{c_{i-1}(\alpha)+1}$, for each $i = 0, 1, \cdots, t(\alpha) - 1$. Let $R_i(\alpha)$ be the maximum weight of any color class created in the i^{th} round of *RandGeomFit*. Then,

Lemma 3.5.11 *For each* $i = 0, 1, \cdots, t(\alpha) - 1$, $R_i(\alpha) \leq W_i(\alpha)$.

Proof: For $i = 0$, the lemma clearly holds. $R_0(\alpha) = W_0(\alpha) = \max_{v \in V} w(v)$. Let v_i be the first vertex in non-increasing weight order not selected by *mkc* in round 0. By the greedy choice made by *mkc* in selecting subgraphs of G, it follows that $\{v_0, \cdots, v_{i-1}\}$ is maximal $c \cdot c_0$ colorable, and the chromatic number of the graph induced by the vertices $\{v_0, \cdots, v_i\}$ is greater than c_0. Thus, the first c_0 color classes of OPT cannot cover all vertices of weight larger than or equal to v_i. Hence the heaviest vertex in V_1

would have weight at least as large as R_1. By a similar argument, since $V_0 \cup \cdots \cup V_{i-1}$ is c_{i-1}-colorable, and the $i-1^{th}$ round uses $c \cdot c_{i-1}$ colors, it follows that $R_i \leq W_i$.

□

Before we state the main theorem, we prove a technical lemma that is useful in the proof.

Lemma 3.5.12 *Let $Y_1 \geq Y_2 \geq \ldots \geq Y_k$ and $a_1 \leq a_2 \leq \ldots \leq a_k$. Then,*

$$\sum_{i=1}^{k} a_i \cdot Y_i \leq \Big(\frac{\sum_{j=1}^{k} a_i}{k}\Big) \sum_{i=1}^{k} Y_i.$$

Proof: This can be proved by induction on k. For $k = 1$, the lemma clearly holds. Assume the lemma is true for all $k' \leq k$. Assume k is even. For odd k, we can add an extra term $Y_{k+1} = 0$ and get a slightly sharper inequality.

$$\begin{aligned}
\sum_{i=1}^{k} a_i \cdot Y_i &= \sum_{i=1}^{k/2} a_i \cdot Y_i + \sum_{i=k/2+1}^{k} a_i \cdot Y_i \\
&\leq \frac{\sum_{i=1}^{k/2} a_i}{k/2} \sum_{i=1}^{k/2} Y_i + \frac{\sum_{i=k/2+1}^{k} a_i}{k/2} \cdot \sum_{i=k/2+1}^{k} Y_i \\
&\leq \frac{\frac{\sum_{i=1}^{k/2} a_i}{k/2} + \frac{\sum_{i=k/2+1}^{k} a_i}{k/2}}{2} \Big(\sum_{i=1}^{k/2} Y_i + \sum_{i=k/2+1}^{k} Y_i \Big) \\
&= (\sum_{i=1}^{k} a_i)/k \cdot \Big(\sum_{i=1}^{k} Y_i \Big)
\end{aligned}$$

The first inequality follows from the inductive hypothesis. The second inequality again follows from the inductive hypothesis, setting $k = 2$. □

Now, we are ready to state the main theorem.

Theorem 3.5.13 *Let $\mathcal{G}$ be a hereditary class of graphs on which the minimum vertex coloring problem has a c-approximation algorithm. Then, algorithm RandGeomFit is a randomized 3c-approximation algorithm for the max-coloring problem on $\mathcal{G}$.*

Proof: Let $A(\alpha)$ denote the weight of the coloring returned by *RandGeomFit*. Then,

$$A(\alpha) \leq \sum_{i=0}^{t(\alpha)-1} R_i(\alpha) \cdot c \cdot c_i(\alpha),$$

where $R_i(\alpha)$ is the weight of the heaviest color class created in round i.

From Lemma 3.5.10 and 3.5.11, we get

$$A(\alpha) \leq c \cdot \sum_{i=0}^{t(\alpha)-1} W_i(\alpha) \cdot c_i(\alpha) \leq c \cdot \sum_{i=0}^{t(\alpha)-1} X_{c_{i-1}(\alpha)+1} \cdot c_i(\alpha).$$

This means that each term in $E[A(\alpha)]$ is of the form $X_{j+1} \cdot k$ times the probability that $c_{i-1}(\alpha) = j$ and $c_i(\alpha) = k$ for some i. Note that when $j = 0$, that is, when $c_{i-1}(\alpha) = 0$, it implies that $i = 0$. As a result, we get that $k = c_0(\alpha) = \lfloor \alpha \rfloor$. This means that depending on the value chosen for α, k can take on any integer value in $\{1, 2, \ldots, q-1\}$. Furthermore, it can take on each of these values with probability $1/(q-1)$. Therefore, in $E[A(\alpha)]$, the term X_1 appears as

$$X_1\Big(\frac{1}{(q-1)} + \frac{2}{(q-1)} + \cdots + \frac{(q-1)}{(q-1)}\Big) = X_1 \cdot \frac{q}{2}.$$

Now we consider terms in $E[A(\alpha)]$ containing $X_{j+1} \cdot k$ for $j = 1, 2, \ldots$. The

contribution of each such term to $E[A(\alpha)]$ is:

$$\begin{aligned} & X_{j+1} \cdot k \cdot \text{Prob}[(c_{i-1}(\alpha) = j) \wedge (c_i(\alpha) = k) \text{ for some i}] \\ \leq \; & X_{j+1} \cdot k \cdot \sum_i \text{Prob}[(c_{i-1}(\alpha) = j) \wedge (c_i(\alpha) = k)] \\ = \; & X_{j+1} \cdot k \cdot \sum_i \text{Prob}[(c_{i-1}(\alpha) = j] \cdot \text{Prob}[(c_i(\alpha) = k) \mid (c_{i-1}(\alpha) = j)] \end{aligned}$$

Assuming that $c_{i-1}(\alpha) = j$, we get $j \leq q^{i-1}\alpha < j+1$ and therefore $qj \leq q^i\alpha < qj+q$. This implies that

$$\text{Prob}[(c_i(\alpha) = k) \mid (c_{i-1}(\alpha) = j] = \frac{1}{q},$$

when $k \in \{qj, qj+1, qj+2, \ldots, qj+(q-1)\}$ (and this probability equals 0 for other values of k). Therefore, the contribution to $E[A(\alpha)]$ of all the terms containing X_{j+1} is

$$\begin{aligned} & \frac{1}{q} \cdot X_{j+1} \cdot \sum_i \text{Prob}[(c_{i-1}(\alpha) = j] \cdot \sum_{k=qj}^{qj+(q-1)} k \\ = \; & X_{j+1} \cdot \left(qj + \frac{q-1}{2}\right) \sum_i \text{Prob}[(c_{i-1}(\alpha) = j] \\ < \; & X_{j+1} \cdot q(j+1/2) \sum_i \text{Prob}[(c_{i-1}(\alpha) = j]. \end{aligned}$$

We now evaluate the probability $\sum_i \text{Prob}[(c_{i-1}(\alpha) = j]$.

$$\begin{aligned}\sum_i \text{Prob}[j = c_{i-1}(\alpha)] &= \sum_i \text{Prob}[j = \lfloor q^{i-1}\alpha \rfloor] \\ &= \sum_i \text{Prob}[j \leq q^{i-1}\alpha < (j+1)] \\ &= \sum_i \text{Prob}[\frac{j}{q^{i-1}} \leq \alpha < \frac{(j+1)}{q^{i-1}}]\end{aligned}$$

Given that $q > 1$ is an integer, there are three cases dealing with the relative values of j and q^i. These are:

(A) $q^{i-1} \leq j < j+1 \leq q^i$.

(B) $j < j+1 \leq q^{i-1}$.

(C) $q^i \leq j < j+1$.

In Case (B), $(j+1)/q^{i-1} \leq 1$ and since α is chosen from $[1, q)$, the probability in this case is 0. In Case (C), $j/q^{i-1} \geq q$ and since α is chosen from $[1, q)$, the probability in this case is also 0. This leaves Case (A) and there is exactly one value of i satisfying $q^{i-1} \leq j < j+1 \leq q^i$. Therefore we get that

$$\sum_i \text{Prob}[j = c_{i-1}(\alpha)] \leq \frac{1}{(q-1) \cdot q^{i-1}},$$

where i is the unique value satisfying $q^{i-1} \leq j < j+1 \leq q^i$. Thus we get that

$$E[A(\alpha)] \leq c \cdot X_1 \cdot \frac{q}{2} + c \cdot \frac{q}{(q-1)} \sum_{j=1}^{p-1} X_{j+1} \cdot \frac{(j+1/2)}{q^{i-1}} \tag{3.7}$$

where i is such that $q^{i-1} \leq j < j+1 \leq q^i$. Note that for all j satisfying $q^{i-1} \leq j \leq q^i - 1$, the same term $1/q^{i-1}$ occurs as a coefficient of $X_{j+1} \cdot (j+1/2)$ in the above summation. So this summation can be rewritten as

$$E[A(\alpha)] \leq c \cdot X_1 \cdot \frac{q}{2} + c \cdot \frac{q}{(q-1)} \sum_{i \geq 1} \frac{1}{q^{i-1}} \sum_{j=q^{i-1}}^{q^i-1} X_{j+1} \cdot (j+1/2). \tag{3.8}$$

Since we have that

$$X_{q^{i-1}+1} \geq X_{q^{i-1}+2} \geq \cdots \geq X_{q^i-1},$$

we can use Lemma 3.5.12 to simplify the summation. Applying the lemma, we get :

$$\sum_{j=q^{i-1}}^{q^i-1} X_{j+1} \cdot (j+1/2) \leq \frac{(q^{i-1}+q^i)}{2} \sum_{j=q^{i-1}}^{q^i-1} X_{j+1}.$$

Substituting this upper bound in Eqn (3.7) and letting OPT_i denote $\sum_{j=q^{i-1}}^{q^i-1} X_{j+1}$, we get

$$\begin{aligned} E[A(\alpha)] &\leq c \cdot X_1 \cdot \frac{q}{2} + c \cdot \frac{q}{(q-1)} \sum_{i \geq 1} \frac{(q^{i-1}+q^i)}{2q^{i-1}} \cdot OPT_i \\ &= c \cdot X_1 \cdot \frac{q}{2} + c \cdot \frac{q(q+1)}{2(q-1)} \cdot \sum_{i \geq 1} OPT_i \end{aligned}$$

To get a specific approximation ratio, let us set $q = 2$. Then,

$$E[A(\alpha)] \leq c \cdot X_1 + 3 \cdot c \cdot \sum_{i \geq 1} OPT_i \leq 3 \cdot c \cdot OPT.$$

□

Derandomization: The theorem above shows that there exists a value of $\alpha \in [1, q)$ such that *RandGeomFit* produces a solution that is at most $3c \cdot OPT$. For a graph that is k-colorable, there are at most $O(k)$ distinct values of α that results in a different coloring. Since the number of colors in an optimal max-coloring requires at most $O(n)$ colors, we can use at most $O(n)$ distinct values of α and pick the best solution found. This gives a deterministic $3c$-approximation algorithm for max-coloring.

