Scheduling Light-trails on WDM Paths and Rings : an Application of Component Coloring of Graphs

Submitted in partial fulfillment of
the requirements of the degree of

Doctor of Philosophy

by

Soumitra Kumar Pal
(Roll No. 07405008)

Under the guidance of
Professor Abhiram G Ranade

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY – BOMBAY
2013

To my Master

Thesis Approval

The thesis entitled

Scheduling Light-trails on WDM Paths and Rings : an Application of Component Coloring of Graphs

by

Soumitra Kumar Pal

(Roll No. 07405008)

is approved for the degree of

Doctor of Philosophy

[illegible]

Professor Ajit A. Diwan
CSE, IIT Bombay
Internal Examiner

[illegible]

Professor Sandeep Sen
CSE, IIT Delhi
External Examiner

[illegible]

Professor Abhiram G. Ranade
CSE, IIT Bombay
Supervisor

[illegible]

Professor Sachin B. Patkar
EE, IIT Bombay
Chairman

Date: 31 July 2013

Place: Powai, Mumbai

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

SB Pal

(Signature of student)

Soumitra Kumar Pal

(Name of student)

Roll No. **07405008**

Date: 6 August 2013

Abstract

We consider the problem of scheduling communication on optical WDM (wavelength division multiplexing) networks using the *light-trails* technology, an attractive solution to the problem of bandwidth provisioning. We seek to design scheduling algorithms such that the given transmission requests can be scheduled using minimum number of wavelengths. We provide algorithms and close lower bounds for two versions of the problem. In the *stationary* version, the pattern of transmissions (given) is assumed to not change over time. In the *online* version, the transmissions arrive and depart dynamically, and must be scheduled without upsetting the previously scheduled transmissions. We also consider two weighted versions in which transmissions have an integer weight (bandwidth requirement). In the *splittable* weighted version, (the weight of) each transmissions can be split into several parts which can be scheduled in different light-trails. In the *nonsplittable* weighted version the transmissions cannot be split.

The light-trail scheduling problem can be abstracted as a generalization of the graph vertex coloring problem which we call the graph *component coloring* problem. Here the objective is to color the vertices of the given graph using the minimum number of colors. Unlike vertex coloring, here adjacent vertices are allowed to have the same color. However, the size of a connected component in the induced subgraph having the same color must not exceed some parameter C. Vertex coloring is a special case of the problem for $C = 1$. In the context of the applications in scheduling light-trails on path/ring networks, the graphs in component coloring are interval/circular-arc graphs. We also consider the problem on proper interval graphs (PIGs) which are known to admit polynomial time algorithms for many problems which are hard on general graphs. Component coloring also has applications in scheduling communications in reconfigurable bus architectures and in efficiently managing storage for evolving databases.

Our main contributions are as follows. Firstly, we study the stationary version of component coloring on PIGs. We give a linear time exact algorithm for the unweighted problem. The algorithm exploits a nice structure of PIGs for which the problem reduces to partitioning the vertex set such that the graph generated by contracting each part admits a vertex coloring with the minimum number of colors. We extend this idea to give a quadratic time exact algorithm for the splittable problem and a 2-approximation algorithm for the NP-hard nonsplittable problem.

Secondly, we study the stationary light-trail scheduling problem on optical path and ring networks. On path networks, the nonsplittable problem is NP-hard and the complexity of the unweighted problem and the splittable problem are not known. On ring networks, even the unweighted problem is NP-hard. For the problem on a p processor path/ring network a simple lower bound is ω, the congestion or the maximum total traffic required to pass through any link. We give an algorithm that schedules the transmissions using $O(\omega + \log p)$ wavelengths, and so our algorithm can be seen to use a number of wavelengths close to the optimal. We show that there are input transmissions for which lower bound ω is small, but which yet require $\Omega(\omega + \log p/\log\log p)$ wavelengths. In a weak sense, this justifies the additive $\log p$ term in the number of wavelengths taken by our algorithm.

For the online problem, we use the notion of competitive analysis. In this, an algorithm for the online problem does not know the input transmissions in advance and must schedule the transmissions as and when they arrive. The algorithm is evaluated by comparing its performance to that of an *offline adversary*, an algorithm which is given at the beginning all the transmissions including their arrival and departure sequence. We give two online algorithms that work on both path and ring networks. We establish that our first algorithm is $\Theta(\log p)$-competitive, *i.e.,* it requires $\Theta(\log p)$ times as many wavelengths as needed by the offline adversary. We also prove that no online algorithm can do better by showing the lower bound on the competitive ratio of any algorithm for the problem to be $\Omega(\log p)$. We also give a second algorithm for this problem, it is in fact a simplified version of the first. It actually performs better than the first algorithm in many situations; however, we can prove that its competitive ratio is worse, between $\Omega(\log^2 p/\log\log p)$ and $O(\log^2 p)$.

Finally, we simulate our online algorithms for some traffic models of the light-trail based optical ring networks and compare them to a baseline algorithm. We find that except for the case of very low traffic, our algorithms are better than the baseline. For very local traffic, our algorithms are in fact much superior.

Keywords: Graph, Proper, Interval, Component, WDM, Light-trail, Reconfigurable Bus Architecture, Evolving Database, Weighted, Splittable, Fragmented, Generalized, Coloring, Partition, Scheduling, Routing, Algorithm, Online, Approximation

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graph coloring is one of the most well studied fields in graph theory. In the simplest graph coloring problem, commonly known as *vertex coloring*, we need to color the vertices of given a graph using the minimum number of colors so that no two adjacent vertices are assigned the same color. Because of many applications of graph coloring in mathematics, computer science and other areas, vertex coloring has been generalized in many ways. In this thesis we study one of such generalizations which we call the *component coloring* problem.

1.1 Motivation

Our formulation of the component coloring problem is motivated by a problem on scheduling transmission requests on the *light-trails* based optical WDM (wavelength division multiplexing) networks. Light-trails [18] are considered to be an attractive solution to the problem of bandwidth provisioning. The key idea in the light-trails is the use of optical shutters which are inserted into the optical fiber, and which can be configured to either block the optical signal or let it pass through. By configuring some shutters ON (signal let through) and some OFF (signal blocked), the network can be partitioned into subnetworks, called *light-trails*. Thus by increasing the number of light-trails, more simultaneous transmissions are possible, albeit going a shorter distance. The optical shutters are controlled by an auxiliary network ("out-of-band channel") which is typically electronic, and the shutter switching time is of the order of milliseconds as opposed to optical signals which have frequencies of several gigahertz. However, the high bandwidth available per shutter configuration typically remains underutilized because (i) typically the transmissions have comparatively very low bandwidth requirement, and (ii) at any given time and using a given wavelength, there can be at most one transmission in progress to avoid optical interference. The light-trails based networks circumvent this problem by using time division multiplexing inside a single light-trail; in other words, a single light-trail is used to serve several transmission requests even during the period when shutter configurations cannot be changed, provided the communicating processors lie within the light-trail.

Figure 1.1 shows an example of scheduling transmissions on a light-trail based optical path network of 14 processors. We need to schedule 7 transmissions $[0, 4]$, $[1, 5]$, $[2, 6]$, $[3, 10]$, $[7, 11]$, $[8, 12]$, $[9, 13]$ of unit bandwidth requirements. The capacity of an wavelength, *i.e.,* the maximum bandwidth served by a wavelength is $C = 2$ units. Thus at most 2 transmissions can be in any light-trail. We use 3 wavelengths. On the first wavelength we create a single light-trail spanning the whole network and assign transmissions $[2, 6]$ and $[3, 10]$ on it. Note that transmissions $[2, 6]$, $[3, 10]$ are time multiplexed and the whole span of light-trail $[0, 13]$ is dedicated to each of them during the time they are active. The dotted regions cannot be used for any other transmission because light signal used for a transmission travels all the way till the end and hence at any time instant only one transmission can be active in the light-trail. On the second wavelength by switching off the shutters of that wavelength at processors $0, 7, 13$ two light-trails $[0, 7]$, $[7, 13]$ are created. Transmissions $[0, 4]$, $[1, 5]$ are assigned to light-trail $[0, 7]$ and transmissions $[7, 11]$, $[8, 12]$ are assigned to light-trail $[7, 13]$. The transmission $[9, 13]$ is assigned to a single light-trail on the third wavelength.

In this thesis we consider the problem of scheduling transmissions on light-trail based optical *path* and *ring* networks. It is customary to consider two problem variations: *nonsplittable*, in which a transmission must be assigned to a single light-trail, and *splittable*, in which a transmission can be split into two or more transmissions by dividing up the bandwidth requirement, and each of them can be assigned to different light-trails. Note that when a transmission is split into multiple transmissions, the source and destination remain the same, only the required bandwidth is divided.

We examine the problem in the *stationary* setting, in which interprocessor transmission demands are known and do not change, as well as the *dynamic* setting in which transmission requests arrive and depart after being served, in an online manner. For both problems, our objective is to minimize the number of wavelengths needed to accommodate the given traffic, using the best possible partitioning of the network into light-trails (for each wavelength), and the best possible assignment of requests to light-trails. Our approach is applicable to the setting

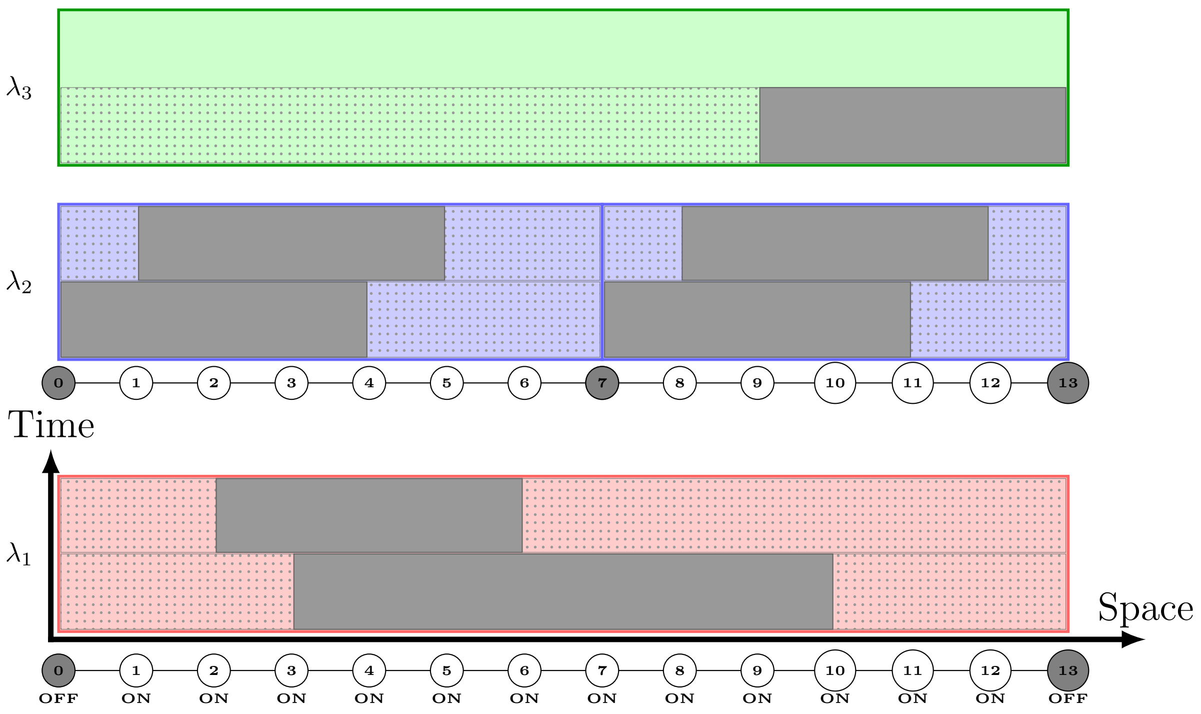


Figure 1.1: Optical Transmission using Light-trails

in which a fixed number of wavelengths is available as follows. If our analysis indicates that some λ wavelengths are needed while only λ_0 are available, then effectively the system will have to be slowed down by a factor λ/λ_0. This is of course only one formulation; there could be other formulations which allow requests to be dropped and analyze what fraction of requests are served.

Abstraction

The light-trail scheduling problem can be abstracted as the following problem on graphs. For each transmission request, create a vertex with weight equal to the bandwidth requirement. Two vertices are adjacent if the corresponding transmissions overlap, *i.e.,* they use at least one common link. In the most efficient scheduling the transmissions served by a light-trail induce a connected subgraph because otherwise we can divide the light-trail further into smaller light-trails, each accommodating a connected component of the subgraph. Thus the transmissions assigned to all the light-trails on a wavelength induce a subgraph having connected components of weight at most the capacity of a wavelength, say C, and hence light-trail scheduling is equivalent to getting a partition of smallest order of the vertex set such that each class induces components of weight at most C. We call this graph partitioning problem as the *component col-*

oring problem. We give the formal definition of this abstract problem and several of its variants in section 1.2.

Other Applications

The Reconfigurable Bus Architectures are similar to the light-trails based networks. Hence scheduling communications on them is also an application of component coloring. We discuss this in detail in Section 2.4 of Chapter 2 on the related work. Component coloring also has applications in a seemingly unrelated field of evolving databases, the details which are discussed in Section 2.2 of Chapter 2.

1.2 Component Coloring

The component coloring problem is a generalization of the vertex coloring problem. In this generalized problem two adjacent vertices can be assigned the same color but over all the number of vertices in a monochromatic component must not exceed some given upper bound. Motivated by the light-trail scheduling problem, we consider the component coloring problem in several settings – (a) *stationary*, in which all the vertices are known in advance and *online*, in which the vertices are revealed one at a time, (b) *weighted*, in which vertices have weights, as well as *unweighted*, in which all vertices have weights equal to 1, and (c) in case of weighted, both *splittable*, in which the weight of a vertex can be distributed among several of its copies which can be colored separately and *nonsplittable*, in which splitting of weights is not allowed.

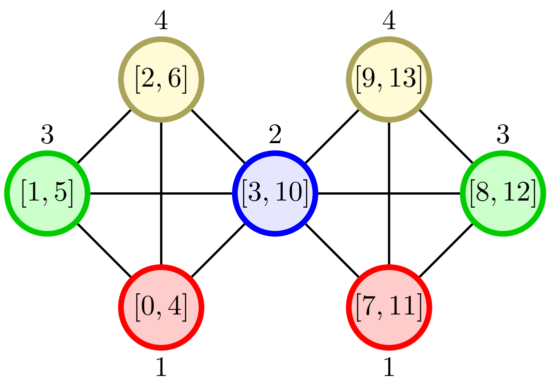
Before we formally define each of the problem variations, we define some useful terms. A *λ-assignment* of a graph $G = (V, E)$ is a map from V to some set of λ colors such as $\{1, 2, \ldots, \lambda\}$; this assignment may not be 'proper' in the standard notion that two adjacent vertices must be assigned different colors. A *color class* i is the set of vertices assigned color i under the λ-assignment. A *monochromatic component* or *chromon* of G under a λ-assignment is a connected component of the sub-graph induced by a single color class. The *size of a chromon* is the number of vertices in it. For a weighted graph, the *weight of a chromon* is the sum of weights of vertices in it.

Stationary Unweighted Problem

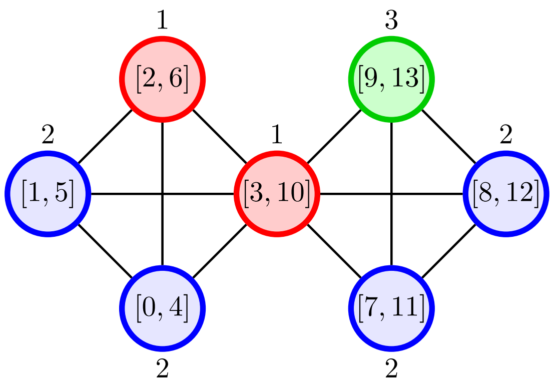
A graph G is (λ, C)-*colorable* if it has a λ-assignment in which every chromon has size at most C and such an assignment is called a (λ, C)-*coloring*. The C-*component chromatic number* of G, denoted by $\chi_C(G)$ is the minimum λ for which G is (λ, C)-colorable. The stationary unweighted problem is defined as follows:

Problem	Stationary unweighted component coloring
Given	A simple graph G, and a constant C
Goal	Find a (λ, C)-coloring of G such that $\lambda = \chi_C(G)$

Note that vertex coloring is a special case of component coloring where $C = 1$, *i.e.*, each chromon is an individual vertex. Figure 1.2 shows the graph G corresponding to the instance of the light-trail scheduling problem shown in Figure 1.1. There is a vertex for each transmission and there is an edge between two vertices if the two corresponding transmissions use a common link. Figure 1.2 also shows the solutions to the stationary unweighted problem on G for $C = 1, 2$. For vertex coloring we need at least 4 colors because either clique has 4 mutually adjacent vertices. For $C = 2$ either clique requires at least 2 colors. Let the left clique be assigned colors $1, 2$. Since the middle vertex is in an already full chromon of color 1, no more vertex of right clique can be assigned color 1. At most 2 of the remaining 3 vertices of the right clique can be assigned color 2. So we need an additional color. Thus the colorings are optimal for $C = 1, 2$, respectively.