Theorem 3.5.14 *For a hereditary class of graphs $\mathcal{G}$ that has a c-approximation algorithm for minimum vertex coloring, there is a deterministic $3c$-approximation algorithm for max-coloring on $\mathcal{G}$.*

3.6 Future Work

The constant factor approximation algorithms for max-coloring, namely *Geomfit* and *WtPartition* show that despite the *non-local* nature of the objective function. i.e., the weight of a color class is defined by a single heavy weight vertex, we can do almost as well as classical coloring by appropriately partitioning the graph and using a classical coloring algorithm for each partition. On the other hand, the only hardness of approximation result for max-coloring is the $8/7 - \epsilon$ hardness for bipartite graphs. Hence, it would be interesting to try to improve the approximation ratio for max-coloring on perfect graphs to below a factor of e, or obtain better hardness of approximation results for perfect graphs.

For interval graphs, the best approximation algorithm known is the factor 2

approximation guaranteed by BETTER-MCA. It would be interesting to show APX-hardness of max-coloring and obtain a better approximation ratio.

Halldòrsson, et al. [19] have recently studied the problem of batch scheduling conflicting conflicting jobs with the objective of minimizing the average completion time. They present algorithms with an approximation ratio of 4 for perfect graphs, again by partitioning the graph into weight classes and then applying a bin-packing algorithm for each weight class. Again, no hardness of approximation results are known for this problem on perfect graphs.

Several researchers [5, 6, 30, 41–43] have studied the problem of scheduling conflicting jobs with preemption. While the preemptive and batch scheduling versions of scheduling with conflicts seem to be well understood, the situation as regards non-preemptive scheduling is unclear. Halldorsson, et al. [41] present a PTAS for non-preemptive scheduling on planar graphs and partial k-trees. However, the best known approximations for non-preemptive scheduling are only $O(\log n)$ on perfect graphs.

Finally, in all our models, we have assumed that we have an arbitrarily large number of machines, while in many real applications, we only have a finite set of machines. For example, in the problem of scheduling jobs on a multiprocessor machine so that jobs requiring access to the same file are not scheduled simultaneously, we only have a fixed number of machines. It would be interesting to study the max-coloring problem, and other scheduling problems with conflicts when we have only a fixed number of machines.

CHAPTER 4
EXPERIMENTAL EVALUATION

4.1 Introduction

The max-coloring and interval coloring problems, as described in Chapter 3 were motivated by problems of buffer allocation and storage allocation for computer programs. In Chapter 3, we developed a set of approximation algorithms for the max-coloring problem on various classes of graphs. In this chapter, we experimentally evaluate some of the constant-factor approximation algorithms in the previous chapter as well as two simple heuristics for both problems on chordal graphs. Our main result is that the *Geomfit* algorithm, described in Chapter 3 is not only a good approximation algorithm in theory, it also works extremely well in practice. It works better than the first-fit algorithm for the max-coloring problem, while a small modification of *Geomfit* works better than the other heuristics for the interval coloring problem. The *WtPartition* algorithm however, works poorly compared to all other algorithms, despite the constant-factor guarantee. We start with the definitions of the max-coloring and interval coloring problems again.

Interval coloring. Given a graph $G = (V, E)$ and positive integral vertex weights $w : V \rightarrow \mathbf{N}$, the *interval coloring* problem seeks to find an assignment of an interval $I(u)$ to each vertex $u \in V$ such that two constraints are satisfied: (i) for every vertex $u \in V$, $|I(u)| = w(u)$ and (ii) for every pair of adjacent vertices u and v, $I(u) \cap I(v) = \emptyset$. The goal is to minimize the span $|\cup_v I(v)|$.

Max-coloring. Like interval coloring, the *max-coloring problem* takes as input a vertex-weighted graph $G = (V, E)$ with weight function $w : V \to \mathbf{N}$. The problem requires that we find a proper vertex coloring of G whose color classes $C_1, C_2, \ldots, C_k$, minimize the sum of the weights of the heaviest vertices in the color classes, that is, $\sum_{i=1}^{k} max_{v \in C_i} w(v)$.

Connections between interval coloring and max-coloring. Given a coloring of a vertex weighted graph $G = (V, E)$ with color classes $C_1, C_2, \ldots, C_k$, we can construct an assignment of intervals to the vertices as follows. For each i, $1 \leq i \leq k$, let $v_i \in C_i$ be the vertex with maximum weight in C_i. Let $H(1) = 0$, and for each i, $2 \leq i \leq k$, let $H(i) = \sum_{j=1}^{i-1} w(v_j)$. For each vertex $v \in C_i$, we set $I(v) = (H(i), H(i) + w(v))$. Clearly, no two vertices in distinct color classes have overlapping intervals and therefore this is a valid interval coloring of G. We say that this is the interval coloring *induced* by the coloring $C_1, C_2, \ldots, C_k$. The span of this interval coloring is $\sum_{i=1}^{k} w(v_i)$, which is the same as the weight of the coloring $C_1, C_2, \ldots, C_k$ viewed as a max-coloring. In other words, if there is a max-coloring of weight W for a vertex weighted graph G, then there is an interval coloring of G of the same weight.

However, in Chapter 3 we show an instance of a vertex weighted interval graph on n vertices for which the weight of an optimal max-coloring is $\Omega(\log n)$ times the weight of the heaviest clique. This translates into an $\Omega(\log n)$ gap between the weight of an optimal max-coloring and the span of an optimal interval coloring because an optimal interval coloring of an interval graph has span that is within $O(1)$ of the weight of a heaviest clique. In general, algorithms for max-coloring can be used for interval

coloring with minor modifications to make the interval assignment more "compact". While the worst case performance of these algorithms can be bad for interval coloring, our experiments show that in practice, algorithms that do well for max-coloring do well for interval coloring as well. For example, the performance of *GeomFit* for the interval coloring provides strong evidence of this phenomenon. These connections motivate us to study interval coloring and max-coloring in the same framework.

Chordal graphs. For both the interval coloring and max-coloring problems, the assumption that the underlying graph is an interval graph is somewhat restrictive. As mentioned before, in memory allocation applications, if the underlying program is straight-line, then the corresponding conflicts can be modeled as an interval graph. However, most programs contain conditional statements and loops. We consider a natural generalization of interval graphs called chordal graphs. A graph is a *chordal graph* if it has no induced cycles of length 4 or more. Alternately, every cycle of length 4 or more in a chordal graph has a chord.

There are many alternate characterizations of chordal graphs. One that will be useful for our purposes is the existence of a perfect elimination ordering of the vertices of any chordal graph. An ordering $v_n, v_{n-1}, \ldots, v_1$ of the vertex set of a graph is said to be a *perfect elimination ordering* if when vertices are deleted in this order, for each i, the neighbors of vertex v_i in the remaining graph, $G[\{v_1, v_2, \ldots, v_i\}]$ form a clique. A graph is a chordal graph iff it has a perfect elimination ordering. Tarjan and Yannakakis [67] describe a simple linear-time algorithm called *maximum cardinality search* that can be used to determine if a given graph has a perfect elimination

ordering and to construct such an ordering if it exists. Given a perfect elimination ordering of a graph G, the graph can be colored by considering vertices in reverse perfect elimination order and assigning to each vertex the minimum available color. It is easy to see that this greedy coloring algorithm uses exactly as many colors as the size of the largest clique in the graph and therefore produces an optimal vertex coloring.

Every interval graph is also a chordal graph (but not vice versa). To see this, take an interval representation of an interval graph and order the intervals in left-to-right order of their left endpoints. It is easy to verify that this gives a perfect elimination ordering of the interval graph. Thus chordal graphs generalize interval graphs and one of our motivations in considering chordal graphs is to determine if the constant-factor algorithms for interval coloring interval graphs can be extended to chordal graphs. Another motivation for considering chordal graphs is that the way certain kinds of compilers such as algebraic compilers process source code, the interference graph of source objects ends up being a chordal graph [65]. A final motivation is that others have considered the problem of finding approximation algorithms for interval coloring chordal graphs, but with limited success. For example, [16] shows a 2-approximation algorithm for the interval coloring problem on claw-free chordal graphs, leaving the problem open for chordal graphs in general.

In this chapter, we consider four simple heuristics and evaluate their performance on chordal graphs for both the max-coloring and interval coloring problems.

These heuristics are:

- **Chromatic Partitioning (*Geomfit*).** Vertices are colored in rounds. In each round, a subgraph of the graph is chosen and colored optimally. The subgraph in each round is chosen in non-increasing weight order, such that its chromatic number is determined by the round number.

- **First fit.** Vertices are considered in decreasing order of weight and each vertex is assigned the first available color or interval.

- **Best fit.** Vertices are considered in reverse perfect elimination order and each vertex is assigned the color class or interval it "fits" in best.

- **Weight partitioning (*WtPartition*).** Vertices are partitioned into groups where each group consists of vertices of similar weight. Each subgraph induced by vertices of a group are colored optimally. The interval assignment induced by this coloring is returned as the solution to the interval coloring problem.

In Chapter 3, we showed that *GeomFit* and *WtPartition* are both 4-approximation algorithms for max-coloring on perfect graphs. *GeomFit* as is, is known to do badly for interval coloring, since OPT for max-coloring can be much larger than OPT for interval coloring. *First fit* and *Best-fit* are fairly standard heuristics for many resource allocation problems and have been analyzed extensively for problems such as the bin packing problem. Using old results and a few new observations, we point out that *First-fit* provides an $O(\log n)$ approximation guarantee. *Best-fit* provides no such guarantee and we provide an example of a vertex weighted interval graph for which *Best-fit* returns a solution to the max-coloring problem whose weight is $\Omega(\sqrt{n})$ times

the weight of the optimal solution.

Our experiments show that in general *GeomFit* performs better than the rest of the heuristics and is typically very close to OPT, deviating by about 1.5% on average for max-coloring as well as for interval coloring. *First-fit* also performs well on average, deviating from OPT by about 6% for both problems. Best-fit comes third and *WtPartition* performs significantly worse than the other heuristics. Our basic data comes from about 10000 runs of each of the three heuristics for each of the two problems on randomly generated chordal graphs of various sizes, sparsity, and structure.

Our experiments also reveal that *Best-fit* performs better on chordal graphs that are "irregular". Here, "regularity" refers to the variance in the sizes of maximal cliques – the greater this variance, the more irregular the graph.

4.2 The Algorithms

In this section we describe four simple algorithms for the interval coloring and max-coloring problems.

4.2.1 Algorithm 1: Chromatic Partitioning(*Geomfit*)

For the max-coloring problem, we described and analyzed *GeomFit* in Chapter 3. For the interval coloring problem, *GeomFit* works as follows. This algorithm constructs a coloring in rounds, using a fresh set of colors in each round. In round i, a maximal c_i colorable subgraph of the graph is chosen in non-increasing weight order, and colored with the fewest possible colors. For the interval coloring problem, we

return the solution induced by *Geomfit* for max-coloring. i.e., if a vertex v is assigned a color i by *Geomfit*, then we assign the interval $(\sum_{j=1}^{i-1} w(C_j), \sum_{j=1}^{i-1} w(C_j) + w(v))$ to v. We denote *GeomFit* for max-coloring and interval coloring by `GFM` and `GFI` respectively.

4.2.2 Algorithm 2: First-fit in weight order

For the interval coloring problem, we preprocess the vertices and "round up" their weights to the nearest power of 2. Then, for both problems we order the vertices of the graph in non-increasing order of weights. Let $v_1, v_2, \ldots, v_n$ be this ordering. We process vertices in this order and use a "first-fit heuristic" to assign intervals and colors to vertices to solve the interval coloring and max-coloring problem respectively. We round up the weights to ensure an $O(\log n)$ approximation guarantee for interval coloring, as described in Theorem 4.2.1.

The algorithm for interval coloring is as follows. To each vertex we assign a real interval with non-negative endpoints. To vertex v_1, we assign $(0, w(v_1))$. When we get to vertex v_i, $i > 1$, each vertex v_j, $1 \leq j \leq i-1$ has been assigned an interval $I(v_j)$. Let U_i be the union of the intervals already assigned to neighbors of v_i. Then $(0, \infty) - U_i$ is a non-empty collection of disjoint intervals. Because the weights are powers of 2 and vertices are considered in non-increasing order of weights, every interval in $(0, \infty) - U_i$ has length at least $w(v_i)$. Of these, pick an interval $I = (a, b)$ with smallest right endpoint and assign the interval $(a, a + w(v_i))$ to v_i. This is $I(v_i)$.

For a solution to the max-coloring problem, we assume that the colors to be assigned to vertices are natural numbers, and assign to each vertex v_i the smallest

color not already assigned to a neighbor of v_i. We denote the two algorithms described above by `FFI` (short for first-fit for interval coloring) and `FFM` (short for first-fit for max-coloring) respectively.

We now observe that both algorithms provide an $O(\log(n))$-approximation guarantee. The following result is a generalization of the result from [14].

Theorem 4.2.1 *Let C be a class of graphs that is closed under duplication of vertices and suppose there is a function $\alpha(n)$ such that the first-fit on-line graph coloring algorithm colors any n-vertex graph G in C with at most $\alpha(n) \cdot \chi(G)$ colors. Then, for any n-vertex graph G in C the* `FFI` *algorithm produces a solution with span at most $2\alpha(n) \cdot OPT_I(G)$, where $OPT_I(G)$ is the optimal span of any feasible assignment of intervals to vertices.*

Note that perfect graphs are closed under vertex duplication [38], and since chordal graphs are a sub-class of perfect graphs, Theorem 4.2.1 holds for chordal graphs.

Irani [46] has shown that the first-fit graph coloring algorithm uses at most $O(\log(n)) \cdot \chi(G)$ colors for any n-vertex chordal graph G. This fact together with Theorem 4.2.1 implies that `FFI` provides an $O(\log n)$ approximation guarantee for interval coloring. Recall from Chapter 3, Theorem 3.3.1, that if a hereditary class of graphs $\mathcal{G}$, has an α-competitive algorithm for online coloring, then there exists an α-approximation for interval coloring on this class of graphs. Using this theorem along with the theorem of Irani on the performance of *First-fit* for online coloring chordal graphs shows that `FFM` is an $O(\log n)$-approximation algorithm for max-coloring chordal graphs.

An example that is tight for both algorithms is easy to construct. Let $T_0, T_1, T_2, \ldots$ be a sequence of trees where T_0 is a single vertex and T_i, $i > 0$, is constructed from T_{i-1} as follows. Let $V(T_{i-1}) = \{u_1, u_2, \ldots, u_k\}$. To construct T_i, start with T_{i-1} and add vertices $\{v_1, v_2, \ldots, v_k\}$ and edges $\{u_i, v_i\}$ for all $i = 1, 2, \ldots, k$. Thus the leaves of T_i are $\{v_1, v_2, \ldots, v_k\}$ and every other vertex in T_i has a neighbor v_j for some j. Now consider a tree T_n in this sequence. Clearly, $|V(T_n)| = 2^n$. Assign to each vertex in T_n a unit weight. To construct an ordering on the vertices of T_n first delete the leaves of T_n. This leaves the tree T_{n-1}. Recursively construct the ordering on vertices of T_{n-1}, and prepend to this the leaves of T_n in some order. It is easy to see that first-fit coloring algorithm that considers the vertices of T_n in this order uses n colors. As a result, both `FFI` and `FFM` have weight n, whereas OPT in both cases is 2. See Figure 4.1 for T_0, T_1, T_2, and T_3.

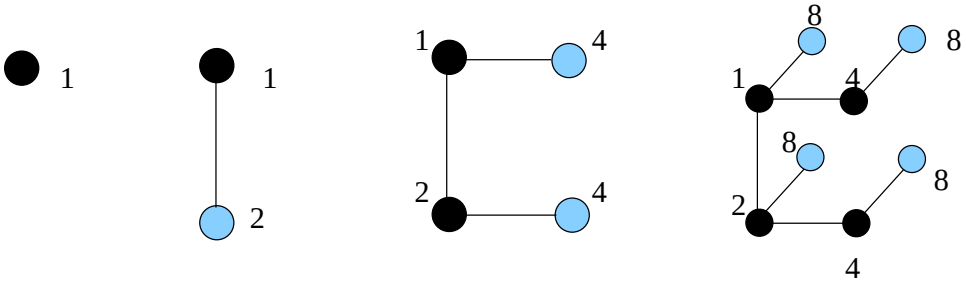

Figure 4.1: The family of tight examples for `FFI` and `FFM`.

4.2.3 Algorithm 3: Best-fit in reverse perfect elimination order

The third of the algorithms that we experiment with are obtained by considering vertices in reverse perfect elimination order and using a "best-fit" heuristic to assign intervals or colors. Let $v_1, v_2, \ldots, v_n$ be the reverse of a perfect elimination ordering of the vertices of G. Recall that if vertices are considered in reverse perfect elimination order and colored, using the smallest color at each step, we get an optimal coloring of the given chordal graph. This essentially implies that the example of a tree with unit weights that forced `FFI` and `FFM` into worst case behavior will not be an obstacle for this pair of algorithms.

The algorithm for interval coloring is as follows. As before, to each vertex we assign a real interval with non-negative endpoints and to vertex v_1, we assign $(0, w(v_1))$. When we get to vertex v_i, $i > 1$, each vertex v_j, $1 \leq j \leq i-1$ has been assigned an interval $I(v_j)$. Let $M = |\cup_{j=1}^{i-1} I(v_j)|$ and let U_i be the union of the intervals $I(v_j)$, where $1 \leq j \leq i-1$ and v_j is a neighbor of v_i. If $U_i = (0, M)$, then v_i is assigned the interval $(M, M + w(v_i))$. Otherwise, if $U_i \neq (0, M)$, then $(0, M) - U_i$ is a non-empty collection of disjoint intervals. However, since the vertices were not processed in weight order, we are no longer guaranteed that there is any interval in $(0, M) - U_i$ with length at least $w(v_i)$. There are two cases.

Case 1. If there is an interval in $(0, M) - U_i$ of length at least $w(v_i)$, then pick an interval $I \in (0, M) - U_i$ of smallest length such that $|I| \geq w(v_i)$. Suppose $I = (a, b)$. Then assign the interval $(a, a + w(v_i))$to v_i.

Case 2. Otherwise, if all intervals in $(0, M) - U_i$ have length less than $w(v_i)$, pick the

largest interval $I = (a, b)$ in $(0, M) - U_i$ (breaking ties arbitrarily) and assign $(a, a + w(v_i))$ to v_i. Note that this assignment of an interval to v_i causes the interval assignment to become infeasible. This is because there is some neighbor of v_i that has been assigned an interval with left endpoint b and $(a, a + w(v_i))$ intersects this interval. To restore feasibility, we lift all intervals "above" b by $\Delta = (a+w(v_i))-b$. In other words, for every vertex v_j, $1 \leq j \leq i$, $I(v_j) = (c, d)$, if $c \geq b$, then set $I(v_j) = (c + \Delta, d + \Delta)$. It is easy to see that this restores feasibility to the interval assignment.

Best-fit tries to minimize the increase in span of the intervals at each instance. Consider the chordal graph shown in Figure 4.2. The numbers next to vertices are vertex weights and the letters are vertex labels. The ordering of vertices A, B, C, D, E is a reverse perfect elimination ordering. By the time we get to processing vertex E, the assignment of intervals to vertices is as shown in the middle in Figure 4.2. When E is processed, we look for "space" to fit it in and find the interval $(10, 15)$, which is not large enough for E. So we move the interval $I(D)$ up by 5 units to make space for $I(E)$ and obtain the assignment shown on the right.