(a) Vertex Coloring



(b) Component Coloring for $C = 2$

Figure 1.2: Stationary Component Coloring

Stationary Nonsplittable Weighted Problem

A graph G with weight W on vertices is (λ, C)*-colorable* if it has a λ-assignment in which every chromon has weight at most C and such an assignment is called a (λ, C)*-coloring* of G with weight W. The C*-component chromatic number* of G with weight W, denoted by $\chi_C(G, W)$ is the minimum λ for which G is (λ, C)-colorable. The stationary (nonsplittable) weighted problem is defined as follows:

Problem	Stationary (nonsplittable) weighted component coloring
Given	A simple graph G with integer weights W on vertices, and a constant C
Goal	Find a (λ, C)-coloring of G with W such that $\lambda = \chi_C(G, W)$

Stationary Splittable Weighted Problem

For a graph $G = (V, E)$ with weight W on V, consider another graph $G' = (V', E')$ with weight W' on V' such that (i) the weight $W(v_i)$ of $v_i \in V$ is divided among a separate set of n_i vertices $v_{i1}, v_{i2}, \ldots, v_{in_i}$ in V', *i.e.,* $W(v_i) = \sum_{k=1}^{n_i} W'(v_{ik})$, (ii) $V' = \cup_{i,k}\{v_{ik}\}$, and (iii) for each edge $(v_i, v_j) \in E$ there is an edge $(v_{ik}, v_{jl}) \in E'$ for all $k = 1, \ldots, n_i$ and $l = 1, \ldots, n_j$. We call G' with W' a *weight-split graph* of G with W. A graph G with weight W is (λ, C)-*split colorable* if it has a weight-split graph G' which is (λ, C)*-colorable* and such a coloring is called a (λ, C)*-split coloring* of G. The C*-component splittable chromatic number* of G with weight W, denoted by $\hat{\chi}_C(G, W)$ is the minimum λ for which G is (λ, C)-split colorable. The stationary splittable weighted problem is defined as follows:

Problem	Stationary splittable weighted component coloring
Given	A simple graph G with integer weight W on vertices, and a constant C
Goal	Find a (λ, C)-split coloring of G with W such that $\lambda = \hat{\chi}_C(G)$

Online Problems

In the online setting the vertices arrive and depart in stages. A vertex is said to be *active* between its arrival and departure. An algorithm for the online problem must immediately assign a color to a newly arrived vertex, must not change the color as long as the vertex is active and can reuse the color assigned to a departed vertex for some other vertices that arrive in later stages. Here we define the nonsplittable weighted version of the problem; the unweighted and the splittable weighted cases can be similarly defined.

A graph G with weight W on vertices and the order σ of arrivals and departures of the vertices is said to be (λ, C, σ)*-colorable* if G has a λ-assignment such that at every stage in σ every chromon of active vertices has weight at most C and such an assignment is called a (λ, C, σ)*-coloring* of G with W. The *online* C*-component chromatic number* of G with weight W and order σ, denoted by $\chi_C(G, W, \sigma)$ is the minimum λ for which G is (λ, C, σ)-colorable. The online weighted problem is defined as follows:

Problem	Online weighted component coloring
Given	A simple graph G with integer weights W on vertices, order σ of arrivals and departures of vertices, and a constant C
Goal	Find a (λ, C, σ)-coloring of G with W such that $\lambda = \chi_C(G, W, \sigma)$

1.3 Contribution

In this thesis we focus on the light-trail scheduling problem on path and ring networks. Hence we study and solve the component coloring problem on the classes of graphs that arise on such networks, *i.e.,* interval graphs and circular-arc graphs, respectively. We also consider the problem on proper interval graphs (PIGs) which are known to admit polynomial time algorithms for many problems which are hard on general graphs.

Our main contributions are as follows. Firstly, we study the stationary problems on the PIGs. For the unweighted problem we give a linear time exact algorithm. Our algorithm first solves the decision version of the problem for which the paper [52] gives a super quadratic time algorithm in the context of an application in evolving databases. Though the graphs considered in [52] are also PIGs, our algorithm achieves better performance by exploiting a nice structure of the PIGs for which the problem reduces to partitioning the vertex set such that the new graph generated by contracting each part admits a vertex coloring with the minimum number of colors. We extend the algorithm for the unweighted problem to solve the splittable weighted problem on PIG in time quadratic in number of vertices. However, the nonsplittable weighted version of the problem is NP-hard even on PIGs as the bin-packing problem which is NP-hard [31], is a special case of the problem where each item corresponds to a vertex with weight equal to the size of the item, there is an edge between every pair of vertices and each bin corresponds to a color. The graph in this special case is a complete graph which is also a PIG. We use the algorithm for the splittable weighted problem to get a 2-approximation algorithm for the nonsplittable weighted problem on PIGs. Note that the authors in [52] use a similar idea to give a 2-approximation algorithm for the weighted storage problem by using the algorithm for unweighted problem, but they optimize a different objective as discussed in Section 2.2.

Secondly, we study the stationary problems on general interval graphs and circular-arc graphs. For these two classes of graphs the algorithms in the literature are based on heuristics and no provable bounds are provided on their performances. The unweighted problem is NP-hard on circular-arc graphs because in the special case of $C = 1$ the problem is same as vertex coloring which is known to be NP-hard on circular-arc graphs. Though vertex coloring has polynomial time algorithm on general interval graphs, we do not know the complexity of the unweighted problem and the splittable weighted problem on general interval graphs. The nonsplittable weighted problem on general interval graphs is NP-hard as we have seen it to be NP-hard even for the PIGs.

For the weighted stationary problem we give an approximation algorithm which works on both graph classes – interval graphs and circular-arc graphs and in both cases – splittable and nonsplittable. We describe the algorithm using the terminologies from the light-trail scheduling problem; it can be similarly described using equivalent coloring terminologies. For the light-trail scheduling problem a simple lower bound can be computed as the congestion ω, the maximum bandwidth requirement across any link, divided by C, the maximum bandwidth served by a wavelength. For a network of p nodes, our algorithm schedules all transmission requests using $O(\omega/C + \log p)$ wavelengths and hence our algorithm can be seen to use a number of colors close to the optimal. The reader may wonder why the additive $\log p$ term arises in the result. We show that there are input transmissions for which lower bound ω/C is small, but which yet require $\Omega(\omega/C + \log p/\log\log p)$ colors. In a weak sense, this justifies the additive $\log p$ term in the number of wavelengths taken by our algorithm. In the graph coloring terminology, ω represents the weight of a maximum clique in the graph and p denotes the number of distinct endpoints of the corresponding interval/circular-arc representation of the input graph.

Thirdly, we study the online problem for which the performance of the algorithms in the literature has been based on simulations. Here also we describe our algorithms using he terminologies from the light-trail scheduling problem. For the online problem, we use the notion of competitive analysis [1, 14]. In this, an online algorithm which must respond without the knowledge of the future is evaluated by comparing its performance to that of an *offline adversary*, an algorithm which is given all the transmissions at the beginning. Clearly, the offline adversary must perform at least as well as the best online algorithm. We establish that our first algorithm is $\Theta(\log p)$-competitive, *i.e.,* it requires $\Theta(\log p)$ times as many wavelengths as needed by the offline adversary. We also prove that no online algorithm can do better by showing the lower bound on the competitive ratio of any algorithm for the problem to be $\Omega(\log p)$. A multiplicative $\Theta(\log p)$ factor might be considered to be too large to be relevant for practice; however, the experience with online algorithm design is that such algorithms often give good hints for designing practical algorithms. We also give a second algorithm for this problem: it is in fact a simplified version of the first. It actually performs better than the first algorithm in many situations; however, we can prove that its competitive ratio is worse, between $\Omega(\log^2 p/\log\log p)$ and $O(\log^2 p)$.

Finally, we also simulate two algorithms based on our online algorithms for inputs arising in some traffic models of the light-trail networks. We compare them to a baseline algorithm which keeps the optical shutter switched OFF only in one processor for each wavelength. Note that at least one processor should switch OFF its optical shutter, otherwise the light signal will interfere with itself after traversing around the ring. We find that except for the case of very low traffic, our algorithms are better than the baseline. For very local traffic, our algorithms are in fact much superior.

1.4 Outline

The rest of the thesis is organized as follows. We begin in Chapter 2 by comparing our work with previous related work. In Chapter 3 we describe our solutions to the stationary problems on proper interval graphs. Chapter 4 discusses our algorithm for the stationary problem on general interval graphs. The same algorithm also works on circular-arc graphs. For the same classes of graphs we give our solutions to the online problem in Chapter 5. Finally we conclude in Chapter 6.

1.5 Credit

The results in Chapter 3 are based on the paper [22] which is a joint work with Professor Ajit Diwan, member of my Research Progress Committee, and Professor Abhiram Ranade, my advisor. The results in Chapters 4 and 5 are based on the papers [73, 74], both are joint work with my advisor.

Chapter 2

Literature Review

In this chapter we discuss previous work on the component coloring problem and other related problems in the literature which can be categorized into the following four broad areas: (i) graph coloring and its generalizations, (ii) applications in evolving databases, (iii) applications in light-trails, and (iv) applications in reconfigurable bus architectures. We discuss the work on these four areas in Sections 2.1, 2.2, 2.3 and 2.4, respectively. We conclude each of these sections separately by contrasting with our work.

2.1 Graph Coloring

Graph coloring has been one of the most studied fields of graph theory [47, 53, 77]. In the simplest graph coloring problem, commonly known as *(proper) vertex coloring*, given a graph G with the set of vertices V and the set of edges E, we need to assign a set of λ colors, generally indexed by integers $1, 2, \ldots, \lambda$, to the vertices in V such that the two endpoints of any edge in E are assigned different colors. Such a coloring of V is called a *(proper) λ-coloring*. The minimum λ for which a there is a λ-coloring for G is called the *chromatic number* of G denoted by $\chi(G)$. The objective of vertex coloring is to obtain a $\chi(G)$-coloring of a given G.

Karp [50] showed that it is NP-complete to decide if a graph has a vertex coloring using 3 colors. There have been attempts to get *approximation algorithms* for vertex coloring, *i.e.,* polynomial time algorithms which take at most $\alpha \cdot \chi(G)$ number of colors where α is called the *approximation factor*. However, the approximate coloring also turns out to be hard. So far the best known approximation factor for general graphs is $O(n(\log \log n)^2 / \log^3 n)$ [43]. Garey and Johnson [30] proved that it is NP-hard to approximate the chromatic number within a factor of $(2-\epsilon)$ for any $\epsilon > 0$. Getting an approximation algorithm with a larger constant factor also seems impossible. Lund and Yannakakis [62] showed that there is no polynomial time algorithm that approximates $\chi(G)$ of a graph G with n vertices with a factor $n^{1-\epsilon}$ for some particular small $\epsilon > 0$, unless $P = NP$.

There are also efforts in getting polynomial time algorithms for special classes of graphs. *Planar graphs* have been of huge interest because of the applications to map coloring. Though Stockmeyer [88] showed that it is NP-complete to decide if a planar graph G is 3-colorable, even when G has maximum degree at most four, it is folklore that four colors are enough for coloring planar graphs. In the celebrated Four Color Theorem Appel, Haken and Koch [4] showed that for every planar graph G on n vertices, $\chi(G) \leqslant 4$ and there is an $O(n^2)$ time algorithm for 4-coloring G. For *interval graphs*, the greedy algorithm applied on the vertices ordered by the left endpoints of the corresponding interval representation, gives an exact solution. However, vertex coloring on *circular-arc graphs* is NP-hard [32] and has a 2-approximation algorithm [91].

2.1.1 Generalized Coloring

There have also been attempts to generalize the vertex coloring problem in many ways [97]. A λ-coloring of a graph $G = (V, E)$ can also be considered as a partition $\{V_1, V_2, \ldots, V_\lambda\}$ of the vertex set V such that the class V_i contains the vertices assigned to color i. In proper vertex coloring each class is an independent set. In the generalized colorings this constraint is relaxed. The generalized colorings are also known as *improper* colorings, *relaxed* colorings, *conditional* colorings and so on.

In the most generalized coloring, each color class V_i of G must satisfy some graph property $\mathcal{P}$ and in such case G is said to be $(\lambda, \mathcal{P})$-colorable. A graph property $\mathcal{P}$ is a set of graphs and a graph G is said to satisfy $\mathcal{P}$ if $G \in \mathcal{P}$. Several properties have been considered for $\mathcal{P}$ – graphs of low diameters [58], graphs of small number of edges [86], graphs of smaller tree widths [21], graphs without any cycle of a given length [20] and so on.

However a significant amount of work on $\mathcal{P}$-colorings, which are more closely related to the component coloring can be described using the notation of $(\lambda, C)^\gamma$-coloring introduced by Frick [28]. Here $\mathcal{P}$ contains graphs G with the property $\gamma(G) \leqslant C$ where γ is a graph invariant and C is a given constant. The minimum λ for which G has an $(\lambda, C)^\gamma$-coloring is called the *Cth γ-chromatic number* of G and will be denoted as $\chi^\gamma_C(G)$. A graph G is called *$(\lambda, C)^\gamma$-colorable* if $\chi^\gamma_C(G) \leqslant \lambda$ and *$(\lambda, C)^\gamma$-chromatic* if $\chi^\gamma_C(G) = \lambda$. The problem has been studied for several graph invariants γ – (i) clique number ω, the size of a largest complete subgraph [15, 27], (ii) maximum degree Δ [3, 29, 49, 70, 71], (iii) the degeneracy $\rho = \max_{H \subseteq G} \delta(G)$ where $\delta(G)$ is the minimum degree of a vertex in G [5, 12, 17, 81, 83], (iv) the path number τ, the length of the longest path [16, 48, 97].

2.1.2 Component Coloring

Using Frick's notation component coloring can be represented as $(\lambda, C)^{\nu}$-coloring where $\nu(G)$ denotes the number of vertices in the largest component in G. The only work [85] available in the graph coloring literature on $(\lambda, C)^{\nu}$-coloring gives several graph theoretic results such as bounds on the generalized chromatic number, extensions of some existential results on vertex coloring and so on. There also exists a significant amount of work on *fragmented coloring* [52] which is somewhat dual to component coloring: given a graph and a fixed set of λ colors, find a (λ, C)-coloring with minimum C. The minimum value of C is denoted by D_{λ}.

The paper [52] showed that for each C there exists a planar graph that has no $(3, C)$-coloring, thus establishing that the straightforward bound $\chi_C \leqslant 4$ on planar graphs due to Four Color Theorem [4] is in fact tight. The authors [52] also proved that for sufficiently large Δ there are graphs of maximum degree Δ and order n which have both $\chi_{O(n)}$ and $D_{O(\sqrt{\Delta})}$ arbitrarily large.

Most of the remaining work on fragmented coloring is on bounding D_2 for graphs with smaller Δ. For $\Delta \leqslant 3$ any maximum edge-cut proves that $D_2 \leqslant 2$. For $\Delta = 4$, Haxell, Szabó, and Tardos [45] showed that $D_2 \leqslant 6$ and for $\Delta = 4$, Berke [10] showed that $D_2 \leqslant 1908$. For $\Delta = 6$, Alon, Ding, Oporowski, and Vertigan [2] showed that for every k there is a graph H_k with n vertices such that $D_2(H_k) = \Omega(\sqrt{n})$. For outer planar graphs Berke [10] showed that $D_2 \leqslant 2\Delta - 1$.

Matoušek, and Přívetivỳ [65] showed that for grid graphs $\{1, 2, \ldots, n\}^d$ with diagonals, $\Omega(n^{d-1} - d^2 n^{d-2}) \leqslant D_2 \leqslant n^{d-1}$ and for triangulated grid graphs, $D_2 = \Omega(n^{d-1}/\sqrt{d})$. Edwards and Farr [23] showed that for a n-vertex graph with $\Delta = 4$, there exists a local optimal algorithm that produces a $(2, O(2^{(2 \log_2 n)^{1/2}}))$-coloring. For the family of minor-closed graphs, Linial, Matoušek, Sheffet, and Tardos [59] showed that $\Omega(n^{2/(2\lambda-1)}) \leqslant D_{\lambda} \leqslant O(n^{2/(\lambda+1)})$ for every fixed λ. A minor of G is graph obtained by deleting or contracting edges in G.

Berman and Paul [11] worked on finding D_{λ} on k-trees. A k-tree is recursively defined as follows: the complete graph K_k is a k-tree and if G is a k-tree of $n - 1$ vertices then the graph obtained by adding a new vertex to any of the induced K_k subgraphs of G results in a new k-tree. Normal trees are 1-trees. For k-trees [11] the authors showed that $\lfloor n^{1/\lambda} \rfloor \leqslant D_\lambda \leqslant k\lceil n^{1/\lambda} \rceil$.