A similar "best-fit" solution to the max-coloring problem is obtained as follows. Let k be the size of a maximum clique in G. Start with a palette of colors $C = \{1, 2, \ldots, k\}$ and an assignment of color 1 to vertex v_1. Let $AC(v_i) \subseteq C$ be the colors available for v_i. For each color j, let W_j denote the maximum weight among all vertices colored j; for an empty color class j, $W_j = 0$. Color vertex v_i with a color $j \in AC$ that maximizes W_j, with ties broken arbitrarily. This ensures that the color

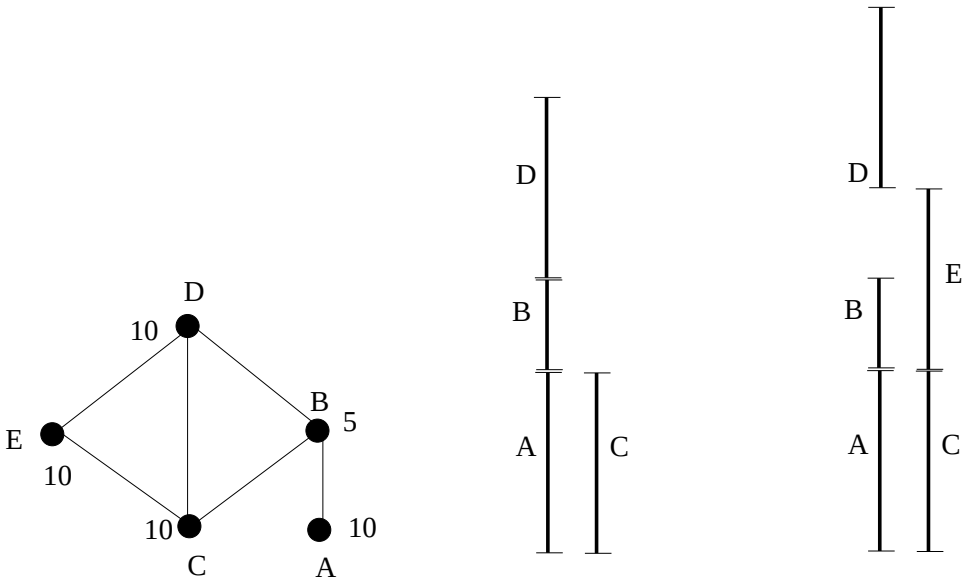

Figure 4.2: The best-fit heuristic in action for interval coloring.

we assign to v_i minimizes the increase in the weight of the coloring.

We will call these "best-fit" algorithms for interval coloring and max-coloring, `BFI` and `BFM` respectively.

An example that forces the best-fit algorithms, `BFI` and `BFM` to perform badly is the following. Consider a graph G with m disjoint cliques, each clique containing m vertices. Let the cliques be labeled $Q_1, Q_2, \ldots, Q_m$. For each i, $1 \leq i \leq m$, the distribution of weights of vertices in Q_i is as follows: there are $(i-1)$ vertices with weight 2, one vertex with weight W, and $(m-i)$ vertices with weight 1. Any ordering of vertices is a perfect elimination ordering of G. So suppose that `BFM` processes vertices in the following order: vertices of Q_1, followed by vertices of Q_2, followed by vertices of Q_3, etc. The vertices of each Q_i are ordered as follows: vertices with

weight 2 come first, followed by the vertex of weight W, followed by the vertices of weight 1. It is easy to check that if vertices are processed in this order then `BFM` will produce a coloring with m color classes, such that each color class contains a vertex of weight W. This solution has weight $m \cdot W$ as compared to OPT which has weight $W + 2(m-1)$ and thus this is an example that forces `BFM` to produce a solution at least $\Omega(m)$ times OPT. This is $\Omega(\sqrt{n})$, since the number of vertices $n = m^2$. In Figure 4.3, this example is shown as a set of intervals with $m = 4$. The intervals correspond to vertices and pairwise intersection of intervals corresponds to edges. Each row of intervals corresponds to a color class chosen by `BFM`. The optimal coloring in this instance would put the intervals with weight W in one row.

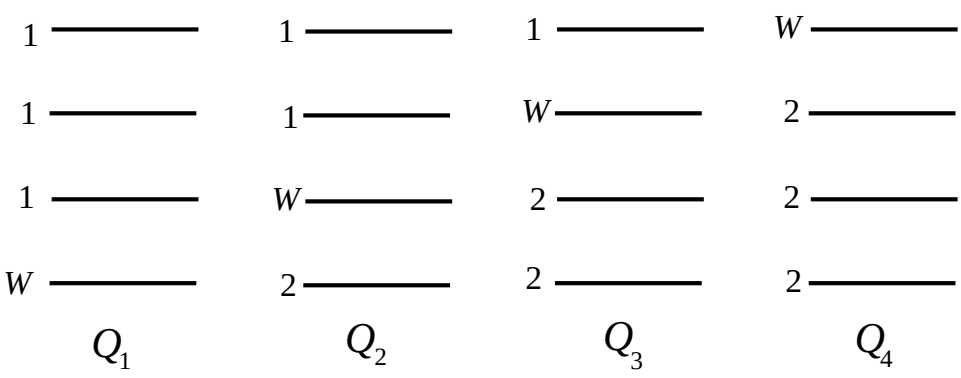

Figure 4.3: A bad example for the best-fit heuristic.

4.2.4 Algorithm 4: Weight Partitioning (*WtPartition*)

Another pair of algorithms for interval coloring and max-coloring can be obtained by partitioning the vertices of the given graph into groups with similar weight. Let W be the maximum vertex weight. Fix an integer $k \geq 1$ and partition the range $[1, W]$ into $(k+1)$ subranges:

$$\Big[1, \frac{W}{2^k}\Big], \Big(\frac{W}{2^k}, \frac{W}{2^{k-1}}\Big], \ldots, \Big(\frac{W}{2^2}, \frac{W}{2}\Big] \Big(\frac{W}{2}, W\Big].$$

For i, $1 \leq i \leq k$, let $R_i = (W/2^i, W/2^{i-1}]$ and let $R_{k+1} = [1, W/2^k]$. Partition the vertex set V into subsets V_i, $1 \leq i \leq (k+1)$ defined as $V_i = \{v \in V \mid w(v) \in R_i\}$. For each i, $1 \leq i \leq (k+1)$ let G_i be the induced subgraph $G[V_i]$. We ignore the weights and color each subgraph G_i with the fewest number of colors, using a fresh palette of colors for each subgraph G_i. For the max-coloring problem, we simply use this coloring as the solution.

For the interval coloring problem, we turn the coloring into an assignment of intervals to vertices as follows. Let $Q_1, Q_2, \ldots, Q_t$ be color classes produced by the above algorithm. For each i, $1 \leq i \leq t$, let W_i be the maximum vertex weight of a vertex in Q_i. Let $W_0 = 0$. For each $v \in C_i$, we assign the interval $\Big(\sum_{s=0}^{i-1} W_s, \sum_{s=0}^{i-1} W_s + w(v)\Big)$. This is an interval coloring of G because vertices in distinct color classes are assigned disjoint intervals. The span of this interval assignment is $\sum_{s=1}^{t} W_i$. This is identical to the total weight of the solution to max-coloring as well.

We will call these graph partitioning based algorithms for interval coloring

and max-coloring, `WPI` and `WPM` respectively. In Chapter 3 we showed that `WPM` is a 4-approximation algorithm for max-coloring. We now show that `WPI` is an $O(\log n)$ approximation algorithm for interval coloring

Theorem 4.2.2 *If we set $k = 2\log(n)$, then* `WPI` *produces a $(4 \cdot \log(n) + o(1))$-approximation to the interval coloring problem on perfect graphs.*

Proof: We show the proof in two stages. For i, $1 \leq i \leq k$, let α_i be the weight of the heaviest clique in $G[V_i]$. Let $\chi_i = \chi(G[V_i])$. Clearly, $\alpha_i \geq \chi_i \cdot W/2^i$. Let OPT refer to the weight of an optimal interval coloring and let OPT_i refer to the weight of an optimal interval coloring restricted to vertices in V_i. Note that $OPT_i \geq \alpha_i$. Since `WPI` colors each V_i with exactly χ_i colors and since the weight of each vertex in V_i is at most $W/2^{i-1}$, the weight of the coloring that `WPI` assigns to V_i is at most $\chi_i \cdot W/2^{i-1} \leq 2 \cdot \alpha_i \leq 2 \cdot OPT_i$. Since `WPI` uses a fresh palette of colors for each V_i, the weight of the coloring of $\cup_{i=1}^{k} V_i$ is at most

$$2 \cdot \sum_{i=1}^{k} OPT_i \leq 2 \cdot \sum_{i=1}^{k} OPT = 4\log(n) \cdot OPT.$$

Since $k = 2\log(n)$, $W/2^k = W/n^2$. Therefore, any coloring of V_{k+1} adds a weight of at most W/n to the coloring of the rest of the graph. Since $W \leq OPT$, `WPI` colors the entire graph with weight at most $(4 \cdot \log(n) + 1/n)OPT$.

□

4.3 Overview of the Experiments

4.3.1 How chordal graphs are generated

We have implemented an algorithm that takes in parameters n (a positive integer) and α (a real number in $[0, 1]$) and generates a random chordal graph with n vertices, whose sparsity is characterized by α. The smaller the value of α the more sparse the graph. In addition, the algorithm can run in two modes; in Mode 1 it generates somewhat "regular" chordal graphs and in Mode 2 it generates somewhat "irregular" chordal graphs.

The algorithm generates chordal graphs with $n, (n-1), \ldots, 2, 1$ as a perfect elimination ordering. In the ith iteration of the algorithm vertex i is connected to some subset of the vertices in $\{1, 2, \ldots, i-1\}$. Let G_{i-1} be the graph containing vertices $1, 2, \ldots, (i-1)$, generated after iteration $(i-1)$. Let $\{C_1, C_2, \ldots, C_t\}$ be the set of maximal cliques in G_{i-1}. It is well known that any chordal graph on n vertices has at most n maximal cliques. So we explicitly maintain the list of maximal cliques in G_{i-1}. We pick a maximal clique C_j and a random subset $S \subseteq C_j$ and connect i to the vertices in S. This ensures that the neighbors of i in $\{1, 2, \ldots, i-1\}$ form a clique, thereby ensuring that $n, (n-1), \ldots, 2, 1$ is a perfect elimination ordering.