The Hadwiger's conjecture is one of the long standing open problems in graph coloring which says that for a graph G without any K_k minor, $\chi(G) \leqslant k-1$ or in other words, $\chi_{f(k)}(G) \leqslant h(k)$ where f, k are constant functions of k such that $f(k) = 1$ and $h(k) = k - 1$. The authors in [59] linked the component coloring problem with the Hadwiger's conjecture by mentioning that a result by Kawarabayashi and Mohar [51] can be interpreted as the following relaxed form of the conjecture. For any $k > 1$ and any graph G having no K_k as a minor there exists a function f of k such that $\chi_{f(k)} \leqslant \lceil 15.5k \rceil$.

2.1.3 Remarks

There are a number of dimensions along which the work in the literature on graph coloring can be classified: (a) property of each color class such as bounded clique size [15, 27], path length [16, 48, 97], degeneracy [17, 81, 83], degree [3, 49, 70], component size [10, 23, 45, 51, 52, 59, 65, 85] and several others [20, 21, 58, 86], (b) results inspected varying from graph theoretic results such as relations between different chromatic numbers and graph invariants [17, 85], bounds on chromatic numbers [17, 20], generalizations of results from proper coloring such as existence results [85], and so on; complexity results such as hardness and approximibility [26]; algorithmic results such as polynomial time algorithms [5, 52, 83], approximations algorithms [12, 29, 52], exact and heuristic algorithms [71], (c) type of graphs investigated such as planar [17, 83], grid [65], interval and circular-arc [29, 49] etc. The stationary version as well as the online version have also been considered.

Our results can be put in this full spectrum of results as follows. We seek polynomial time exact (if possible), approximation and online algorithms with provable bounds on performance for both weighted and unweighted, both stationary and online versions of the component coloring (ν-coloring) problem on interval graphs and circular-arc graphs. The only work that is very close to our result is the work in [52] which is discussed separately in Section 2.2. Though the authors in [52] also gave competitive algorithms for the online component coloring problem on unit interval graphs, they optimize D_C^ν whereas we optimize χ_C^ν. Moreover we also give algorithms for general interval graphs and circular-arc graphs.

2.2 Storage Management in Evolving Databases

Among the three applications of component coloring, the work closest to our work in terms of the objectives of investigations and the solution techniques used, is the work on the storage management in evolving databases.

In an evolving database system, large volume of data arriving continuously over time makes it difficult for the database management system to efficiently store data as well as to quickly serve queries on the stored data. One of the models that has been considered is to use *sliding window indices* [52] for some parameter T, *i.e.,* to maintain data for the last T days with the reasonable assumption that most of the queries are on the most recent data.

The problem of storing data in a sliding window setting can be stated as follows. There are λ compartments, $b_1, b_2, \ldots, b_\lambda$, each with a capacity B. Data items $x_1, x_2, \ldots$ with integer weights arrive dynamically and *expire* after some fixed T days counting from the day of arrival. An item x_i is said to be *active* after it arrives and until it expires. At any given time a compartment b_j is said to be *active* if it contains at least one active item; its *load* is defined to be the total weight of all items (live or expired) that were assigned to it since it was last inactive. We need to efficiently assign the items to the compartments without violating the constraint that an expired item from a compartment can be removed only when the compartment is inactive.

There are many ways to measure and optimize the quality of an assignment [52]. The objective relevant to our work is to minimize the *maximum* load of any compartment, over all time. The problem has been considered in both settings, (a) *offline* in which the data items are known in advance and (b) *online* in which the data items arrive in some arbitrary odder and an algorithm must assign a newly arrived item to some compartment without any knowledge about the items that will arrive next.

To solve the offline optimization problem the authors of [52] solved the following decision problem: given a set of n items $x_1, x_2, \ldots, x_n$ does there exist an assignment of the items to λ compartments such that the maximum load of any compartment is at most C? The authors gave a polynomial time algorithm for this decision problem for the special case when all items have weight equal to 1 and using that algorithm as a subroutine, gave a 2-approximation algorithm for the optimization problem.

For the online problem, the paper [52] gave a $(2\lambda - 1)/(\lambda - 1)$-competitive algorithm and an optimal 2-competitive algorithm for the special case of $\lambda = 2$, *i.e.,* when there are only two compartments.

The authors [52] also treated the storage management problem to be an instance of a generalized graph coloring problem which they called as *fragmented coloring* and gave some graph theoretic results which we discussed in Section 2.1.2.

2.2.1 Remarks

The decision version of the offline storage management problem is related to the component coloring problem as follows. Given an instance of the storage problem, create a graph G with a vertex for each item. There is an edge between two vertices if there is a time instant when both the corresponding items are live. Then there is an YES answer to the storage problem if and only if there is a (λ, C)-coloring of G. Note that since all items are active for exactly T days, G is a proper interval graph.

However, fragmented coloring and hence the storage management problem seek to optimize an objective which is somewhat dual to the objective of component coloring. Particularly, fragmented coloring is to minimize the weight of the largest chromon given a fixed set of colors whereas component coloring is to minimize the number of colors while making sure that the weight of the largest chromon is at most C.

Thus the only thing that is common to both our work and the work of [52] is that both give a polynomial time algorithm to decide if there is a (λ, C)-coloring for given PIG G and fixed λ, C. Both algorithms use a fact that solving this decision problem is equivalent to deciding if the vertex set of G can be partitioned such that each part contains vertices consecutive in the order of the left endpoints in the corresponding interval representation of G. However, the difference is that the paper [52] solves this partition problem using a naive approach making the overall algorithm take super quadratic time whereas we exploit a nice structure of PIGs to give a linear time algorithm.

2.3 Light-trail Scheduling

We discussed about the light-trail scheduling problem in Section 1.1 of Chapter 1. Here we discuss the previous work on the problem and other related problems available in the literature.

2.3.1 Hardware Models

After the light-trail technology was introduced by Chlamtac and Gumaste [18], a variety of hardware implementations have emerged. Our model of the problem is based on the implementation in [37]. Chlamtac and Gumaste[36] also introduced a mesh implementation of light-trails for general networks. Gumaste, Kuper, and Chlamtac [40] implemented a tree-shaped variant of light-trails, called as clustered light-trail, for general networks. The paper [99] describes a 'tunable light-trail' in which the hardware at the beginning works just like a simple light-path but can be tuned later to act as a light-trail. There is some preliminary work on multi-hop light-trails [41] in which transmissions are allowed to go through a sequence of overlapping light-trails. Survivability in the case of failures is considered in [7] by assigning each transmission request to two disjoint light-trails.

2.3.2 Stationary Problems

A variety of performance objectives have been proposed. Several objectives are mentioned in the seminal paper [37] – to minimize the total number of light-trails used, to minimize queuing delay, to maximize network utilization etc. Most of the work in the literature seems to solve the problem by minimizing the total number of light-trails used [6, 25, 38, 98]. Though the paper [38] suggests that minimizing the total number of light-trails also minimizes the total number of wavelengths, it may not always be true. For example, consider an input instance in which there are 3 transmissions – transmissions $(1, 2)$ and $(3, 4)$ each with bandwidth requirement 0.5 and transmission $(2, 3)$ with bandwidth requirement 1. To minimize the total number of light-trails used, we create two light-trails on two different wavelengths. Both light-trails extend all the way from 1 to 4. Transmission $(2, 3)$ is put in one light-trail and transmissions $(1, 2)$ and $(3, 4)$ are put in the other light-trail. On the other hand, to minimize the total number of wavelengths, we put each of them in a separate light-trail, and the three light-trails are created on a single wavelength. We believe that minimizing the number of light-trails is motivated by the goal of minimizing the book-keeping and the scheduler overhead. However, we do not think this can be more important than reducing the number of wavelengths needed (or reducing the slowdown the system will face if the number of wavelengths is fixed). There are a few other models as well, *i.e.*, the paper [8] minimizes the total number of transmitters and receivers used in all light-trails.

2.3.3 Solution Techniques

ILP Based Solutions

The general approach followed in the literature to solve the stationary problem is to formulate the problem as an integer linear program (ILP) and then to solve the ILP using standard solvers. The papers [25, 38] give two different ILP formulations.

This ILP formulation assumes that a large number of wavelengths are available and hence all transmission requests can be satisfied. The light-trails for different wavelengths are treated as distinct light-trails and accordingly the set LT is determined.

However, solving these ILP formulations takes prohibitive time even with moderate problem size since the problem is NP-hard. To reduce the time to solve the ILP, the paper [6] removed some redundant constraints from the formulation and added some valid-inequalities to reduce the search space. However, the ILP formulation still remains difficult to solve.

Heuristic Solutions

Heuristics have also been used. The paper [6] solves the problem in a general network. It first enumerates all possible light-trails of length not exceeding a given limit. Then it creates a list of eligible light-trails for each transmission and a list of eligible transmissions for each light-trail. Transmissions are allocated in an order combining descending order of bandwidth requirement and ascending order of number of eligible light-trails. Among the eligible light-trails for a transmission, the one with higher number of eligible transmissions and higher number of already allocated transmissions is given preference. The paper [98] used another heuristic for the problem in a general network. For a ring network, [38] used three heuristics.

For the problem on a general network, [7] solves two sub-problems. The first sub-problem considers all possible light-trails on all the available wavelengths as bins and packs the transmissions into compatible bins with the objective of minimising total number of light-trails used. The second sub-problem assigns these light-trails to wavelengths. The first sub-problem is solved using three heuristics and the second problem is solved by converting it to a graph coloring problem where each node corresponds to a light-trail and there is an edge between two nodes if the corresponding light-trails conflict with each other.

The papers [34, 63] gave heuristics for the problem of integrated scheduling at the applications level for tasks running on light-trail based networks.

2.3.4 Online Problems

For the online problem, a number of models are possible. From the point of view of the light-trail scheduler, it is best if transmissions are not moved from one light-trail to another during execution, which is the model we use. It is also appropriate to allow transmissions to be moved, with some penalty. This is the model considered in [38, 60], where the goal is to minimize the penalty, measured as the number of light-trails constructed. The distributions of the transmissions that arrive are also another interesting issue. It is appropriate to assume that the distribution is fixed, as has been considered in many simulation studies including our own. For our theoretical results, however, we assume that the transmission sequence can be arbitrary. The work in [38] assumes that the traffic is an unknown but gradually changing distribution. It uses a stochastic optimization based heuristic which is validated using simulations. The paper [6] considers a model in which transmissions arrive but do not depart. Multi-hop problems have also been considered [100]. An innovative idea to assign transmissions to light-trails using *online auctions* has been considered in [39]. The paper [42] gives a two-stage scheduling algorithm using heuristics based on the utility of each light-trail and estimates performance of the algorithm in terms of average delay and number of required light-trails by modeling a Markov chain.

2.3.5 Remarks

As may be seen, there are a number of dimensions along which the work in the literature may be classified: the network configuration, the kind of problem attempted, and the solution approach. Network configurations starting from simple linear array/rings [38, 60] to full structured/unstructured networks [6, 7, 25, 89, 98, 100] have been considered in the optical communication literature. The stationary problem as well as the dynamic problem has been considered, with additional minor variations in the models. Finally, three solution approaches can be identified. First is the approach in which scheduling is done using exact solutions of Integer Linear Programs [6, 25, 38]. This is useful for very small problems. For larger problems, using the second approach, a variety of heuristics have been used [6, 7, 38, 98]. The evaluation of the scheduling algorithms has been done primarily using simulations. The third approach could be theoretical. We see no theoretical analysis of the performance of the scheduling algorithms available in the light-trail literature.

In contrast, our main contribution is theoretical. We give algorithms with *provable* bounds on performance, both for the stationary and the online case. Our work uses the competitive analysis approach [1, 14] for the online problem. We use techniques of approximation algorithms to solve the stationary problem. To our knowledge, this competitive analysis and approximation algorithm approach to solve the light-trail scheduling problem has not been used in the literature. We also give simulation results for the online algorithms.

2.4 Scheduling in Reconfigurable Bus Architectures

Our problem as formulated is also similar to the problem of scheduling communications on reconfigurable bus architectures [13, 24, 67, 94]. Many models of reconfigurable bus architectures have been proposed and studied – Reconfigurable Networks (RN) [9], Bus Automation [82], Configurable Highly Parallel Computer (CHiP) [87], Content Addressable Array Parallel Processor (CAAPP) [95], Reconfigurable Mesh (RMESH) [44], Reconfigurable Buses with Shift Switching (REBSIS) [57], Reconfigurable Multiple Bus Machine (RMBM) [90], Distributed Memory Bus Computer (DMBC) [84], Mesh With Reconfigurable Bus (M_r) [79], Polymorphic Processor Array (PPA) [64], Processor Array with Reconfigurable Bus System (PARBS or PARBUS) [46] and others. Models using optical buses have also been proposed – Optical Communication Parallel Computer (OCPC) [33], Array with Reconfigurable Optical Bus (AROB) [78], Linear Array with Reconfigurable Pipelined Bus System (LARPBS) [76] and so on. Though these models use optical signals instead of electrical signals for communication, at an abstract level they are equivalent with PARBS and its variants. So we use the generic term *reconfigurable bus system* to denote all the models mentioned above.

A reconfigurable bus system is modeled as a graph in which processors are vertices and edges are communication links; however, a processor can choose to electrically connect (or keep separate) the communication links incident to it. If links are connected together (like setting the shutter ON), the communication goes through (as well as being read by the processor). In this way the entire network can be made to behave like a few long or many short buses, as per the needs of the application running on the network.

At an abstract level, the reconfigurable bus system is similar to our light-trail model, as both models use controllable switches to dynamically reconfigure a bus into multiple subbuses. In both models, changing the state of the switch takes very long as compared to the data rates on the buses. However, typically, reconfigurable bus systems have only one bus, rather than allowing multiple wavelengths like the light-trail model. A second difference is in the context in which the two models have been studied. The light-trail model has been studied more by the optical network community, and the focus has been how to schedule relatively long duration communication requests (connection based) without having any graphical regularity. Reconfigurable bus systems have been studied more in the context of parallel computing, and the analyses have been more of entire algorithms running on them. These analyses typically concern short messages and the communication patterns are often regular, such as those arising in finding maximum/OR/XOR of numbers [54, 66], matrix multiplication [55], prefix computation [66, 68], problems on graphs [89, 93], sorting [75] and so on. PRAM simulation on reconfigurable bus [56, 92], particularly in the case of randomized assignment of shared memory cells, generates random communication patterns. However, because these patterns are drawn from a uniform distribution, they end up being quite regular (and much of the analysis is to find regular patterns that are supersets of what is required). So even this work does not consider truly arbitrary/irregular patterns which are our prime interest, for the online as well as off-line (stationary) scenarios.

It is interesting to note that the communication patterns for PRAM simulation are uniformly random across the network because the PRAM address space is *hashed*, *i.e.,* distributed randomly. Hashing has the effect of converting possibly local communication going a short distance to a random communication which most likely goes long distance. Such a strategy is inherently wasteful in utilization of bandwidth. It seems much better to directly deal with the arbitrary communication pattern which arises in PRAM simulation in the first place.

2.4.1 Remarks

While there is much work in the reconfigurable bus literature, it mostly concerns *regular* interconnection patterns, such as those arising in matrix multiplication, list ranking and so on [55, 75, 89, 93]. The only work we know of dealing with random communication patterns is in relation to the PARBUS architecture. Such patterns are handled using standard techniques such as Chernoff bounds [80]. We do not know of any work which discusses how to schedule arbitrary irregular communication patterns in this setting. This is probably understandable because reconfigurable bus architectures have mostly been motivated as special purpose computers, except for the PRAM simulation motivation of PARBUS where the communication becomes random.

However, if the network is used for general purpose computing, it does make sense to have algorithms to provision bandwidth for arbitrary irregular patterns, as we do here.

Chapter 3

Stationary Problems on Proper Interval Graphs

In this chapter we focus on solving the stationary component coloring problems on interval graphs which arise in scheduling light-trails on path networks. A graph is an interval graph if the vertices can be represented as intervals on the real-line such that two vertices are connected by an edge if the corresponding intervals intersect. In fact, in this chapter we will solve the stationary problems on a subclass of interval graphs known as proper interval graphs (PIGs). An interval graph is also a PIG if in the corresponding set of intervals no interval is a proper subset of another interval.

We assume that the input PIG, $G = (V, E)$ with n vertices and m edges is simple, finite. We also assume that an interval representation of G is available where the intervals have integer endpoints. In the corresponding light-trail scheduling problem each interval represents a set of consecutive processors, each represented by an integer. We further assume that G is connected. If G is not connected, the algorithms in this chapter can be applied separately to each connected component of G to get an overall solution for G.