We use the parameter α in order to pick the random subset S. For each $v \in C_j$, we independently add v to set S with probability α. This makes the expected size of S equal $\alpha \cdot |C_j|$. The algorithm also has a choice to make on how to pick C_j. One approach is to choose C_j uniformly at random from the set $\{C_1, C_2, \ldots, C_t\}$. This is Mode 1 and it leads to "regular" random chordal graphs, that is, random

chordal graphs in which the sizes of maximal cliques show small variance. Another approach is to choose a maximal clique with largest size from among $\{C_1, C_2, \ldots, C_t\}$. This is Mode 2 and it leads to more "irregular" random chordal graphs, that is, random chordal graphs in which there are a small number of very large maximal cliques and many very small maximal cliques. Graphs generated in the two modes seem to be structurally quite different. This is illustrated in Table 4.3.2, where we show information associated with 10 instances of graphs with $n = 250$ and $\alpha = 0.9$ generated in Mode 1 and in Mode 2. Each column corresponds to one of the 10 instances and comparing corresponding Mode 1 and Mode 2 rows easily reveals the fairly dramatic difference in these graphs. For example, the mean clique size in Mode 1 is about 8.5, while it is about 22 in Mode 2. Even more dramatic is the large difference in the variance of the clique sizes and this justifies our earlier observation that Mode 2 chordal graphs tend to have a few large cliques and many very small cliques, relative to Mode 1 chordal graphs.

4.3.2 How Weights are Assigned

Once we have generated a chordal graph G we assign weights to the vertices as follows. This process is parameterized by W, the maximum possible weight of a vertex. Let k be the chromatic number of G and let $\{C_1, C_2, \ldots, C_k\}$ be a k-coloring of G. Since G is a chordal graph, it contains a clique of size k. Let $Q = \{v_1, v_2, \ldots, v_k\}$ be a clique in G with $v_i \in C_i$. For each v_i, pick $w(v_i)$ uniformly at random from the set of integers $\{1, 2, \ldots, W\}$. Thus the weight of Q is $\sum_{i=1}^{k} w(v_i)$. For each vertex $v \in C_i - \{v_i\}$, pick $w(v)$ uniformly at random from $\{1, 2, \ldots, w(v_i)\}$. This ensures

that $\{C_1, C_2, \ldots, C_k\}$ is a solution to max-coloring with weight $\sum_{i=1}^{k} w(v_i)$ and the interval assignment induced by this coloring is an interval coloring of span $\sum_{i=1}^{k} w(v_i)$. Since $\sum_{i=1}^{k} w(v_i)$ is also the weight of the clique Q, which is a lower bound on OPT in both cases, we have that OPT $= \sum_{i=1}^{k} w(v_i)$ in both cases. The advantage of this method of assigning weights is that it is simple and gives us the value of OPT for both problems. The disadvantage is that, in general OPT for both problems can be strictly larger than the weight of the heaviest clique and thus by generating only those instances for which OPT equals the weight of the heaviest clique, we might be missing a rich class of problem instances. So we additionally tested our algorithms on instances of chordal graphs for which the weights were assigned at random. For these algorithms, we use the maximum weighted clique as a lower bound for OPT and as a consequence, the deviations reported for the algorithms are an overestimate of the real deviations.

4.3.3 Main Observations

For our main experiment we generated instances of random chordal graphs with number of vertices $n = 10, 20, 30, \ldots, 550$. For each value of n, we used values of $\alpha = 0.1, 0.2, \ldots, 0.9$. For each of the 55×9 (n, α) pairs, we generated 10 random vertex weighted chordal graphs in Mode 1 and 10 random vertex weighted chordal graphs in Mode 2. The vertex weights are assigned as described above, with the maximum weight W fixed at 1000. We ran each of the four heuristics for each of the two problems and averaged the weight and span of the solutions over the 10 instances for each (n, α) pairs separately for Mode 1 and Mode 2 graphs. Thus each

	MODE 1									
No. of maximal cliques	149	126	110	126	147	116	119	119	128	149
Size of largest clique	13	14	12	12	14	11	12	12	12	14
Size of smallest clique	4	3	5	3	5	4	4	4	3	5
Mean clique size	8.58	7.35	8.35	7.83	9.41	7.51	7.10	7.31	7.98	9.53
Variance	4.06	3.83	3.23	2.35	3.32	1.99	2.36	2.82	3.61	3.23
	MODE 2									
No. of maximal cliques	220	216	216	218	218	219	213	216	219	219
Size of largest clique	29	33	33	31	14	31	30	36	30	30
Size of smallest clique	5	3	5	7	4	5	7	5	4	4
Mean clique size	20.13	22.34	22.37	24.00	21.15	21.83	25.17	23.70	22.80	20.89
Variance	28.43	30.02	30.69	29.55	33.98	31.73	48.72	32.66	25.81	29.68

Table 4.1: Properties of 20 instances of graphs with $n = 250$ and $\alpha = 0.9$.Ten of these were generated in Mode 1 and the other ten in Mode 2.

heuristic was evaluated on 4950 instances of each Mode, for each problem. We then generated the same number of instances, with n, α, and the modes varying just as before, but this time assigning to each vertex, a weight chosen uniformly at random from $[0, 1000]$. We repeated all eight algorithms on these random instances, and used the maximum weighted clique as a lower bound for OPT.

For the max-coloring problem on interval graphs, the gap between the maximum-weight clique and OPT can be unbounded. We can construct examples where the gap is as large as $\Omega(\log n)$. For the interval coloring problem, the gap between OPT and the maximum-weight clique is known to be a constant, while for chordal graphs it is unknown. This gap may affect the reported deviations in ways that are hard to determine.

4.3.3.1 Max-coloring

The data for the max-coloring problem is presented in the following tables[1] and graphs. Table 2 summarizes the performance of the four heuristics for the max-coloring problem for both Mode 1 and Mode 2 chordal graphs. Graphs showing the performance of the four heuristics for the max-coloring problem on Mode 1 and Mode 2 chordal graphs are shown in Figures 4.4 and 4.5.

Based on our results we make the following observations regarding the max-coloring problem.

---

[1]All the raw data and code for the experiments is available at http://www.cs.uiowa.edu/∼rraman/chordalExp.html

	MODE 1				MODE 2			
	BFM	FFM	WPM	GFM	BFM	FFM	WPM	GFM
Equals OPT	936	3170	0	3580	3693	3451	0	3820
Equals χ	4950	3539	0	4019	4950	3451	0	3820
% Deviation	14.40	1.627	58.26	1.31	2.45	1.94	29.99	1.53
% Max Deviation	157.12	30.38	127.67	30.38	54.99	28.88	82.78	28.88
Equals LB (R)	90	111	0	121	101	111	0	121
Equals χ(R)	4950	2712	20	3232	4950	588	0	707
% Deviation (R)	24.79	17.08	74.69	16.08	24.90	12.88	44.08	12.74
% Max Deviation (R)	74.88	62.52	129.8	47.03	141.16	63.29	122.9	63.29

Table 4.2: This table shows aggregate performance over all 4950 runs of the four heuristics for max-coloring, separately for Mode 1 and Mode 2 graphs. The first four rows show the case when the weights are chosen so that OPT equals the weight of the maximum weight clique. The next four rows show the case when the weights are chosen randomly. In this case, the algorithms are compared against the weight of the maximum weight clique. The row "Equals OPT" lists the number of times each heuristic produces a coloring with weight equal to OPT, the row "Equals χ" lists the number of times each heuristic produces a coloring using minimum number of colors, and the row "Deviation" lists the percentage deviation of the weight of the solution produced from OPT or the lower bound on OPT, averaged over the 4950 runs. The row "Max Deviation" lists the maximum percentage deviation of the four algorithms from OPT or the lower bound.

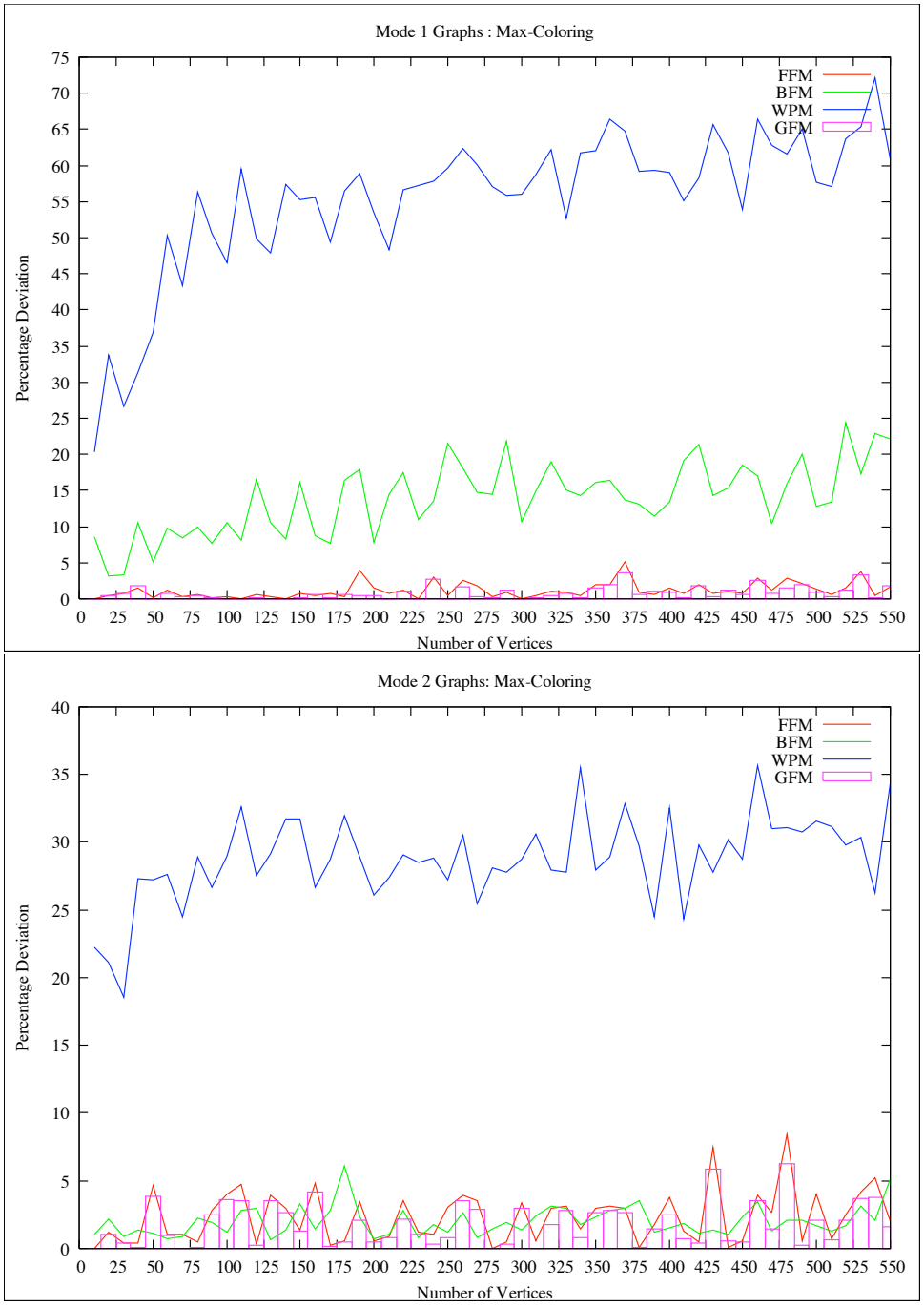

Figure 4.4: Graphs showing values for max-coloring Mode 1 and Mode 2 chordal graphs. The x-axis corresponds to the number of vertices in the graph, and the y-axis corresponds to the percentage deviation from OPT.

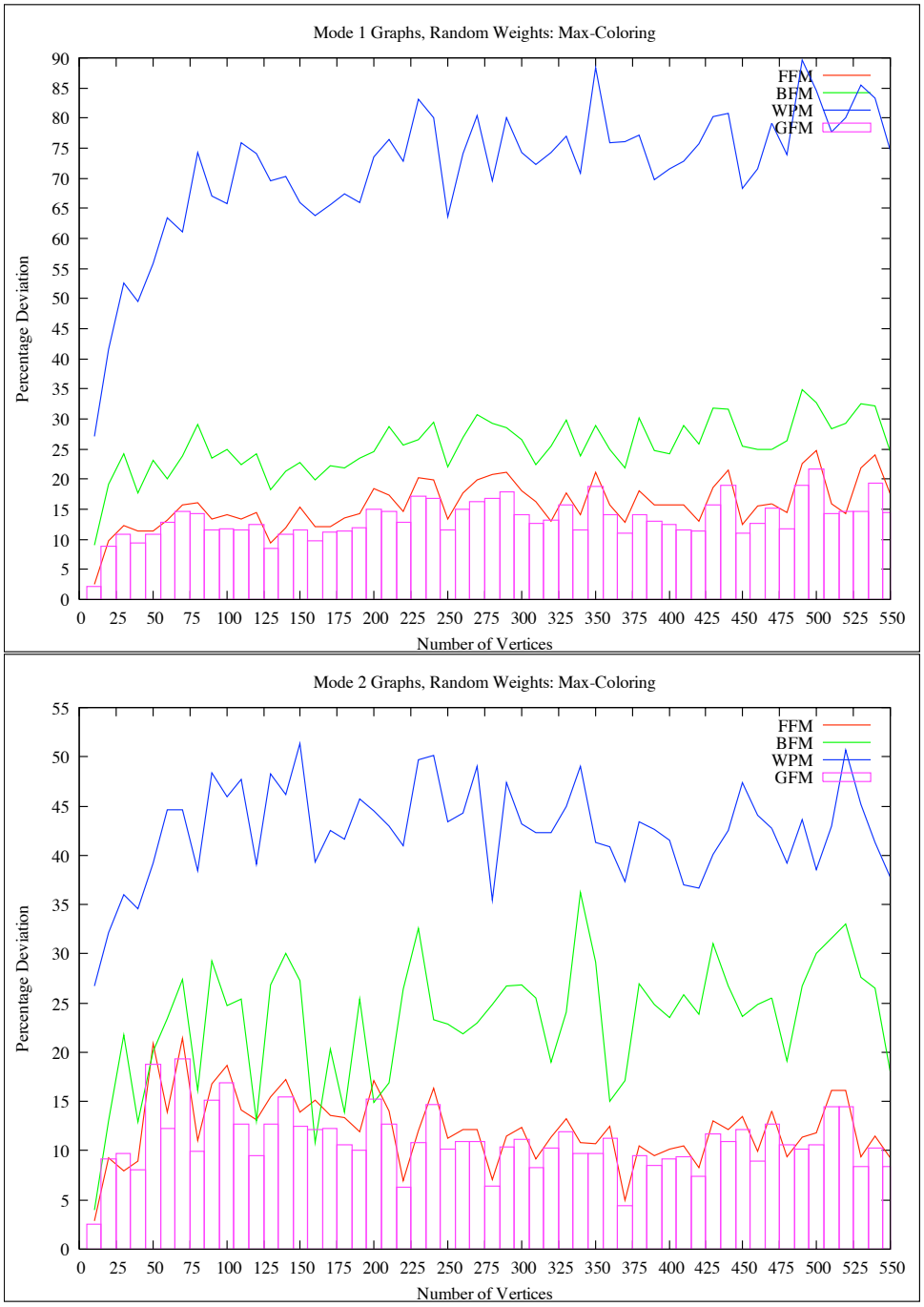

Figure 4.5: Graphs showing values for max-coloring Mode 1 and Mode 2 chordal graphs with randomly assigned weights. The x-axis corresponds to the number of vertices in the graph, and the y-axis corresponds to the percentage deviation from the weight of the maximum weight clique.

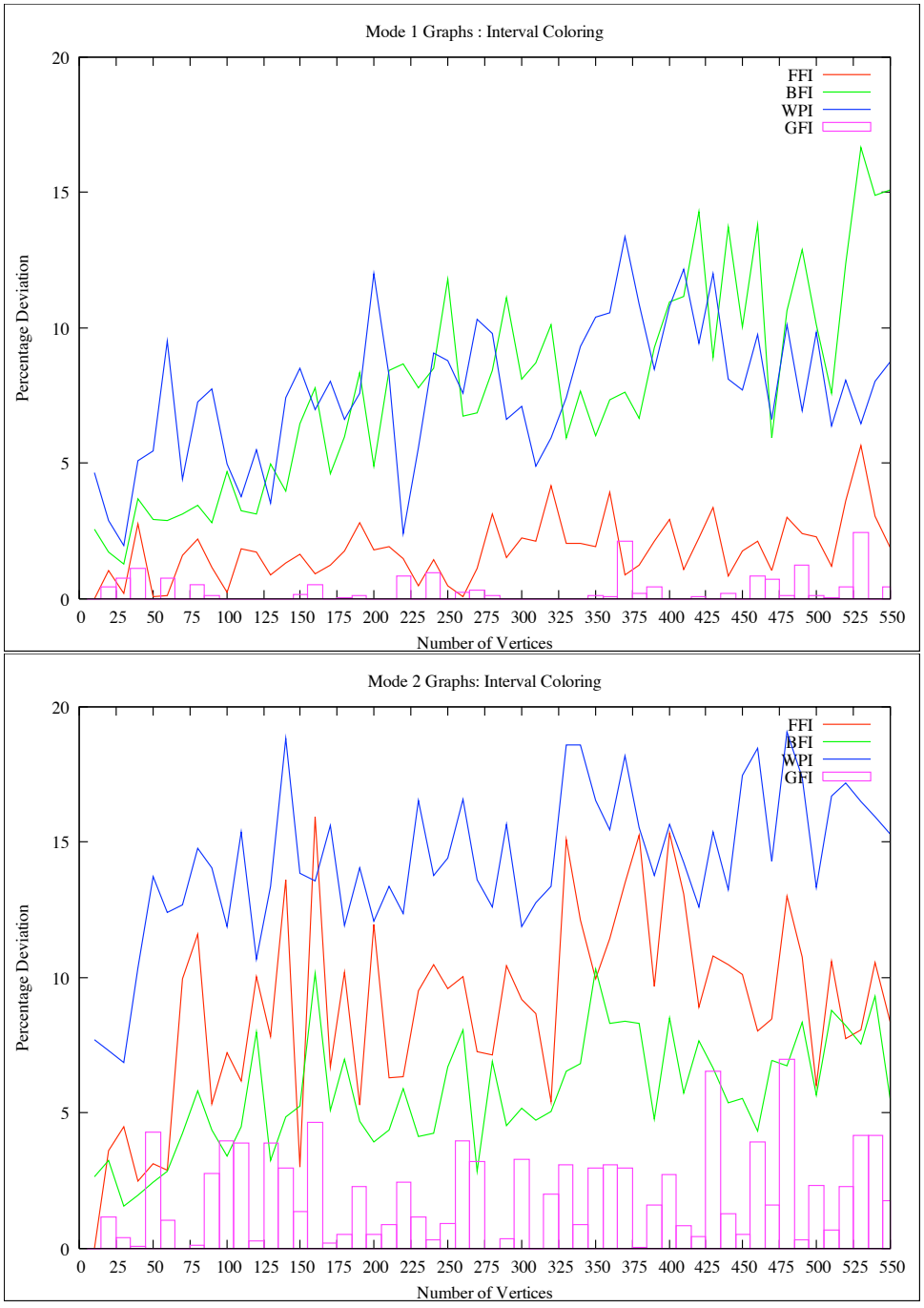

Figure 4.6: Graph showing values for interval coloring Mode 1 and Mode 2 chordal graphs. The x-axis corresponds to the number of nodes in the graph, and the y-axis corresponds to the percentage deviation from OPT for interval coloring.