Our first solution is for the unweighted problem on PIGs. However, to build up the solution we first show in Section 3.2 that component coloring on chordal graphs is equivalent to a vertex *partitioning* problem which we formally define in Section 3.1. Note that interval graphs are chordal. Later in Section 3.3 we show that for PIGs, it is enough to solve a simpler version of the partitioning problem which we call the *block-partitioning* problem. For block-partitioning we give an LP based algorithm in Section 3.4 and a direct combinatorial algorithm without the need of LP scaffolding in Section 3.5, thus establishing the following result.

Theorem 3.1. *There exists an algorithm to solve the stationary unweighted component coloring problem on a PIG with n vertices and a given interval representation in $O(n)$ time.*

In Section 3.6 we extend the algorithm for the unweighted problem to solve the splittable weighted problem on PIGs, thus establishing our second result of this chapter.

Theorem 3.2. *There exists an algorithm to solve the stationary splittable weighted component coloring problem on a PIG with n vertices and a given interval representation in $O(n^2M/C)$ time where M is the maximum weight of a vertex.*

Considering the applications in scheduling light-trails on path networks, we may assume that the bandwidth requirement of a transmission request is at most the capacity of wavelength, *i.e.,* weight of a vertex is at most C. In that case the algorithm mentioned in Theorem 3.2 takes time $O(n^2)$.

In the nonsplittable problem, however, by definition, a vertex has weight at most C. Using the algorithm for the splittable problem as a subroutine we devise a two factor approximation algorithm for the NP-hard nonsplittable problem PIGs in Section 3.7, thus establishing our final result of this chapter.

Theorem 3.3. *There exists a 2-approximation algorithm to solve the stationary nonsplittable weighted component coloring problem on a PIG with n vertices and a given interval representation in $O(n^2)$ time.*

3.1 (λ, C)-partition

The component coloring problem can be seen as solving two problems simultaneously, (i) partitioning the vertex set into parts each of which will eventually form a monochromatic component (chromon) after coloring, and (ii) assigning colors to these parts. The partitioning should be such that if each part is contracted to a single vertex, the resulting graph can be colored using as few colors as possible. Before we formally define the partitioning problem, we give the following definitions.

For a set $S \subseteq V$, the subgraph of $G = (V, E)$ *induced by* S, denotes by $G[S]$ is the graph $G[S] = (S, E(S))$ where $E(S) = \{(u, v) \in E \mid u, v \in S\}$. The contraction of an edge $e = (u, v)$ of a graph $G = (V, E)$ is to remove e from E and to replace $u, v \in V$ by a new vertex w such that w is adjacent to all vertices that were adjacent to either u or v. For a graph

$G = (V, E)$ and a partition $\Pi = \{P_1, P_2, \ldots, P_t\}$ of V, $P_i \subseteq V$, $P_i \cap P_j = \varnothing$ for all $i \neq j$, the *partition chromatic number* of Π is the chromatic number of the graph obtained by contracting each class P_i into a single vertex.

Definition 3.4. *A graph $G = (V, E)$ is said to have a (λ, C)-*partition *if and only if there is a partition $\Pi = \{P_1, P_2, \ldots, P_t\}$ of V such that the following constraints are satisfied:*

- connectedness – *the subgraph induced by each part P_i,* i.e., *$G[P_i]$ is connected,*
- size – *each part P_i has at most C vertices, and*
- partition chromatic number *of Π is at most λ.*

A *C-component partition* of a graph is a (λ, C)-partition with the minimum λ. We will refer to the problem of finding a C-component partition as the *partition* problem.

The size of the largest clique $\omega(G)$ in a graph G plays a major role in determining the chromatic number $\chi(G)$, at least for some graphs classes such as perfect graphs. In fact, in a perfect graph G each induced subgraph $H \subseteq G$ satisfy $\chi(H) = \omega(G)$. The *clique intersection* of a partition Π of G is the maximum number of parts of Π intersected by any clique in G. Thus for a perfect graph the partition chromatic number matches the clique intersection and hence an equivalent definition of (λ, C)-partition on perfect graphs is as follows.

Definition 3.5. *A perfect graph $G = (V, E)$ has a (λ, C)-partition if and only if there is a partition $\Pi = \{P_1, P_2, \ldots, P_t\}$ of V that satisfies connectedness constraint, size constraint as well as* clique intersection *constraint,* i.e., *the clique intersection of Π is at most λ.*

We will use the above definition as the class of PIGs is a subclass of chordal graphs which again is a subclass of perfect graphs. The stationary component coloring problem is equivalent to the partition problem on chordal graphs which we discuss in the next section.

3.2 Coloring $\equiv$ Partition on Chordal Graphs

Before we prove the equivalence, let us recall the definition of (λ, C)-coloring given in Section 1.2 of Chapter 1. A (λ, C)-coloring of a graph G is the assignment of λ colors to the vertices of G such that every monochromatic component in the coloring has size at most C.

Lemma 3.6. *If a graph G has a (λ, C)-coloring then it has a (λ, C)-partition.*

Proof. Suppose G has a (λ, C)-coloring $\mathcal{C}$. Consider the partition Π induced by $\mathcal{C}$ where each part is exactly a chromon. The connectedness constraint is immediately satisfied. The coloring $\mathcal{C}$ itself shows that partition chromatic number of Π is at most λ. Since a chromon has size at most C, the size constraint is also satisfied. Hence, Π is a (λ, C)-partition. $\square$

Next we will show that for chordal graphs the converse is also true. A graph is *chordal* if each of its cycles of four or more vertices has a *chord*, which is an edge joining two vertices that are not adjacent in the cycle. There are many characterizations of chordal graphs (see [35] for more details). We will use the characterization of a chordal graph based on *perfect elimination ordering* or, in short, *PEO*. The open neighborhood $N(v)$ of v is the set of vertices adjacent to u, *i.e.,* $N(v) = \{u \mid (u, v) \in E\}$, and the closed neighborhood $N[v]$ of v includes v as well as all vertices in $N(v)$. A vertex v of G is called *simplicial* if $N[v]$ forms a clique. An ordering $\sigma = [v_1, v_2, \ldots, v_n]$ of vertices is a PEO if each vertex v_i is a simplicial vertex of the induced subgraph $G[v_i, \ldots, v_n]$.

Proposition 3.7 ([35])**.** *Let $G = (V, E)$ be an undirected graph. Then G is a chordal graph if and only if G has a PEO. Moreover, any simplicial vertex can start a PEO.* $\square$

To complete the proof of the equivalence of component coloring and partition on chordal graphs we need to prove the following lemma. Since each chordal graph is also perfect, we use the definition of a (λ, C)-partition given by Lemma 3.5.

Lemma 3.8. *If a chordal graph G has a (λ, C)-partition then it has a (λ, C)-coloring.*

Proof. Suppose G has a (λ, C)-partition $\Pi = \{P_1, P_2, \ldots, P_t\}$. We prove that there exists a (λ, C)-coloring of G in which each P_i is a chromon. Let the colors be numbered $1, 2, \ldots$. We prove by induction on number of vertices n. For $n = 1$, assigning color 1 to the single vertex gives a (λ, C)-coloring for any $\lambda, C \geqslant 1$.

For $n > 1$, let u be a simplicial vertex of G. Without loss of generality assume $u \in P_1$. Consider the graph G' obtained by removing u from G. Then $\Pi' = \{P_1 \backslash \{u\}, P_2, \ldots, P_t\}$ is a (λ, C)-partition for G'. By induction, there is a (λ, C)-coloring $\mathcal{C}'$ of G' in which each part of Π' is a chromon. We obtain a coloring $\mathcal{C}$ of G as follows. If $|P_1| > 1$ we assign the color of other vertices in P_1 to u too. Otherwise we assign u the lowest numbered color that is not assigned to $N(u)$ in $\mathcal{C}'$. To show that $\mathcal{C}$ is a (λ, C)-coloring, it is enough to show that at most λ colors are used in $\mathcal{C}$. For $|P_1| > 1$ it is obvious as no new color is used. For $|P_1| = 1$ if it requires $\lambda + 1$ colors then it implies that the clique $N[u]$ associated with the simplicial vertex u intersects $\lambda + 1$ parts which is not possible. □

Thus, on chordal graphs, solving the component coloring problem is equivalent to solving the partition problem. In the rest of this chapter we solve the partition problem only because the solution can be converted to a solution to the coloring problem using the procedure described in the proof of Lemma 3.8.

3.3 Coloring ≡ Block-partition on PIGs

For PIGs, we introduce a more restricted way of partitioning the vertex set which we call *block-partitioning*. Before we formally define block-partitioning we need to introduce an important property of PIGs.

A graph $G = (V, E)$ is an *interval graph* if there exists a family $\mathcal{I} = \{I_v \mid v \in V\}$ of intervals on a real line such that for distinct vertices u, v in G, $(u, v) \in E$ if and only if $I_u \cap I_v \neq \emptyset$. Such a family $\mathcal{I}$ of intervals is commonly referred to as the *interval representation* of G. Given an interval representation of G, consider a cycle of more than 3 vertices, and the corresponding intervals in ascending left endpoints. Since the rightmost interval intersects the leftmost interval, it also intersects the intervals in between them. Hence G is also chordal. It will be convenient to let $Left(I_v)$ and $Right(I_v)$ stand for the left and right endpoint of the interval I_v, respectively. The family $\mathcal{I}$ is the interval representation of a *proper interval graph (PIG)* if and only if no interval is properly contained in another.

We will see that our algorithms need an interval representation of the input graph. There exists an $O(m+n)$ time algorithm [72] to get an interval representation of a given interval graph. Nevertheless, since the component coloring problem is motivated by the light-trail scheduling problem, in this thesis, without loss of generality, we assume that an interval representation $\mathcal{I} = \{I_v \mid v \in V\}$ is given for the input PIG $G = (V, E)$. In that way we can ignore the additive $O(m)$ factor and our algorithms only depend on n. Furthermore, we assume that the endpoints of the intervals are integers as the intervals are supposed to represent a set of consecutive processors in the light-trail scheduling problem where each processor is represented by an integer.

Now consider the linear order $\prec$ on V defined as follows. For $u, v \in V$, $u \prec v$ if and only if $Left(I_u) < Left(I_v)$ or $[(Left(I_u) = Left(I_v))$ and $(Right(I_u) \leqslant Right(I_v))]$. We call this ordering $v_1 \prec v_2 \prec \cdots \prec v_n$ the *canonical ordering*. In the rest of this chapter, we use numbers 1 to n to represent the vertices where i represents the ith vertex in the canonical ordering. Hence, v will be interchangeably used to represent a vertex $v \in V$ as well as its position in the canonical ordering. If $u \prec v$ then u is said to be on the left of v and v is said to be on the right of u.

Proposition 3.9 ([72]). *A graph $G = (V, E)$ is an interval graph if and only if there exists a linear order $\prec$ on V such that for every choice of vertices u, v, w with $u \prec v \prec w$, $(u, w) \in E$ implies $(u, v) \in E$.* □

For PIGs the canonical ordering not only satisfies the conditions in Proposition 3.9 but, in fact, satisfies a stronger property:

Proposition 3.10 ("The Umbrella Property" [61]). *A graph $G = (V, E)$ is a PIG, if and only if, there exists a linear order $\prec$ on V such that for every choice of vertices u, v, w, with $u \prec v \prec w$, $(u, w) \in E$ implies both $(u, v) \in E$ and $(v, w) \in E$.* □

An immediate corollary of Proposition 3.10 is that every edge $(u, v) \in E$ induces a clique $[u, v]$. Also, any maximal clique of a PIG can be represented by a single edge between the two end vertices, say (u, v), or by the block $[u, v]$.

Corollary 3.11. *Let S be a connected subgraph of a PIG and $v_1, v_2, \ldots, v_t$ be the vertices of S arranged in the canonical ordering. Then there must be edges (v_i, v_{i+1}) for all $i = 1, \ldots, t-1$.*

Proof. Consider the two vertices v_i and v_{i+1}. Since S is connected there must be an edge (v_j, v_k) where $j \leqslant i$ and $i + 1 \leqslant k$. Then $[v_j, v_k]$ is a clique. Thus there is an edge (v_i, v_{i+1}). □

A *block*[1] in a PIG a is a set of vertices which are consecutive in the canonical ordering. We will represent a block starting at a vertex u and ending at a vertex v as the interval $[u, v]$. Now we are all set to formally define block-partitioning on PIGs.

Definition 3.12. *A PIG is said to have a (λ, C)-*block partition *if it has a (λ, C)-partition in which each part also satisfy* consecutiveness *constraint,* i.e., *each part is also a block.*

Lemma 3.13. *A PIG G has a (λ, C)-partition if and only if G has a (λ, C)-block partition.*

Proof. A (λ, C)-block partition is also a (λ, C)-partition. Now suppose G has a (λ, C)-partition Π. If the parts in Π also satisfy the consecutiveness constraint, we are done. Otherwise we

[1]Some authors use the term block to represent what we call a clique.

convert Π to a new partition Π' that also satisfies the consecutiveness constraint. The conversion is done by exchanging vertices among the parts in Π, step-by-step, as follows.

We call a vertex u to be *terminal* if u and some $v > u + 1$ belong to one part but $u + 1$ belongs to a different part, *non-terminal* otherwise. Let P_1 be the leftmost part whose vertices are not consecutive. Let $i \in P_1$ be smallest terminal vertex such that $i + 1$ is in some $P_2 \neq P_1$, and there exists $i + k \in P_1$ for some $k > 1$. We will show how to repartition $P = P_1 \cup P_2$ into parts P_1' and P_2' such that in the new partition, each vertex in the range $[1, i]$ is non-terminal. Then by repeating this process all vertices can be made non-terminal and hence consecutiveness constraint will be satisfied. Note that P is connected as both P_1, P_2 are connected and P_2 has a vertex in between two vertices of P_1. There are two cases.

Case 1: There are at most C vertices in P to the right of i. In this case we set P_2' to be the vertices in P to the right of i, and the P_1' to be the vertices in P to the left of and including i. Clearly, i is no more a terminal vertex. Since P is connected, the vertices of P considered in the canonical ordering form a path. P_1', P_2' are formed by breaking this path in the middle, so P_1', P_2' are both connected. Let Q be any maximal clique which intersects P_1', P_2'. Since we know that the vertices of Q are consecutive, and i is the rightmost vertex in P_1' and $i + 1$ the leftmost vertex in P_2', the vertices $i, i + 1$ must be in Q. Thus Q intersects P_1, P_2 as well. All other parts intersecting Q remain unchanged, so the number of parts intersected by Q is the same in the new partition as the old.

Case 2: There are more than C vertices in P to the right of i. In this case we set P_2' to be the C rightmost vertices in P, and the remaining go to P_1'. As before we see that i is no more a terminal vertex and P_1', P_2' satisfy the connectedness property. Consider a maximal clique Q that intersects P_1', P_2'. We show that it must intersect the same number of parts in the new partition as the old. Q must contain the rightmost vertex u of P_1' and leftmost vertex v of P_2'.

Note first that P_1' contains both $i, i + 1$, *i.e.,* it has at least one vertex from P_1 and one vertex from P_2. But P_2' has C vertices, so they cannot all be from P_1, or all from P_2 because both P_1, P_2 had at most C vertices each. Thus P_2' also contains at least one vertex j from P_1 and one vertex k from P_2. Since $i, j \in P_1$, there must be a path in P_1 from i to j. There must exist an edge (u', v') in this path such that $u' \leqslant u$, and $v \leqslant v'$ (see Figure 3.1). Since Q is maximal, it must contain u', v'. Thus Q intersects P_1. In a similar manner, we see that it must intersect P_2. Thus it follows that Q intersects the same number of parts in the old and new partitions. □

Figure 3.2 shows a simple example of a general (non-proper) interval graph where Lemma 3.13

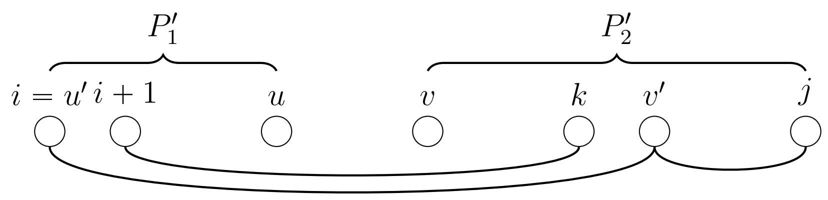


Figure 3.1: Sketch Showing Clique Intersection Remains Unchanged in Exchange

does not work. The graph is given by the intervals in canonical ordering: $a = [1,9], b = [2,5], c = [3,6], d = [4,12], e = [7,10], f = [8,11]$. It has two maximal cliques $Q_1 = \{a,b,c,d\}, Q_2 = \{a,d,e,f\}$. For $C = 2$, the optimal partition $\{\{a,d\},\{b,c\},\{e,f\}\}$ has clique intersection 2 but the part $\{a,d\}$ is not a block as a,d are not consecutive according to canonical ordering. All block partitions have clique intersection 3 or more. The reason is as follows. If a is the only vertex in a part then to cover the remaining 3 vertices of Q_1 we need at least 2 more parts. On the other hand if a is paired with b or c then to cover the remaining 3 vertices of Q_2 we need at least 2 more parts.

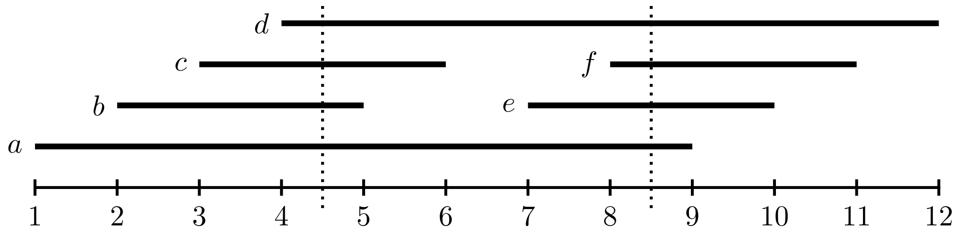


Figure 3.2: Example Showing Lemma 3.13 not Valid for General Interval Graphs

Since for a PIG, the notions of partition and block partition are equivalent, in the rest of this chapter we will abuse the notation (λ, C)-partition to actually mean a (λ, C)-block partition in the context of PIGs. The following three lemmas will be useful for our algorithms for the partition problem given in subsequent sections.

Proposition 3.14. *If an interval representation with integer endpoints is given for a PIG G, then the vertices of G can be arranged in canonical ordering in $O(n)$ time.*

Proof. Sort the vertices in ascending left endpoints of the intervals using bucket sort. Since there are n integer endpoints, bucket sort takes $O(n)$ time. We claim that after sorting the vertices are already in canonical ordering. The proof is as follows.

Consider three vertices u, v, w in the sorted order, and the corresponding intervals $[u_1, u_2]$, $[v_1, v_2]$, $[w_1, w_2]$. We have $u_1 \leqslant v_1 \leqslant w_1$. Since intervals for u, v are not proper subset of each other, it follows that $u_2 \leqslant v_2$. Similarly $v_2 \leqslant w_2$.

Now if there is an edge (u, w) then $w_1 \leqslant u_2$. Since $v_1 \leqslant w_1$ and $u_2 \leqslant v_2$ we have $v_1 \leqslant u_2$ and $w_1 \leqslant v_2$. Thus there are edges (u, v) and (v, w). □

Lemma 3.15. *Given a PIG G with an interval representation with integer endpoints, if there is an $O(f(n))$ algorithm to solve the (block) partition problem on G, then there is an $O(n + f(n))$ algorithm to solve the coloring problem on G.*

Proof. We first get the canonical ordering of by sorting the left endpoints of the intervals as mentioned in Proposition 3.14. Suppose the partition algorithm returns a partition with clique intersection λ and the parts sorted in canonical ordering are $P_1, P_2, \ldots, P_t$. For each $1 \leqslant i \leqslant t$, we assign color $(i - 1) \bmod \lambda + 1$ to P_i. This is a valid coloring because otherwise, there is an edge (u, v) between two parts of the same color implying that the clique $[u, v]$ in G intersects more that λ parts; this is not possible in a (λ, C)-partition. □

Proposition 3.16. *If an interval representation is given for a PIG G, then the maximal cliques of G can be found in $O(n)$ time.*

Proof. Let $\mathcal{I} = \{I_v \mid v \in V\}$ be an interval representation of G. Without loss of generality we assume that the endpoints of all intervals are unique. Otherwise we can suitably extend some of the intervals on either side so that all endpoints become distinct without altering the maximal cliques. We construct the sorted array $A(1, \ldots, 2n)$ of all endpoints in $O(n)$ time using bucket sort. The maximal cliques are identified as follows. Traverse A left to right and whenever $A(i)$ is $Left(I_v)$ and $A(i + 1)$ is $Right(I_u)$ for some u, v in V then output $[u, v]$. Clearly u and v are adjacent and hence $[u, v]$ is a clique. Since u is the leftmost possible and v is the rightmost possible for such a clique, $[u, v]$ is a maximal clique. The traversal takes $O(n)$ time. □

3.4 An LP Based Algorithm

Let $G = (V, E)$ be a PIG with vertices already sorted in canonical ordering and $\mathcal{Q}$ be the set of maximal cliques. Let $Left(Q)$ denote the leftmost vertex of Q. Then the partition problem on

G can be formulated as the following Integer Linear Program:

$$
\begin{aligned}
\textsc{IlpPart}: \quad \min \quad & \lambda \\
\text{s.t.} \quad & x_n = 1 && && (3.1)\\
& \sum_{j=i}^{i+C-1} x_j \geqslant 1 && 1 \leqslant i \leqslant n - C + 1 && (3.2)\\
& \sum_{j=Left(Q)}^{Left(Q)+|Q|-2} x_j \leqslant \lambda - 1 && \forall\, Q \in \mathcal{Q} && (3.3)\\
& x_j \in \{0,1\} && 1 \leqslant j \leqslant n && (3.4)\\
& \lambda \quad \text{integer} && && (3.5)
\end{aligned}
$$

where x_j is a binary variable to denote if vertex j is the rightmost vertex of a block and λ denotes the maximum clique intersection. Constraint (3.1) ensures that some block must end at n. Constraints (3.2) ensure that among C consecutive vertices there must be at least one vertex which is the rightmost vertex of a block because a block has size at most C. Since a clique Q intersects at most λ blocks, constraints (3.3) ensure that the vertices in Q, except the rightmost, can include the rightmost vertices of at most $\lambda - 1$ blocks. The objective is to minimize the maximum clique intersection λ.

Let LpPart be the LP relaxation of IlpPart obtained by making x_j a real variable in $[0,1]$ and making λ unconstrained. LpPart can be written in the standard form: $\min c^T z$ such that $Az \geqslant b, z \geqslant 0$. All rows of the $0,1$ matrix A has consecutive 1s. Hence A is totally unimodular and it implies that LpPart has an integer optimal solution [69, Chapter III.1]. However, we give an alternative proof using rounding.

Lemma 3.17. *If x, λ is a fractional solution to* LpPart *then it can be rounded to an integer feasible solution $\bar{x}, \bar{\lambda}$ in polynomial time where $\bar{\lambda} \leqslant \lambda$.*

Proof. Consider the following rounding scheme which takes $O(n)$ time. We use a set of intermediate variables $y_0, y_1, \ldots, y_n$. We set

$$
y_0 = 0, \quad y_j = \sum_{i=1}^{j} x_i, \quad \bar{\lambda} = \lfloor \lambda \rfloor \quad \text{and} \quad \bar{x}_j = \begin{cases} 1 & \text{if } \lceil y_{j-1} \rceil \neq \lceil y_j \rceil \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } 1 \leqslant j \leqslant n.
$$

Note that each $\bar{x}_j$ is a 0-1 variable and $\bar{\lambda}$ is an integer. Since $x_n = 1$ and $\lceil y_{n-1} \rceil \neq \lceil y_n \rceil$, by construction $\bar{x}_n = 1$. Hence $\bar{x}$ satisfies constraint (3.1).

Now we prove that $\bar{x}$ satisfies the constraints in (3.2). Since x satisfies jth of such constraints, $x_j + x_{j+1} + \ldots + x_{j+C-1} \geqslant 1$, *i.e.*, $y_{j+C-1} - y_{j-1} \geqslant 1$. So there must be at least one index k in $[j, j+C-1]$ such that $\lceil y_{k-1} \rceil \neq \lceil y_k \rceil$ implying that $\bar{x}_k = 1$. Thus $\bar{x}$ also satisfies the jth constraint in (3.2).

Finally we prove that $\bar{x}, \bar{\lambda}$ satisfy constraints in (3.3) too. Consider the constraint for clique Q and let $j = Left(Q)$. Since x satisfies this constraint, $x_j + x_{j+1} + \ldots + x_{j+|Q|-2} \leqslant \lambda - 1$, *i.e.*, $y_{j+|Q|-1} - y_j \leqslant \lambda - 1$. So there can be at most $\lambda - 1$ indices k in $[j+1, j+|Q|-1]$ such that $\lceil y_{k-1} \rceil \neq \lceil y_k \rceil$ implying that at most $\lfloor \lambda \rfloor - 1$ of the corresponding $\bar{x}_k$s are set to 1. Thus $\bar{x}, \bar{\lambda}$ also satisfy the constraint for clique Q. □

Clearly $\bar{\lambda} \leqslant \lambda$. Since the integer objective value $\bar{\lambda}$ cannot be strictly less than the fractional value λ, $\bar{\lambda} = \lambda$ and hence $(\bar{x}, \bar{\lambda})$ is an optimal solution to ILPPART. Thus solving LPPART and rounding the solution using the procedure given in the proof of Lemma 3.17 gives a polynomial time algorithm for the partition problem.

3.5 A Combinatorial Algorithm

We now give a combinatorial algorithm for the partition problem on PIGs. The algorithm does not use LP scaffolding and hence is more efficient.

3.5.1 Lower Bound

Lemma 3.18. *If a PIG G has a (λ, C)-partition then $\lambda \geqslant \lfloor (\omega(G) + C - 1)/C \rfloor$.*

Proof. Let Q be a maximum clique of G, *i.e.*, $|Q| = \omega(G)$. To cover all vertices of Q by parts of size at most C, we need at least $\lceil \omega(G)/C \rceil$ parts. Hence, clique intersection $\lambda \geqslant \lceil \omega(G)/C \rceil = \lfloor (\omega(G) + C - 1)/C \rfloor$. □

There are examples where $\lambda = \lceil \omega/C \rceil$ is not enough to have a (λ, C)-partition. Consider the graph given by the intervals $\{[1,3], [2,5], [4,6]\}$. Here $\omega = 2$. For $C = 2$, lower bound $\lceil \omega/C \rceil = 1$. But a single part common to both cliques cannot contain 3 connected vertices.

3.5.2 Upper Bound

Lemma 3.19. *If an interval representation is given for a connected PIG G then there exists an algorithm that produces a $[\lceil (\omega(G) + C - 1)/C \rceil, C]$-partition.*

Proof. Consider the following algorithm which we call SIMPLEPART. We first arrange the vertices of G in the canonical ordering time using Proposition 3.14. Then we assign the block of vertices $[(i-1)C+1, \min\{n, iC\}]$ to part P_i for each $1 \leqslant i \leqslant \lceil n/C \rceil$.

Each part P_i produced by SIMPLEPART clearly has consecutive vertices and has size at most C. Since G is connected, by Corollary 3.11, there is an edge between v_j and v_{j+1} for all $j = 1, 2, \ldots, n-1$. Hence P_i is also connected. Thus it will be enough to show that any clique intersects at most $\lambda = \lceil (\omega + C - 1)/C \rceil$ parts.

If a clique Q intersects λ parts $P_i, \ldots, P_{i+\lambda-1}$, then Q must contain at least one vertex of each of P_i and $P_{i+\lambda-1}$ and all vertices of remaining parts $P_2, \ldots, P_{i+\lambda-2}$. So the minimum size of Q is $1 + (\lambda-2)C + 1 = \lambda C - 2C + 2$. Thus $\omega \geqslant \lambda C - 2C + 2$. Hence $\lambda \leqslant (\omega + 2C - 2)/C$. This implies $\lambda \leqslant \lfloor (\omega + 2C - 2)/C \rfloor = \lceil (\omega + C - 1)/C \rceil$. □

There are examples where SIMPLEPART does not give the optimal partition. Consider the graph given by the intervals $a = [1,6], b = [2,7], c = [3,10], d = [4,11], e = [5,12], f = [8,13], g = [9,14]$. It has two maximal cliques $\{a,b,c,d,e\}, \{c,d,e,f,g\}$ and $\omega = 5$. For $C = 3$, SIMPLEPART produces the partition $\{\{a,b,c\}, \{d,e,f\}, \{g\}\}$ which has clique intersection 3. But there exists a better partition $\{\{a,b\}, \{c,d,e\}, \{f,g\}\}$ with clique intersection 2.

However, a close analysis reveals that SIMPLEPART is not that bad. In fact, when $\omega(G) = kC + 1$ for some integer k, the two bounds match and hence SIMPLEPART gives the optimal solution. For other values of $\omega(G)$, which can be represented as $kC + r$ for integer k, r such that $2 \leqslant r \leqslant C$, the two bounds are $k+1$ and $k+2$ respectively, hence differ by 1 and one of the two bounds is optimum. Thus it will be enough to solve the following special case of the problem.

Problem	Partition subproblem
Given	PIG G with $\omega(G) = kC + r$, k integer and $2 \leqslant r \leqslant C$
Goal	Does G have a $(k+1, C)$-partition? If yes, generate the partition

If there is an algorithm ALG for the partition subproblem then we apply ALG to G to check if there is a $(k+1, C)$-partition. If yes, ALG also gives the required partition. Otherwise, SIMPLEPART gives an optimal solution.

In the rest of this chapter we will let $k(G)$, or in short k, denote $\lfloor (\omega(G) - 1)/C \rfloor$.

3.5.3 Forbidden Vertices

The key idea in our algorithm is to first identify those vertices that cannot be right endpoints of a block in a possible $(k+1, C)$-partition.

Definition 3.20. *A vertex v in a PIG is said to be* primarily forbidden *if the block $[v-kC, v+1]$ is a clique.*

Lemma 3.21. *If the vertex v in a PIG is primarily forbidden then no block in a $(k+1, C)$-partition can end at v.*

Proof. Suppose a block in a $(k+1, C)$-partition ends at the vertex v. Consider the clique $[v-kC, v+1]$ which must be covered by at most $k+1$ blocks. To cover the vertex $v+1$ we need one block. Then the remaining $kC+1$ vertices $[v-kC, v]$ must be covered by at most k blocks. This is not possible as the size of a block is at most C. $\square$

Definition 3.22. *A vertex in a PIG is* forbidden *if it is primarily forbidden or secondarily forbidden, where secondarily forbidden vertices are defined recursively as follows. Suppose there exist v, F, Q such that (a) Q is clique, (b) F is a consecutive sequence of forbidden vertices starting at the rightmost vertex of Q and ending with v, (c) $|Q| + |F| - 1 = kC + 1$, (d) $1 <= |F| <= C - 1$. Then vertices $P(v) = \{v - qC \mid 1 \leqslant q \leqslant k\}$ are* secondarily forbidden. *Further, we will say that v is the* leader *of all secondarily forbidden vertices in $P(v)$ and v itself. Similarly each secondarily forbidden vertex in $P(v)$ is a* follower *of v.*

Note that a primarily forbidden vertex is the leader of itself. Furthermore, any (primarily or secondarily) forbidden vertex, v has a leader $v + qC$ where q is an integer and $0 \leqslant q \leqslant k$.

Lemma 3.23. *If the vertex v in a PIG is secondarily forbidden then no block in a $(k+1, C)$-partition can end at v.*

Proof. Suppose a block ends at the vertex v. By Definition 3.22 the leader of v is $v + qC$ for some $1 \leqslant q \leqslant k$ and there exists a forbidden block F ending at $v + qC$ and a clique Q of size $kC + 2 - |F|$ ending at the leftmost vertex of F and starting at $v + qC - kC$. It will be enough to show that Q intersects at least $k+2$ blocks.

All forbidden vertices in F must be covered by a single block, say B, and B must end on the right of $v + qC$. In the best case B ends at $v + qC + 1$ and covers rightmost $C - |F|$ vertices of Q including the leftmost vertex of F. Among the $(k-1)C + 2$ remaining vertices of Q, the leftmost $(k-q)C+1$ vertices $[v+qC-kC, v]$ require at least $k-q+1$ blocks and $(q-1)C+1$ middle vertices require at least q blocks. Thus overall Q requires at least $k+2$ blocks. $\square$

Our algorithm is as follows. We mark all forbidden vertices, and then try to form blocks by a greedy left to right strategy.

3.5.4 Marking Forbidden Vertices

The algorithm for marking forbidden vertices is given in Algorithm 1. The algorithm assumes that we are given an array Lmn, where $\texttt{Lmn}(v)$ denotes the leftmost neighbor of v. It is easily seen that Lmn can be computed in $O(n)$ time given an interval representation of the input graph. The algorithm constructs the array F, where $F(v) = 1$ if and only if v is forbidden.