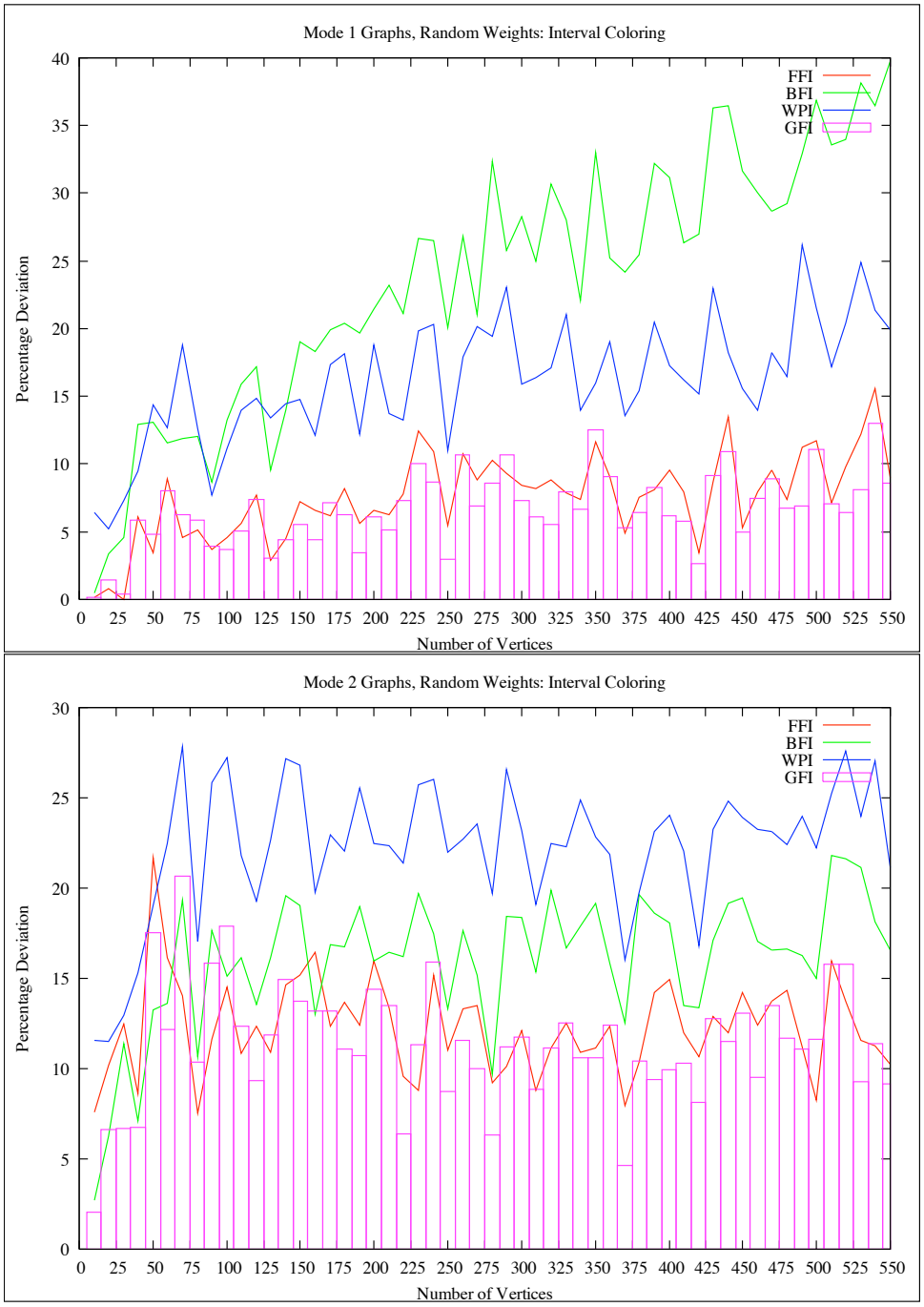

Figure 4.7: Graph showing values for interval coloring Mode 1 and Mode 2 chordal graphs with randomly assigned weights. The x-axis corresponds to the number of nodes in the graph, and the y-axis corresponds to the percentage deviation from the weight of the maximum weight clique.

1. Algorithm *Geomfit* consistently outperforms other algorithms for max-coloring deviating only by about 1.5% on average for both Mode 1 and Mode 2 graphs. Although the constant-factor guarantee for *Geomfit* holds only in the worst case, this by itself does not give much indication regarding the performance of *Geomfit* in practice.

2. Although first-fit's performance is not as good as *Geomfit*, its performance is only slightly worse than *Geomfit*, deviating only about 2% on both Mode 1 and Mode 2 graphs.

3. In the case with random weights, although the average deviation of *Geomfit* and first-fit improves as we move from Mode 1 to Mode 2 graphs, the maximum percentage deviation deteriorates significantly for *Geomfit* and the maximum percentage deviation deteriorates only slightly for first-fit. However, the performance of first-fit is almost as good as *Geomfit* for Mode 1 and Mode 2 graphs with random weights.

4. The best-fit heuristic seems to be at a disadvantage because it is constrained to always use as many colors as the chromatic number. In general, it does worse than the first-fit heuristic. However, the performance of best-fit improves as we move from Mode 1 to Mode 2 graphs. The first-fit and *Geomfit* heuristics use more colors than the chromatic number a fair number of times. 28% and 30% of the time for Mode 1 and Mode 2 graphs respectively.

5. The weight partitioning heuristic, *WtPartition*, despite providing a constant

factor approximation in the worst case, is not competitive at all relative to best-fit and first fit. A possible explanation is that *WtPartition* ends up using too many colors, while *Geomfit* strikes the right balance between grouping heavy weight vertices into the same color class, and simultaneously not using too many colors. As rows 2 and 6 of Table 4.3.3 show, *Geomfit* only uses as many colors as the chromatic number a large number of times, while *WtPartition* almost always ends up using far more colors than the chromatic number. In fact, it follows from Lemma 3.5.6 in Chapter 3 that *Geomfit* uses at most $4 \cdot \chi(G)$ colors to achieve an approximation ratio of 4, while *WtPartition* might end up using an unbounded number of colors, compared to the chromatic number of the graph.

4.3.4 Interval Coloring

We now present the data for the interval coloring problem on chordal graphs. We summarize our results in Table 3 , which shows the average deviation of the four algorithms over all the runs, for both modes. The graphs showing the performance of the algorithms are presented in Figures 4.6 and 4.7. We make the following observations.

1. For Mode 1 and Mode 2 graphs where the value of OPT is known, *Geomfit* consistently outperforms the other algorithms for interval coloring, deviating only about 1.5% on average.

2. Mode 2 graphs are significantly harder for all heuristics, except best-fit. Although all heuristics equal OPT fewer times on Mode 2 graphs than on Mode

	MODE 1				MODE 2			
	BFI	FFI	WPI	GFI	BFI	FFI	WPI	GFI
Equals OPT	2959	3330	1874	4450	2738	1971	240	3820
% Deviation	7.52	2.63	7.99	0.399	5.64	5.95	14.76	1.54
% Max Deviation	75.72	28.15	45.67	24.64	50.47	54.45	52.32	28.89
Equals LB (R)	1820	1741	958	1882	407	291	91	272
% Deviation (R)	22.75	11.99	14.75	7.34	18.11	11.26	23.76	11.72
% Max Deviation (R)	191.13	52.94	57.23	43.24	102	52.14	70.04	52.14

Table 4.3: This table shows aggregate performance over all 4950 runs of the four heuristics for interval coloring, separately for Mode 1 and Mode 2 graphs. The first three rows are for the case when the weights are chosen so that OPT is equal to the maximum weight clique, and the next three rows are the case when the weights are chosen randomly. For the case with random weights, the performance of the algorithms is compared against the weight of the maximum weight clique. The row "Equals OPT" lists the number of times each heuristic produces a coloring with weight equal to OPT, the row "% Deviation" lists the percentage deviation of the weight of the solution produced from OPT or the lower bound on OPT, averaged over the 4950 runs, and the row "Max Deviation" lists maximum percentage deviation of any algorithm from OPT or the lower bound.

1 graphs, the average and maximum percentage deviations for best-fit improve as we move from Mode 1 to Mode 2 graphs in contrast with the other three heuristics.

3. For Mode 1 and Mode 2 graphs with random weights however, the situation is less clear. Although *Geomfit* performs better than the other algorithms for Mode 1 graphs with random weights, first-fit has the smallest average deviation for Mode 2 graphs with random weights. Further, best-fit equals OPT more often than any other heuristic for Mode 2 graphs with random weights.

4. The weights in first-fit were rounded up to the nearest power of 2 in order to ensure the $O(\log n)$ bound on the approximation factor. In fact, there are simple examples where first-fit can be made to perform worse if the weights of the vertices are not raised to the nearest power of 2. However, in practice the performance of first-fit improves for interval coloring of Mode 2 graphs if we use the original weights. The deviation is at most 0.54% on Mode 1 graphs and at most 1.39% for Mode 2 graphs, compared to 2.63% and 5.95% for Mode 1 and Mode 2 graphs respectively with the weights rounded up to the nearest power of 2.

5. For the case with random weights as well, the performance of best-fit improves as we move from Mode 1 to Mode 2 graphs.

4.4 Conclusion

Our goal was to evaluate algorithms for max-coloring and interval coloring of chordal graphs. For both problems, the new *Geomfit* heuristic works as well if not better than first-fit. The other heuristics seem to be significantly worse for both problems. More generally, our experiments indicate that for the max-coloring problem, *Geomfit*, first-fit, best-fit, and *WtPartition* is the ordering of the algorithms in worsening order of performance. However, for the interval coloring problem, both *Geomfit* and first-fit perform better than the other heuristics. While *Geomfit* performs better on Mode 1 graphs, first-fit performs better on Mode 2 graphs with random weights. From an implementation point of view, first-fit may be better than *Geomfit*. First-fit is a simple heuristic to implement irrespective of the underlying graph structure. *Geomfit* relies on the procedure `mkc`(G, c_i) to compute the optimum coloring of a graph. For classes of graphs that have less structure than chordal graphs, there is typically no efficient algorithm to find an optimal coloring or even approximate it. For example, the only known algorithm to compute a coloring of a perfect graph uses the ellipsoid algorithm of Grötschel, Lovász and Schrijver [38], and this is prohibitively expensive in practice. However, for special classes of perfect graphs, like chordal graphs and interval graphs, we can compute a minimum coloring efficiently. Best-fit exhibits a performance that is different from the other algorithms in going from Mode 1 to Mode 2 graphs. While the performance of the three heuristics becomes worse as we move from Mode 1 to Mode 2 graphs, the performance of best-fit improves. This may suggest best-fit as a candidate of choice for highly irregular graphs, though this

issue needs further exploration. The performance of *Geomfit* for interval coloring on instances of Mode 1 and Mode 2 chordal graphs is surprisingly good, leading us to conjecture that an algorithm similar to *Geomfit* might indeed lead to a constant factor approximation algorithm for interval coloring as well.