Algorithm 1: MARKFORBIDDEN

```
Input  : $\texttt{Lmn}(1, \ldots, n)$ for a PIG $G = (V, E)$
Output: $F(1, \ldots, n)$
1  foreach $v = 1$ to $n$ do  $F(v) = \texttt{Ldist}(v) = 0$;
2  foreach $v = n$ downto $1$ do                                   /* phase 1 */
3      if $\texttt{Lmn}(v) \leqslant v - kC - 1$ then
4          $F(v-1) = 1$;
5  $\texttt{Rnf}(n) = n$;      /* $\texttt{Rnf}(v)$ is rightmost non-forbidden vertex $\leqslant v$ */
6  foreach $v = n - 1$ downto $1$ do                               /* phase 2 */
7      $\texttt{Rnf}(v) = \min\{v, \texttt{Rnf}(v+1)\}$;
8      while $F(\texttt{Rnf}(v))$==1 do $\texttt{Rnf}(v) = \texttt{Rnf}(v) - 1$;  /* now $\texttt{Rnf}(v)$ is proper */
9      if ($F(v)$==1) and ($\texttt{Ldist}(v) \leqslant (k-1)C$) then       /* $v$ is a follower */
10         $F(v-C) = 1$;   $\texttt{Ldist}(v-C) = \texttt{Ldist}(v) + C$;
11     if $\texttt{Lmn}(\texttt{Rnf}(v)+1) \leqslant v - kC$ then                /* $v$ is a leader */
12         $F(v-C) = 1$;   $\texttt{Ldist}(v-C) = C$;
```

The algorithm marks primarily forbidden vertex $v-1$ by checking if v is the rightmost vertex of a clique of size $kC+2$, that is, if $\texttt{Lmn}(v) \leqslant v - kC - 1$.

The algorithm marks secondarily forbidden vertices by checking if a given vertex v is a leader as per Definition 3.22. For this it needs to know whether the forbidden vertex set F and the clique Q exist as per Definition 3.22. Suppose F, Q exist satisfying Definition 3.22. Then suppose that F' is another set of forbidden vertices ending at v, but $|F'| > |F|$. Then we know that $v, F', Q - F'$ will also satisfy Definition 3.22. Thus it suffices to only consider the largest sequence F ending at v. This is what the algorithm does. If the largest sequence F ending at v starts at u then a Q satisfying Definition 3.22 must start at $v - kC$. The algorithm uses this fact.

For marking secondarily forbidden vertices the algorithm maintains the following invariants at the beginning of each iteration in phase 2:

1. All the leaders in the set $[v + 1, n]$ have been marked.
2. All the followers in $[v + 1 - C, n]$ have also been marked.
3. For each follower j in $[v + 1 - C, n]$ its distance from the leader is $\texttt{Ldist}[j]$.
4. $\texttt{Rnf}(v + 1)$ equals the rightmost vertex in the range $[1, v + 1]$ that is neither primarily forbidden nor secondarily forbidden due to the leaders in $[v + 2, n]$.

In each iteration the invariants are extended for the next smaller value of v. We show that the algorithm maintains the invariants by using induction on v.

Since the rightmost vertex n is never forbidden and the algorithm sets $\texttt{Rnf}(n) = n$, all the invariants are maintained at the beginning of the iteration $v = n - 1$.

Suppose the invariants are maintained at the beginning of an iteration. Since $\texttt{Rnf}(v)$ either equals to or is on the left of both v and $\texttt{Rnf}(v + 1)$ the algorithm sets $\texttt{Rnf}(v)$ to the minimum of them in step 7. In step 8 the algorithm moves $\texttt{Rnf}(v)$ further to the left until it gets a non-forbidden vertex. This step is valid because all forbidden vertices, both primarily and secondarily, in the range $[v - C + 1, v]$ are already marked. For the invariants the only things remaining to show are the following: (i) v is correctly marked if it is a leader; the algorithm does that in step 12 by checking the existence of Q for v and $F = [\texttt{Rnf}(v) + 1, v]$ as per Definition 3.22 using the array $\texttt{Lmn}$ (note that by Proposition 3.10 $[\texttt{Lmn}(v), v]$ is a clique), and (ii) $v - C$ is correctly marked if it is a follower of a previously discovered leader l; in that case v is also a follower of l, or a follower of the newly discovered leader v; in that case v is also the leader of itself. In both cases the algorithm correctly sets the distance $\texttt{Ldist}[v - C]$ by adding C to the distance of v from the leader.

The pseudocode of Algorithm 1 clearly shows that MARKFORBIDDEN takes overall $O(n)$ time. Thus we have proved the following result.

Lemma 3.24. *If the array* `Lmn` *for a PIG G is given, the algorithm* MARKFORBIDDEN *correctly marks the forbidden vertices of G in $O(n)$ time.*

3.5.5 Algorithm COMBPART

We now give our algorithm COMBPART, shown in Algorithm 2, to solve the partition subproblem. The algorithm first marks all forbidden vertices and then forms blocks greedily such that no block ends at a forbidden vertex.

Algorithm 2: COMBPART

```
Input  : A PIG $G = (V, E)$
Output: If $G$ has a $(k+1, C)$-partition; if YES also output such a partition
1 $F(1, \dots, n)$ = array returned by MARKFORBIDDEN on $G$;    $u = 1$;
2 while $u \leqslant n$ do
3     $v = \min\{u + C - 1, n\}$;
4     while $(v \geqslant u)$ and $(F(v) == 1)$ do $v = v - 1$;
5     if $v < u$ then return NO else create part $[u, v]$; $u = v + 1$;
6 return YES;
```

Lemma 3.25. *Let B be any block, except the rightmost, created by* COMBPART*. Let u be the leftmost vertex of B. Then the block of vertices $[u + jC + |B|, u + jC + (C - 1)]$ are forbidden for $0 \leqslant j \leqslant k - 1$.*

Proof. If $|B| = C$ then the block $[u + jC + |B|, u + jC + (C - 1)]$ is empty and hence the lemma is vacuously true. So we assume $|B| < C$. Note that $u + |B|, \dots, u + C - 1$ are all forbidden because otherwise COMBPART would have created the block B of bigger size. Also note that either B is the leftmost block or $u - 1$ is the rightmost vertex of a block. So without loss of generality we assume that $u - 1$ is not forbidden.

It will be enough if we prove that for each $|B| \leqslant t \leqslant C - 1$ the vertex $u + kC + t$ is a leader because then its followers $P(u + kC + t)$, *i.e.*, $u + jC + t, 0 \leqslant j \leqslant k - 1$ are all forbidden. We prove by induction on t.

Base case: $t = C - 1$. Since $u + C - 1$ is forbidden, its leader is the vertex $v = u + C - 1 + qC$ for some $0 \leqslant q \leqslant k$ and the followers of v, the vertices in $P(v)$, are forbidden. The vertex

$u-1$ is at a distance multiple of C from v but is not forbidden, *i.e.,* $u-1 \notin P(v)$. Hence $u-1 < v-kC$, implying $q > k-1$. Thus $q = k$, which implies our claim.

Induction case: suppose the claim is true for $t = t'$ where $|B| < t' \leqslant C-1$, *i.e.,* the vertex $u+kC+t'$ is a leader. We need to prove that $u+kC+t'-1$ is also a leader.

Since $u+t'-1$ is forbidden, its leader is the vertex $v = u+t'-1+qC$ for some $0 \leqslant q \leqslant k$. Thus, if $q = k$ then we are done. So assume that $q < k$, *i.e.,* $q+1 \leqslant k$. We will show that this leads to a contradiction.

Since v is a leader, by Definition 3.22, there exist block of forbidden vertices $F_1 = [x, v]$ a clique $Q = [v-kC, x]$ for some vertex x. Again, by induction hypothesis, each of the vertices $[u+kC+t', u+(k+1)C-1]$ is a leader. Thus, the set of vertices $F_2 = [u+qC+t', u+(q+1)C-1]$ is forbidden. But F_2 can be rewritten as $[v+1, u+(q+1)C-1]$. Thus $F = F_1 \cup F_2 = [x, u+(q+1)C-1]$ is a forbidden block ending at $u+(q+1)C-1$. Because of Q, F the vertex $u+(q+1)C-1$ is a leader by Definition 3.22. Among its followers, *i.e.,* $P(u+(q+1)C-1)$, the $(q+1)$th from the right is $u-1$. This is a contradiction because $u-1$ is not forbidden. □

Lemma 3.26. *Suppose* COMBPART *creates consecutive blocks $B_1, \ldots, B_k$, where B_k is not the rightmost block. Then the $kC - \sum_{i=1}^{k} |B_i|$ consecutive vertices following B_k are all forbidden.*

Proof. Let u_i be the leftmost vertex of B_i. Apply Lemma 3.25 to B_i and consider the block of forbidden vertices F_i corresponding to $j = k-i$ only. Then $F_i = [x_i, y_i]$ where $x_i = u_i + (k-i)C + |B_i|$ and $y_i = u_i + (k-i)C + C - 1$. For $i < k$, since $u_{i+1} = u_i + |B_i|$, we have $y_{i+1} = u_{i+1} + (k-i)C - 1 = u_i + |B_i| + (k-i)C - 1 = x_i - 1$. Hence the blocks F_i in the order $i = k, \ldots, 1$ are consecutive and their union $F = \cup_{i=k}^{1} F_i = [x_k, y_1]$. Since $y_i = x_i + (C - |B_i|) - 1 = y_{i+1} + (C - |B_i|)$, we have $y_1 = y_k + \sum_{i=1}^{k-1}(C - |B_1|) - 1 = x_k + kC - \sum_{i=1}^{k} |B_i| - 1$. But $x_k = u_k + |B_k|$ is the first vertex following B_k. Hence F is a set of $kC - \sum_{i=1}^{k} |B_i|$ forbidden vertices following B_k. □

Lemma 3.27. *If an interval representation for a PIG G is given then* COMBPART *correctly solves the partition subproblem on G in $O(n)$ time.*

Proof. If COMBPART outputs NO, then there is a set of C consecutive forbidden vertices. To cover these vertices we need a block of size at least $C+1$. So there cannot be any valid partition. Hence COMBPART is correct.

Now we prove that if CombPart outputs Yes then the partition generated is a valid partition. Since the algorithm generates blocks of size at most C, the size constraint is satisfied. We only need to prove that no clique intersects more than $k+1$ blocks generated by CombPart. We prove this by contradiction.

Suppose there is a clique Q that intersects $k+2$ blocks $B_0, B_1, \ldots, B_{k+1}$. Without loss of generality, we assume that only the leftmost vertex of Q is covered by B_0 and only the rightmost vertex of Q is covered by B_{k+1}. Because, otherwise we can take a sub-clique $Q' \subset Q$ with this property.

Let F denote $kC - \sum_{i=1}^{k} |B_i|$ vertices following B_k and v be the rightmost vertex of F. By Lemma 3.26 the vertices F are forbidden. Also the leftmost vertex of F is the leftmost vertex of B_{k+1}, *i.e.,* the rightmost vertex of Q and $|Q|+|F|-1 = (1+\sum_{i}^{k} |B_i|+1)+(kC-\sum_{i=1}^{k} |B_i|)-1 = kC+1$. By definition 3.22 the vertex $v-kC$ is forbidden. But $v-kC$ is the leftmost vertex of Q, *i.e.,* the rightmost vertex of B_0 which can not be forbidden. It is a contradiction.

If an interval representation is given then by Proposition 3.16 we can find the maximal cliques and hence can compute the array `Lmn` in $O(n)$ time. By Lemma 3.24 the marking of forbidden vertices takes time $O(n)$. The greedy procedure for generating the parts also takes $O(n)$ time. Overall time taken is $O(n)$. □

Combining Lemma 3.15, Lemma 3.27 and the discussions at the end of the subsection 3.5.2 we get a proof of Theorem 3.1.

For the example in Figure 1.2 with the interval representation in Figure 1.1, the vertices are numbered as follows: $1 : [0,4], 2 : [1,5], 3 : [2,6], 4 : [3,10], 5 : [7,11], 6 : [8,12], 7 : [9,13]$. Here, $C = 2, \omega = 4, k = 1$ and the forbidden vertices are $1, 3, 4, 6$. Hence there is no $[k+1, C]$-partition. Using SimplePart we get the following coloring: color 1 for vertices $1, 2, 7$, color 2 for vertices $3, 4$ and color 3 for vertices $5, 6$.

3.6 Splittable Weighted Problem

In Section 1.2 we introduced the notion of weight-splitted graph to formally define the splittable weighted problem. The *weight-expanded graph* of a graph $G = (V, E)$ with weight W, in short $WXP(G, W)$, is the graph $G' = (V', E')$ such that G' with weight 1 to all vertices is a weight-split graph of G with W. It can be seen that a graph G with weight W on vertices is (λ, C)-split colorable if and only if $G' = WXP(G, W)$ is (λ, C)-colorable. If G is a PIG then an interval representation for G' can be obtained from the interval representation of G by creating $W(v)$ copies of the corresponding interval for each v in G. No two of these new set of intervals are proper subset of each other. Hence G' is also a PIG.

Thus to solve the splittable weighted problem on a PIG $G = (V, E)$ with weight W it is enough to solve the unweighted problem on $WXP(G, W)$. In what follows we will denote $G' = (V', E') = WXP(G, W)$, $n' = |V'|$ and $m' = |E'|$. Applying the algorithm described in Section 3.5 on G' gives correct result but it makes the algorithm pseudo-polynomial as it takes $O(n')$ time, proportional to the sum of weights. This is mainly because the algorithm iterates over each vertex in G'.

However, it turns out that iterating over each vertex in G' is not necessary. The forbidden vertices in G' can be divided into blocks such that if the vertices u, v both are in some block b then the leaders of u, v are both in some block l. We call such forbidden blocks *FB*s. Analogous to the vertices, we say that FB l is the *leader* of FB b and b is the *follower* of l. It can be seen that all vertices in an FB can be marked together.

3.6.1 Marking Forbidden Blocks

We now modify the algorithm presented in Section 3.5 to let it work with FBs instead of forbidden vertices. The modified algorithm to mark all the FBs, which we call SPLITMARK, is shown in Algorithm 3.

We use the following correspondence between a vertex $v \in V$ and a vertex $v' \in V'$. The vertex $v' = h(v, q)$ if v' is the qth copy of v where $1 \leqslant q \leqslant W(v)$ and $v = \bar{h}(v')$ if v' is a copy of v. The set $\{h(v, 1), \dots, h(v, W(v))\}$ of copies of v is represented by $H(v)$. As usual, we will interchangeably use $1 \leqslant v' \leqslant n'$ ($1 \leqslant v \leqslant n$) to denote a vertex $v' \in V'$ ($v \in V$) as well as its position in the canonical ordering of vertices in G' (G).

The algorithm uses an auxiliary array $Z(0, \dots, n)$ such that $Z(0) = 0$ and for all $v > 0$, the entry $Z(v) \in G'$ denotes the rightmost copy of v, *i.e.,* $h(v, W(v))$. Since all the copies $H(v) \subseteq V'$ of $v \in V$ appear consecutively in the canonical ordering of G', we have $Z(v) = \sum_{i=1}^{v} W(i)$. If Z is given then the values of the function $\bar{h}(u)$ for all u belonging to a subset of vertices $S \subseteq V'$ can be computed in right to left order, in overall $O(|S| + n)$ time.

The algorithm stores the FBs sorted in canonical ordering in a doubly linked list F. The information stored in auxiliary arrays `Ldist` and `Rnf` earlier, is also kept in the list F itself. Thus each entry b of F has the following fields: (i) `left` denotes the leftmost vertex of the FB b, (ii)

Algorithm 3: SPLITMARK

```
Input  : Maximal cliques of a PIG $G$ with $W$, Lmn$(1,\dots,n)$, $Z(0,\dots,n)$
Output: Doubly linked list of FBs $F$ in $G' = WXP(G,W)$
1  foreach maximal clique $[u,v]$ in $G$ do                         /* phase 1 */
2      $u' = Z(u-1)+1$;   $v' = Z(v)$; /* $[u',v']$ is a maximal clique in $G'$ */
3      if $v' - u' + 1 \geqslant kC + 2$ then
4          $F$.Inlay$(u' + kC, v' - 1, 0)$;
5  $i = F$.end→prev; /* $F$.end→prev is the rightmost non-sentinel FB */
6  while $i \neq F$.begin do                                        /* phase 2 */
7      $v = i$→right;
8      $j = i$→rnf $= i$→next→rnf;   if $v < i$→rnf→right then $j = i$→rnf $= i$;
9      while $j$→prev→right == $j$→left $- 1$ do $i$→rnf $= j$; $j = j$→prev;
10     if $i$→ldist $\leqslant (k-1)C$ then              /* $i$ is a follower block */
11         $F$.Inlay$(i$→left $- C, v - C, i$→ldist $+ C)$;
12     $t =$ Lmn$(\bar{h}(i$→rnf→left$))$;     /* function $\bar{h}$ is computed using $Z$ */
13     if $t \leqslant v - kC$ then               /* new leader block ending at $v$ */
14         $F$.Inlay$(t + kC - C, v - C, C)$;
15     $i = i$→prev;
```

right denotes the rightmost vertex of b, and (iii) ldist denotes the distance of b.right from its leader, (iv) rnf points to the leftmost FB such that all FBs between b.rnf and b are consecutive, *i.e.,* all vertices in $[b.\texttt{rnf}\rightarrow\texttt{left}, b.\texttt{right}]$ are forbidden, (v) prev points to the FB on the left of b, and (vi) next points to the FB on the right of b. Here $p \rightarrow q$ represents the field q of the FB pointed by the pointer p. The algorithm also keeps two sentinel FBs in F, (i) the leftmost FB F.begin and (ii) the rightmost FB F.end, containing two imaginary primarily forbidden vertices numbered $-1, n' + 1$ respectively. Each sentinel has ldist $= 0$. By default rnf of each FB points to itself.

In addition to the standard operations of insert, delete and both way traversals, F supports a new operation F.Inlay(s,t,d) which inserts into F a new FB b with b.left $= s$, b.right $= t$, b.ldist $= d$, and makes sure that the FBs in F remain non-intersecting and sorted. Let $b_1, \dots, b_s$ be the FBs in F which intersect b. The operation Inlay does the following: (i) deletes all the FBs in F which are subsets of b, (ii) if b partly intersects b_1 then truncates b_1 to $[b_1.\texttt{left}, b.\texttt{left} - 1]$, (iii) if b partly intersects b_s then truncates b_s to $[b.\texttt{right} + 1, b_s.\texttt{right}]$, and (iv) inserts b at its proper position in F.

For each $v \in V$ the algorithm stores in $\texttt{Lmn}(v)$ the leftmost neighbor of $Z(v)$ in G'. If the maximal cliques of G are given, then the elements of $\texttt{Lmn}$ can be computed in $O(n)$ time.

For a maximal clique $Q = [u', v']$ in G' of size at least $kC + 2$, each subclique of size $kC+2$ creates a primarily forbidden vertex. These primarily forbidden vertices are consecutive; the leftmost is $u' + kC$ due to the subclique starting at u' and the rightmost is $v' - 1$ due to the subclique ending at v'. The algorithm uses this fact to create an FB $[u', v' - kC]$ in such case. Note that each maximal clique $[u, v]$ in G has one-to-one mapping with the maximal clique $[Z(u - 1) + 1, Z(v)]$ in G'.

We have seen that v is a leader if for the largest forbidden sequence $F = [u, v]$ there exists a clique $Q = [t, u]$ such that $t \leqslant v - kC$. In fact if Q is sufficiently large then for any j the vertex $v - j$ is a leader as long as $t \leqslant v - j - kC$ or $v - j \geqslant t + kC$. Thus if $Q = [t, u]$ is the maximal clique ending at u and $t \leqslant v - kC$ then the sequence of vertices $[t + kC, v]$ is a leader FB. The algorithm uses this fact to mark the secondarily forbidden vertices.

For marking blocks of secondarily forbidden vertices the algorithm maintains the following invariants at the beginning of each iteration in phase 2 where i points to the FB $[u, v]$ and $j = i{\to}\texttt{next}$ points to the FB $[x, y]$:

1. F contains all the leader FBs in the range $[v + 1, n']$.

2. F contains all the follower FBs in $[v + 1 - C, n']$.

3. For each follower FB b in $[v + 1 - C, n']$ its distance from the leader FB is $b.\texttt{ldist}$.

4. $j{\to}\texttt{rnf}$ points to the leftmost FB in the range $[1, y]$ such that all vertices in $[j{\to}\texttt{rnf}{\to}\texttt{left}, y]$ are either primarily forbidden or secondarily forbidden due to the leaders in $[y+1, n'+1]$.

In each iteration the invariants are extended for $i{\to}\texttt{prev}$ which we assume to point to the FB $[s, t]$. We show that the algorithm maintains the invariants by using induction on i.

At the beginning of phase 2, i points to the rightmost FB created due to the primarily forbidden vertices of the rightmost maximal clique. Hence there cannot be any forbidden vertex in the range $[v + 1, n']$. Thus the invariants 1-3 are maintained. The right sentinel $F.\texttt{end}$ is set with imaginary forbidden vertex $n' + 1$ such that invariant 4 is also satisfied.

Suppose the invariants are maintained at the beginning of an iteration. Since $i \to \texttt{rnf}$ points to a block which either equals to or is on the left of the blocks pointed by i and $j \to \texttt{rnf}$, the algorithm sets $i \to \texttt{rnf}$ to the left one of them in step 8 and in step 9 moves $i \to \texttt{rnf}$ further to the left until there is a gap in the FBs implying existence of a non-forbidden vertex. This step is valid because all FBs in the range $[v - C + 1, v]$ are already marked.

If $[u, v]$ is a follower of a previously discovered FB l, at step 11 the algorithm inserts FB $b = [u - C, v - C]$ into F if b, too, is a follower of l. Otherwise there cannot be any new follower in the range $[u - C, v - C]$. In either case, at the end of step 11, the list F contains all FBs in the range $[u - C, v - C]$ except the rightmost follower of a possible new leader FB ending at v, which is inserted into F at step 14. Thus at the end of the iteration, F contains all FBs in the range $[u - C, n']$. If $u = t + 1$ we are done. Otherwise there is no forbidden vertex in the range $[t + 1, u - 1]$ and hence there cannot be any new follower FB in the range $[t - C + 1, u - C - 1]$. Thus we are done in that case too.

For each FB SPLITMARK takes $O(1)$ time except for the operation `Inlay`. Since `Inlay` is invoked on FBs in a right to left order, it can be implemented by maintaining an extra pointer that traverses from right to left through the FBs, in overall $O(|F|)$ time. Suppose the maximum weight of $v \in V$ is M. Then the set of vertices $H(v) \subseteq V'$ contains at most $\lceil M/C \rceil$ followers of each leader FB. There can be as many leader FBs as the number of maximal cliques in G', *i.e.,* at most n. Hence $|F| = O(n^2 M/C)$. Thus, SPLITMARK takes $O(n^2 M/C)$ time.

Now we show that there are weighted PIGs on which SPLITMARK creates $\Omega(n^2 M/C)$ FBs. For some parameters t, α consider a PIG G with $n = 3t + 1$ vertices given by the intervals $I_1, \ldots, I_{3t+1}$ where for $1 \leqslant j \leqslant t + 1$ interval $I_j = [j, 2t + 2j]$ has weight 2, for $1 \leqslant j \leqslant t$ interval $I_{t+1+j} = [t + 1 + j, 4t + 2 + j]$ has weight $2\alpha t$, and again for $1 \leqslant j \leqslant t$ interval $I_{2t+1+j} = [2t + 1 + 2j, 5t + 2 + j]$ has weight 2. Thus in G each of the $t + 1$ maximal cliques has $t + 1$ vertices of weight 2 and t vertices of weight $2\alpha t$. Clearly G' has $(t + 1) \times 2 + t \times 2\alpha t + t \times 2 = 2(\alpha t^2 + 2t + 1)$ vertices and each of the $t + 1$ maximal cliques $[2j - 1, 2j + 2\alpha t^2 + 2t]$, $1 \leqslant j \leqslant t + 1$, has size $2\alpha t^2 + 2t + 2$. For $C = 2t$, we have $k(G') = \alpha t + 1$. Phase 1 of our algorithm creates $t + 1$ FBs each having a single vertex $2j + 2\alpha t^2 + 2t - 1$ where $1 \leqslant j \leqslant t + 1$. Phase 2 creates a FB from each of the remaining odd numbered vertices in G'. Thus $|F|$ = the number of odd vertices in G', *i.e.,* $\alpha t^2 + 2t + 2 = \Omega(\alpha n^2) = \Omega(n^2 M/C)$.

Thus we have proved the following result.

Lemma 3.28. *If the arrays* Z, `Lmn` *and the maximal cliques of a PIG* $G = (V, E)$ *with weights*

W on vertices are given and M is maximum weight of a vertex $v \in V$ then SPLITMARK *correctly marks the FBs of $G' = WXP(G, W)$ in $\Theta(n^2M/C)$ time.*

3.6.2 Algorithm SPLITPART

We now give the modifications to COMBPART to use the FBs. We call this modified algorithm SPLITPART, which is shown in Algorithm 4.

Algorithm 4: SPLITPART

```
Input  : A PIG $G = (V, E, W)$
Output: If $G'$=$WXP(G)$ has a $(k+1, C)$-partition; if YES also output the partition
1 $F$ = list of FBs returned by SPLITMARK on $G$;   $u = 1$;   $i = F$.begin→next;
2 while $u \leqslant n'$ do
3     $v = \min\{u + C - 1, n'\}$;
4     while $i$→right $< v$ do $i = i$→next;
5     $v = i$→left $- 1$;
6     if $v < u$ then return NO else create part $[u, v]$; $u = v + 1$;
7 return YES;
```

Lemma 3.29. *If an interval representation for a PIG G with weight W is given M is the maximum weight of a vertex in G then* SPLITPART *correctly solves the partition subproblem on $WXP(G, W)$ in $O(n^2M/C)$ time.*

Proof. Given that SPLITMARK correctly marks the forbidden blocks of $G' = WXP(G, W)$, it is easy to see that SPLITPART generates the same partition that COMBPART would have generated on G'. Given an interval representation of G, arrays Z, `Lmn` and maximal cliques of G can be computed in $O(n)$ time. Thus by Lemma 3.28, computing F takes $O(n^2M/C)$ time. The block generation step also takes $O(|F|) = O(n^2M/C)$ time. □

SIMPLEPART in Lemma 3.19 can be slightly modified to use vertices in G' but still taking $O(n)$ time. Thus combining Lemmas 3.15, 3.29 and the discussions at the end of Subsection 3.5.2, we get a proof of Theorem 3.2.

3.7 Nonsplittable Weighted Problem

For the nonsplittable weighted problem, by definition, the weight of a vertex cannot exceed C. Using the algorithm for the splittable weighted problem we can get a $O(n^2)$ time 2-approximation algorithm for the NP-hard nonsplittable weighted problem.

Lemma 3.30. *There exists a $O(n^2)$ time algorithm for the nonsplittable weighted partition problem that generates a (λ, C)-partition on a PIG G with weight W such that λ is at most 2 times the clique intersection of the partition generated by an optimal algorithm on G.*

Proof. We first solve the corresponding splittable weighted problem on G using the algorithm described in Section 3.6. As the maximum weight of a vertex is at most C, this takes $O(n^2)$ time. Let the blocks in the (λ', C)-partition created by the algorithm be $\mathcal{P}' = \{P'_1, P'_2, \ldots, P'_t\}$. Then λ' is a lower bound on the clique intersection λ^* of the partition generated by any optimal algorithm on G.

We convert the partition $\mathcal{P}'$ in which a vertex might be splitted into multiple blocks, to a partition $\mathcal{P}$ in which each vertex is completely within a block, in $O(n)$ time as follows. Consider each vertex v left to right and assign it to a block in $\mathcal{P}$. If the weight of v is at most the remaining unassigned capacity of the current block b in $\mathcal{P}$ then assign v to b. Otherwise, create a new block b' in $\mathcal{P}$ and assign v to b'.

Since the weight of a vertex in a nonsplittable problem is at most C, and the vertices $h(v)$ in G' corresponding to a vertex v in G appear in consecutive blocks in $\mathcal{P}'$, the above process creates at most one extra block in $\mathcal{P}$ for each block in $\mathcal{P}'$, hence $\mathcal{P}$ is a (λ, C)-partition with clique intersection $\lambda \leqslant 2\lambda' \leqslant 2\lambda^*$. □

Combining Lemma 3.15 and Lemma 3.30 we get a proof of Theorem 3.3.

3.8 Summary

We gave polynomial time exact algorithms for unweighted and splittable weighted versions and a 2-approximation algorithm for the nonsplittable weighted version of component coloring on PIGs. The ideas of the algorithms for PIG do not apply directly for general interval graphs because there are general interval graphs (see Figure 3.2) where Lemma 3.13 on the equivalence of coloring and block-partition does not hold. We use different ideas in Chapter 4 to solve the problem on general interval graphs.

Chapter 4

Stationary Problems on Interval and Circular-arc Graphs

In this chapter we discuss our algorithm for the stationary problem on general interval and circular-arc graphs. For simplicity we describe the algorithm on the light-trail scheduling problem. The analogous algorithm for stationary component coloring problem can be similarly described.

We start with the description of the stationary light-trail scheduling problem in Section 4.1. In Section 4.2 we give an overview of our algorithm which classifies the transmissions based on their lengths, schedules the transmissions of each class separately and finally merges the schedule. Scheduling small transmissions, *i.e.,* of classes $0, 1$ is easy. We describe how to schedule classes $\geqslant 2$ in Section 4.3 and how to merge schedules of all classes for better performance in Section 4.4. We use the congestion, *i.e.,* maximum total bandwidth requirement at a link over all links as a lower bound to prove the performance of our algorithm. Finally, in Section 4.5 we give an example instance of the stationary problem where the congestion lower bound is weak somewhat justifying our result.

4.1 Stationary Light-trail Scheduling

We consider an optical path network of of p processors, numbered 0 to $p-1$. The link between the two consecutive processors i and $i+1$ is numbered i. Communication is considered undirected. This simplifies the discussion; it should be immediately obvious that all results directly carry over to directed communications and also to ring networks.

In WDM, the physical optic fiber carrying signals of w different wavelengths is logically thought of as w independent parallel fibers each carrying signals of a single wavelength. Each processor can be thought of as having a separate shutter on each of the w fibers. Each shutter can be set ON, meaning it allows the optical signal to pass, or OFF, meaning it does not. The segment between two OFF shutters is a light-trail. A transmission request from processor i to processor j can be assigned to a light-trail if the following conditions are met:

1. $u \leqslant i < j \leqslant v$ where u, v are the OFF processors of the light-trail.

2. The sum of the bandwidth requirements of all requests assigned to any single light-trail does not exceed the *capacity* of a wavelength, *i.e.,* the maximum bandwidth that can be served using a wavelength.

The requests assigned to a light-trail are served by time division multiplexing, with service duration proportional to the bandwidth requirement. Thus at any time instant the light-trail is used by at most one request.

The input for the stationary problem is a matrix B with $B(i, j)$ denoting the bandwidth requirement for the transmission request from processor i to processor j, as a fraction of the bandwidth capacity of a single wavelength which we define to be 1 without loss of generality. The goal is to schedule these in a minimum number of wavelengths w. The output must give w as well as the light-trails used on each wavelength and the mapping of each transmission to a light-trail that serves it.

It will be convenient to represent/visualize schedules geometrically. We will use the x axis to represent our processor array, with processors at integer points and the y axis to represent the wavelengths numbered $0, 1, 2,$ and so on. The region bordered by $y = \bar{y}$ and $y = \bar{y} + 1$ will be used to depict the transmissions assigned to the wavelength numbered k. The region will be partitioned with vertical lines at the processors where the shutters are OFF. Each of the rectangular parts in the partition represents a light-trail created on the corresponding wavelength. A transmission from processor i to processor j having bandwidth requirement b will be denoted as $[i, j]$ and represented as a rectangle of height b located horizontally in the region between $x = i$ and $x = j$ and vertically within the region corresponding to the light-trail in which it is scheduled. We will also use the terms *length*, *extent*, and *height* of transmission $[i, j]$ to mean $j - i$, the interval $[i, j]$, and b respectively. Unless there is ambiguity in the context we will also use $[i, j]$ to denote a light-trail with end processors i and j. Similarly we will also use the terms *length*, and *extent* of light-trail $[i, j]$ to mean $j - i$ and the interval $[i, j]$ respectively.

As an example consider a network with 3 processors, 0,1,2 and a transmission matrix B in which $B(0, 1) = B(1, 2) = 0.6$ and $B(0, 2) = 0.4$ are the only non-zero entries. In order to enable the transmission [0,2], we must have a wavelength with a light-trail which goes all the way from 0 to 2. In this light-trail, we cannot put both the remaining communications, because then the bandwidth would become $0.6 + 0.6 + 0.4 = 1.6$, *i.e.,* larger than 1. Say we put the transmission [0,1] in this light-trail. Then the remaining communication, [1,2] would require its own light-trail, and for that we will need another wavelength. This is shown in Figure 4.1(a). Another way is as follows. We put transmission [0,2] in a single light-trail extending from 0 to 2. Then on another wavelength, we create two light-trails, with the shutter at 1 in the OFF position. The transmissions [0,1] and [1,2] can now be placed in these respective light-trails. This solution is given in Figure 4.1(b). Both the solutions require 2 wavelengths, and they are optimal because the required communication cannot be implemented using just 1 wavelength.