BIBLIOGRAPHY

[1] U. Adamy and T. Erlebach. Online coloring of intervals with bandwidth. In Klaus Jansen and Roberto Solis-Oba, editors, *First International Workshop on Approximation and Online Algorithms, (WAOA 2003)*, volume 2909 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2004.

[2] Sanjeev Arora, Eden Chlamtac, and Moses Charikar. New approximation guarantees for the chromatic number. *Theoretical Computer Science*, 341(1):22–38, 2007.

[3] Sanjeev Arora and Carsten Lund. Hardness of approximations. In *Approximation Algorithms for NP-hard problems, Dorit Hochbaum, ed.*, 1996.

[4] J. Balogh, S.G. Hartke, Q. Liu, and G. Yu. First-fit chromatic number of planar and random graphs. submitted.

[5] A. Bar-Noy, M. Bellare, M. Halldorsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Information and Computation*, 140:183–202, 1998.

[6] A. Bar-Noy, M. Halldorsson, G. Kortsarz, R. Salman, and H. Shachnai. Sum multicoloring of graphs. *Journal of Algorithms*, 37(2):422–450, 2000.

[7] C. Berge. Motivation and history of some of my conjectures. *Discrete Mathematics*, 165:61–70, 1997.

[8] B.N.Clark, C.J.Colbourn, and D.S.Johnson. Unit disk graphs. *Discrete Mathematics*, 86:165–177, 1990.

[9] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 2005.

[10] H. Breu and D.G. Kirkpatrick. Unit disk graph recognition is np-hard. *Computational Geometry: Theory and Applications*, 9(1):3–24, 1998.

[11] G. Brightwell, H. Kierstead, and T. Trotter. First-fit coloring of interval graphs. Personal communication, 2003.

[12] A.L. Buchsbaum, H. Karloff, C. Kenyon, N. Reingold, and M. Thorup. OPT versus LOAD in dynamic storage allocation. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, 2003.

[13] C.A.Christen and S.M. Selkow. Some perfect coloring properties of graphs. *J. Combin. Theory Ser. B*, 27:49–59, 1979.

[14] M. Chrobak and M. Ślusarek. On some packing problems related to dynamic storage allocation. *Informatique théorique et Applications/Theoretical Informatics and Applications*, 22(4):487–499, 1988.

[15] M. Chudnovsky, G. Cornuejols, X. Liu, P.Seymour, and K. Vuskovic. Recognizing berge graphs. *Combinatorica*, 25:143–187, 2005.

[16] G. Confessore, P. Dell'Olmo, and S. Giordani. An approximation result for the interval coloring problem on claw-free chordal graphs. *Discrete Applied Mathematics*, 120:71–88, 2002.

[17] D.G. Corneil, S. Olariu, and L. Stewart. The ultimate interval graph recognition algorithm? (extended abstract). In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 175–180, 1998.

[18] Irit Dinur and Shmuel Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 135:439–485, 2005.

[19] Epstein, M. M. Halldorsson, A. Levin, and H. Shachnai. Weighted sum coloring in batch scheduling of conflicting jobs. APPROX'06, to appear.

[20] Leah Epstein, Thomas Erlebach, and Asaf Levin. Variable sized online interval coloring with bandwidth. In *SWAT*, pages 29–40, 2006.

[21] Leah Epstein, Magnús Halldórsson, Asaf Levin, and Hadas Shachnai. Weighted sum coloring in batch scheduling of conflicting jobs. In 9^{th} *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 116–127, 2006.

[22] Leah Epstein and Meital Levy. Online interval coloring and variants. In *ICALP*, pages 602–613, 2005.

[23] Leah Epstein and Meital Levy. Online interval coloring with packing constraints. In *MFCS*, pages 295–307, 2005.

[24] P. Erdös. Graph theory and probability. *Canad. J. Math*, 11:34–38, 1959.

[25] A.P. Ershov. Alpha - an automatic programming system of high efficiency. *Journal of the ACM*, 13(1), 1966.

[26] Bruno Escoffier, Jérôme Monnot, and Vangelis Th. Paschos. Weighted coloring: further complexity and approximability results. *Inf. Process. Lett.*, 97(3):98–103, 2006.

[27] J. Fabri. Automatic storage optimization. *ACM SIGPLAN Notices: Proceedings of the ACM SIGPLAN '79 on Compiler Construction*, 14(8):83–91, 1979.

[28] U. Feige and J. Killian. Zero knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57(2):187–199, 1998.

[29] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[30] R. Gandhi, M. Halldorsson, G. Kortsarz, and H. Shachnai. Improved bounds for sum multicoloring and scheduling dependent jobs with minsum criteria. In *WAOA*, 2004.

[31] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the theory of NP-completeness.* W.H. Freeman and Company, San Fransisco, 1979.

[32] M.R. Garey, D.S. Johnson, G.L.Miller, and C.H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal Alg. Disc. Methods*, 1:185–200, 1980.

[33] J. Gergov. Approximation algorithms for dynamic storage allocation. In *Proceedings of the 4th European Symposium on Algorithms: Lecture Notes in Computer Science 1136*, pages 52–61, 1996.

[34] J. Gergov. Algorithms for compile-time memory optimization. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages S907–S908, 1999.

[35] M.C. Golumbic. *Algorithmic graph theory and perfect graphs.* Academic Press, NY, 1980.

[36] R. Govindarajan and S. Rengarajan. Buffer allocation in regular dataflow networks: An approach based on coloring circular-arc graphs. In *Proceedings of the 2nd International Conference on High Performance Computing*, 1996.

[37] A. Gräf. Coloring and recognizing special graph classes. *Technical Report, Musikinformatik und Medientechnik Berict 20/95, Johannes Gutenbuerg Universität Mainz*, 1995.

[38] M. Grötschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer Verlag, 1988.

[39] P.M. Grundy. Mathematics and games. *Eureka*, 2:6C–8, 1938.

[40] D. J. Guan and Xuding Zhu. A coloring problem for weighted graphs. *Information Processing Letters*, 61(2):77–81, 1997.

[41] M. Halldorsson and G.Kortsarz. Tools for multicoloring with applications to planar graphs and partial k-trees. *Journal of Algorithms*, 42(2):334–366, 2002.

[42] M. Halldorsson, G. Kortsarz, A. Proskurowski, R. Salman, H. Shachnai, and J.A. Telle. Multicoloring trees. *Information and Computation*, 180(2):113–129, 2003.

[43] M. Halldorsson, G. Kortsarz, and H. Shachnai. Sum coloring interval and k-claw free graphs with application to scheduling dependent jobs. *Algorithmica*, 37(3):187–209, 2003.

[44] P. Hlinený and J. Kratochvil. Representing graphs by disks and balls. *Discrete Mathematics*, 229:101–124, 2001.

[45] D.S. Hochbaum and D.B.Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

[46] S. Irani. Coloring inductive graphs on-line. *Algorithmica*, 11(1):53–72, 1994.

[47] David.R. Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. *Journal of the ACM*, 45(2):246–265, 1998.

[48] Richard Karp. *Reducibility among combinatorial problems*. Plenum Press, New York, NY, 1972.

[49] H. Kierstead and T. Trotter. First-fit coloring of interval graphs. Personal Communication, 2005.

[50] H.A. Kierstead. The linearity of first-fit coloring of interval graphs. *SIAM J. Discrete Math*, 1:526–530, 1988.

[51] H.A. Kierstead. A polynomial time approximation algorithm for dynamic storage allocation. *Discrete Mathematics*, 88:231–237, 1991.

[52] H.A. Kierstead and J. Qin. Coloring interval graphs with first-fit. *Discrete Mathematics*, 144:47–57, 1995.

[53] H.A. Kierstead and W.T. Trotter. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143–153, 1981.

[54] J. Kratochvil. Precoloring extensions with a fixed color bound. *Acta Mathematica Universitatsis Comenianae*, 62:139–153, 1993.

[55] Michael.G. Luby, Joseph (Steffi) Naor, and Ariel Orda. Tight bounds for dynamic storage allocation. *SIAM Journal on Discrete Mathematics*, 9:155–166, 1996.

[56] Ross.M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003.

[57] Jérôme Monnot, Vangelis Th. Paschos, Dominique de Werra, Marc Demange, and Bruno Escoffier. Weighted coloring on planar, bipartite and split graphs: Complexity and improved approximation. In *ISAAC*, pages 896–907, 2004.

[58] N.S. Narayanswamy and S. Babu. First-fit coloring of interval graphs. Personal Communication, 2004.

[59] N.S. Naryanswamy. Dynamic storage allocation and on-line coloring interval graphs. In *Tenth International Computing and Combinatorics Conference, (COCOON)*, 2004.

[60] O.B.Ibarra and C.E.Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22:463–468, 1975.

[61] S. Pemmaraju and R. Raman. Improved analysis of interval coloring with bandwidths. Unpublished Manuscript.

[62] S.V. Pemmaraju and R. Raman. An improved algorithm for max-coloring hereditary graphs. Unpublished Manuscript, 2005.

[63] S.V. Pemmaraju, R. Raman, and K. Varadarajan. Buffer minimization using max-coloring. In *Proceedings of The ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 562–571, 2004.

[64] C. N. Potts and M. Y. Kovalyov. Scheduling with batching: A review. *European Journal of Operations Research*, 120:228–249, 2000.

[65] T. Rus and S.V. Pemmaraju. Using graph coloring in an algebraic compiler. *Acta Informatica*, 34(3):191–209, 1997.

[66] Wei-Kuan Shih and Wen-Lian Hsu. An approximation algorithm for coloring circular-arc graphs. In *SIAM Conference on Discrete Mathematics, San Francisco*, 1990.

[67] R.E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, amd selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13:566–579, 1984.

[68] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[69] Paul.R. Wilson, Mark.S. Johnstone, Michael Neely, and David Boles. Dynamic storage allocation : A survey and critical review. In *Proceedings of the International Workshop on Memory Management (IWMM)*, pages 1–116, 1995.

[70] D.R. Woodall. *Problem no. 4*. Cambridge University Press, Cambridge, UK, 1974.