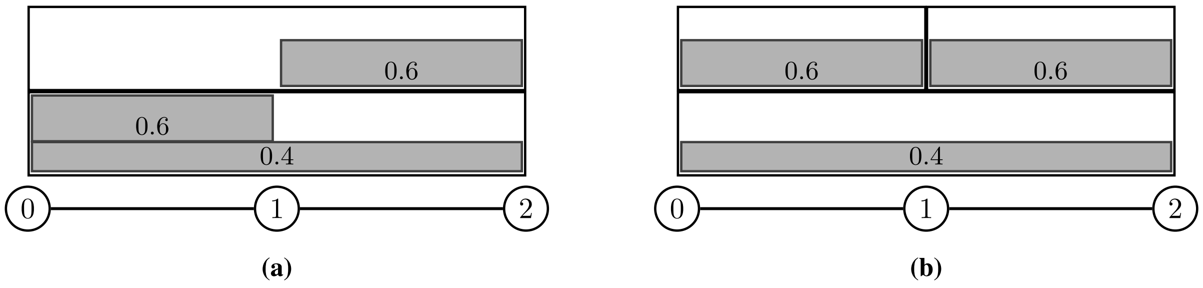


Figure 4.1: Solutions to the Stationary Problem Example

It is customary to consider two problem variations: *nonsplittable*, in which a transmission must be assigned to a single light-trail, and *splittable*, in which a transmission can be split into two or more transmissions by dividing up the bandwidth requirement, and each of them can be assigned to a different light-trail. Note that when a transmission is split into multiple transmissions, the length and extent remain the same, only the height is divided. Our results hold for both variations.

Note that the bin-packing problem which is NP-hard, is a special case of the stationary problem where each item corresponds to a transmission from processor 0 to processor $p - 1$ and each bin corresponds to a light-trail (and to a wavelength too because each light-trail completely occupies a wavelength). Thus the nonsplittable stationary problem is NP-hard. We do not know whether the splittable problem is also NP-hard.

We will use $\omega_l(S)$ to denote the congestion induced on a link l by a set S of transmissions. This is simply the total bandwidth requirement of those transmissions from S requiring to cross link l. Clearly $\omega(S) = \max_l \omega_l(S)$, the maximum congestion over all links, is a lower bound on the number of wavelengths needed. Finally if t is a transmission, then we abuse notation to write $\omega_l(t), \omega(t)$, instead of $\omega_l(\{t\}), \omega(\{t\})$, for the congestion contributed by t only, which is equal to the bandwidth requirement of t. Let R be the set of all transmissions of an instance of the stationary problem. Let n be the size of R. We will use ω to denote the overall congestion $\omega(R)$.

4.2 Algorithm Overview

Getting an algorithm which requires only $O(\omega \log p)$ wavelengths is easy. If $\omega_{p/2}$ denotes the congestion of the link between processor $p/2$ and processor $p/2+1$, then the transmissions crossing this link can be scheduled in $\lceil\omega\rceil \geqslant \lceil\omega_{p/2}\rceil$ wavelengths for the splittable case, and twice that many for the nonsplittable case (using ideas from bin-packing [19]). The remaining transmissions do not cross the middle link, and hence can be scheduled by separately solving two subproblems, one for the transmissions on each half of the array. The two subproblems can share the wavelengths. If $\lambda(p,\omega)$ denotes the number of wavelengths used for scheduling transmissions of congestion at most ω in a linear array of p processors, we have the recurrence $\lambda(p,\omega) = O(\lceil\omega\rceil) + \lambda(p/2,\omega)$. This solves to $O(\omega \log p)$. But we can do better.

Note that it is relatively easy to get a good schedule if all the transmissions have the same length (see Section 4.3). So we divide the transmissions into classes based on their lengths, then schedule each class separately and finally merge the schedules. The merging step is also somewhat sophisticated. This is the outline of our algorithm.

1. *Partition into classes.* Say a transmission belongs to class i if its length is between 2^{i-1} (exclusive) and 2^i (inclusive). Let R_i denote the set of transmissions of class i, for $i = 0$ to $\lceil\log_2(p-1)\rceil$. Let n_i denote the size of R_i.

2. *Schedule transmissions of each class separately.* It will be seen that each class can be scheduled efficiently, *i.e.,* using $O(1+\omega(R_i))$ wavelengths.

3. *Merge the schedules of different classes.* We do not simply collect together the schedules constructed for the different classes, but do need to mix them together, and repartition.

Scheduling classes R_0, R_1 is easy. Note that each transmission in R_0 has length 1. So they can be assigned to light-trails created by simply putting shutters OFF at every processor on all the wavelengths that are to be used. Now for a fixed l consider the light-trails $[l, l+1]$ on all the wavelengths. Each of these light-trails can be thought of as a bin in which the transmissions $[l, l+1]$ are to be assigned. Clearly, $\lceil \omega_l(R_0) \rceil$ light-trails will suffice for the splittable case, and twice that many for the nonsplittable case (using ideas from bin-packing [19]). Since the light-trails for different l do not overlap, they can be on the same wavelength. So $\max_l O(\lceil \omega_l(R_0) \rceil) = O(\lceil \omega(R_0) \rceil)$ wavelengths will suffice. Transmissions in R_1 have length 2. So they can be assigned to light-trails created on two sets of wavelengths – one having shutters OFF at even processors and the other having shutters OFF at odd processors. Transmissions starting at an even (odd) processor are assigned to a light-trail on a wavelength of the first (second) set. Using an argument similar for the transmissions in R_0, we can show that each of these sets requires $O(\lceil \omega(R_1) \rceil)$ wavelengths. So for the rest of this thesis we only consider classes 2 and larger.

4.3 Schedule Class $i \geqslant 2$

It seems reasonable that if the class R_i is further split into subclasses each of which has $O(1)$ congestion, then each subclass could be scheduled using $O(1)$ wavelengths. This intuition is incorrect for an arbitrary collection of transmissions with congestion $O(1)$, as will be seen in Section 4.5. However, the intuition is correct when the transmissions have nearly the same length, as they do when taken from any single R_i.

Lemma 4.1. *There exists an $O(nm_i\omega)$ time procedure to partition R_i into sets $S_1, S_2, \ldots, S_k$ where $k \leqslant \lceil \omega(R_i) \rceil$ such that* (i) $\omega(S_j) < 4$ *for all j, and* (ii) *if a transmission in S_j uses link l then $\lceil \omega_l(R_i) \rceil \geqslant j$.*

Proof. We start with $T_1 = R_i$, and in general given T_j we pick a subset of transmission S_j from T_j using a procedure described below and repeat with the remaining transmissions $T_{j+1} = T_j \backslash S_j$ until T_{j+1} becomes empty for some value k of j.

For each link l from left to right, we pick transmissions one by one from the set of transmissions crossing link l into S_j until we have removed at least unit congestion from $\omega_l(T_j)$ or reduced $\omega_l(T_j)$ to 0. Note that if the transmissions already picked while considering the links on the left of l also have congestion at least 1 at link l then we do not add any more transmission while considering link l. So at the end the following condition holds:

$$\forall l,\ \omega_l(S_j) \begin{cases} = \omega_l(T_j) & \text{if } \omega_l(T_j) \leqslant 1, \text{ and} \\ \geqslant 1 & \text{otherwise.} \end{cases} \tag{4.1}$$

However, to make sure that $\omega(S_j)$ is not large, we move back transmissions from S_j, in the reverse order as they were added, into T_j so long as condition (4.1) remains satisfied. It can be seen that the construction of a single S_j takes at most $O(p|T_j|) = O(nm_i)$ time in both the pick-up step and the move-back step. For all S_j it takes $O(nm_i\omega)$ time.

Now we show that condition (i) of the lemma is satisfied, *i.e.*, $\omega(S_j) < 4$ for all j. At the end of the move-back step, for any transmission $t \in S_j$ there must exist a link l such that $\omega_l(S_j) < 1 + \omega(t)$, otherwise t would have been removed. We call l as a *sweet spot* for t. Since $\omega(t) \leqslant 1$ we have $\omega_l(S_j) < 2$ for any sweet spot l.

Now consider any link x. Of the transmissions through x, let L_1 (L_2) denote transmissions having their sweet spot on the left (right) of x. Consider y, the rightmost of these sweet spots of some transmission $t \in L_1$. Note first that $\omega_y(S_j) < 2$. Also all transmissions in L_1 pass through both x, y. Thus $\omega_x(L_1) = \omega_y(L_1) \leqslant \omega_y(S_j) < 2$. Similarly, $\omega_x(L_2) < 2$. Thus $\omega_x(S_j) = \omega_x(L_1) + \omega_x(L_2) < 4$. But since this applies to all links x, $\omega(S_j) < 4$.

To show that condition (ii) is also satisfied, suppose S_j contains a transmission that uses some link l. The construction process above must have removed at least unit congestion from l in every previous step 1 through $j-1$. Thus $\omega_l(R_i) > j-1$. That implies $\lceil\omega_l(R_i)\rceil \geqslant j$. This also implies that $k \leqslant \max_l\lceil\omega_l(R_i)\rceil = \lceil\omega(R_i)\rceil$. □

A transmission t is said to cross a processor u if t starts at a processor on the left of u and ends at a processor on the right of u. Since every transmission t in S_j has length at least $2^{i-1}+1$, t must cross some processor whose number is a multiple of 2^{i-1}. The smallest such numbered processor is called the *anchor* of t. The *trail-point* of a transmission t is the rightmost processor numbered with a multiple of 2^{i-1} that is on the left of the anchor of t. If the transmission has trail-point at processor $q2^{i-1}$ for some q, then we define $q \bmod 4$ as its *phase*. These definitions are illustrated in Figure 4.2 with an example transmission from node 7 to node 14. Length = 7. Class = 3. Anchor = 8. Trail-point = 4. Phase = 1.

Lemma 4.2. *The set S_j can be scheduled using $O(1)$ wavelengths in $O(p|S_j|)$ time.*

Proof. We partition S_j further into sets S_j^δ containing transmissions of phase δ. This takes time $O(|S_j|)$. Note that the transmissions in any S_j^δ either overlap at their anchors, or do not overlap

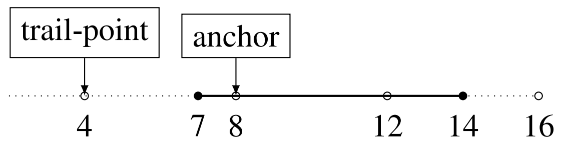


Figure 4.2: Anchor and trail-point of a transmission

at all. This is because if two transmissions in S_j^δ have different anchors, then these two anchors are at least 2^{i+1} distance apart. Since the length of each transmission is at most 2^i, the two transmissions cannot intersect.

Now we show that for each δ the set S_j^δ can be scheduled using $O(1)$ wavelengths. For the splittable case consider 4 wavelengths, each having shutters OFF at processors numbered $(4q + \delta)2^{i-1}$. Let $x = (4q + \delta)2^{i-1}$ and $y = (4(q + 1) + \delta)2^{i-1} = x + 2^{i+1}$ be two nearest processors having shutters OFF. Among the $O(p/2^i)$ light-trails thus created, for a fixed q, each of the 4 light-trails $[x, y]$ can be thought of as a bin in which the transmissions having extent totally within $[x, y]$ and total bandwidth requirement at most 1 are to be assigned. This is an instance of the bin-packing problem. Clearly, for a fixed q, these 4 light-trails will suffice for the splittable case, because $\omega(S_j^\delta) < 4$. This takes time proportional to the number of requests considered. Since the light-trails for different q do not overlap, the instances of the bin-packing problem can share wavelengths and hence these 4 wavelengths will suffice. For the nonsplittable case, 8 wavelengths will suffice, using standard bin-packing ideas, e.g., First-Fit [19]. Overall it takes at most $O(|S_j|p/2^i) = O(p|S_j|)$ time.

Thus all of S_j can be accommodated in at most 16 wavelengths for the splittable case, and at most 32 wavelengths for the nonsplittable case. $\square$

Lemma 4.3. *The entire set R_i can be scheduled in time $O(nm_i\omega)$ such that at each link l there are $O(\omega_l(R_i) + 1)$ light-trails.*

Proof. We first consider the light-trails as constructed in Lemma 4.2. For all S_j the construction takes time $O(nm_i\omega)$. In this construction, uniformly at all links there are at most $\lceil\omega(R_i)\rceil \leqslant \omega(R_i) + 1$ sets of light-trails such that each set corresponds to $O(1)$ light-trails created to schedule the transmissions of an S_j. Note that $\omega(R_i) = \max_l \omega_l(R_i)$. So, in this construction the condition of the lemma is surely satisfied for the link where the congestion is maximum. For other links the condition of the lemma may not be satisfied because (1) there may be empty light-trails and (2) some light-trails may contain links that are not used by any of the transmissions associated with the light-trail. So we remove empty light-trails and in case

(2) we shrink the light-trails by removing the unused links (which can only be near either end of the light-trail because all transmissions assigned to a light-trail overlap at their anchor). This modification takes time proportional to the number of light-trails which is $O(n)$. We prove next that with this modification, the condition of the lemma is satisfied.

Let j be the largest such that a transmission from S_j uses link l. After the modification the light-trails that carries transmissions from $S_{j'}$ for $j' > j$ do not use link l. So now there are j sets of light-trails using link l such that each set has $O(1)$ light-trails. However we know from Lemma 4.1 that $j \leqslant \lceil \omega_l(R_i) \rceil \leqslant \omega_l(R_i) + 1$. Thus there are a total of $O(j) = O(\omega_l(R_i) + 1)$ light-trails at link l. □

4.4 Merge Schedules of All Classes

If we simply collect together the wavelengths as allocated above, we would get a bound $O(\omega \log p)$. Note, however, that if two light-trails, one for transmissions in class i and the other for transmissions in class j, are spatially disjoint, then they could possibly share the same wavelength. Given below is a systematic way of doing this, which gets us a sharper bound.

Theorem 4.4. *The entire set R can be scheduled using $O(\omega + \log p)$ wavelengths in time $O(np\omega + n \log n)$.*

Proof. We know that after the modification in Lemma 4.3, at each link l there are a total of $O(\omega_l(R_i) + 1)$ light-trails for each class i. Thus summing over all classes, the total number of light-trails at l are $O(\omega_l(R) + \log p)$, and total time taken is $O(np\omega)$.

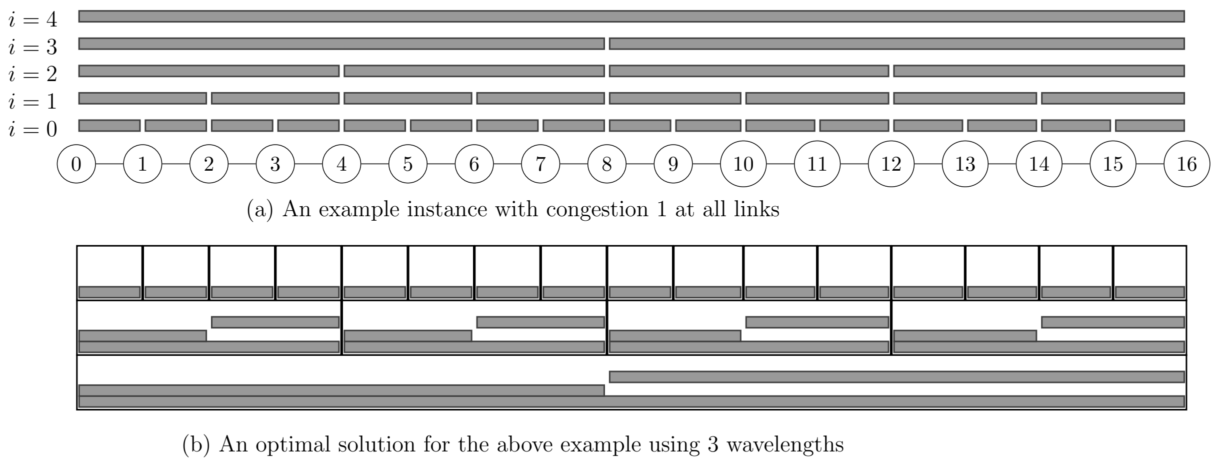
Think of each light-trail as an interval, giving us a collection of, say k, intervals such that any link l has at most $O(\omega_l(R) + \log p) = O(\omega + \log p)$ intervals. Now this collection of k intervals can be colored using $O(\omega + \log p)$ colors [72] in time $O(k \log k)$. Now for each color w, we use a separate wavelength and configure the light-trails corresponding to the intervals of color w by setting the shutters OFF at the processors corresponding to the endpoints of the intervals. Hence $O(\omega + \log p)$ wavelengths suffice. Overall it takes time $O(np\omega + n \log n)$ as k can be at most n. □

For the example in Figure 1.1, our algorithm creates two classes – one containing $[3, 10]$ and the other containing remaining transmissions. The first class uses a single wavelength on which light-trail $[3, 10]$ contains transmission $[3, 10]$ and the second class uses 2 more wavelengths. One wavelength contains light-trails $[0, 5]$ and $[7, 12]$ where light-trail $[0, 5]$ contains transmissions $[0, 4]$ and $[1, 5]$ and light-trail $[7, 12]$ contains transmissions $[7, 11]$ and $[8, 12]$. The other wavelength contains two light-trails $[2, 6]$ and $[9, 13]$ with transmissions $[2, 6]$ and $[9, 13]$ respectively.

4.5 On the Congestion Lower Bound

We show an instance of the stationary problem for which the congestion lower bound is weak. For convenience, we assume there are $p + 1$ processors numbered $0, \ldots, p$ where $p = 2^k$ for some k and all logarithms are with base 2. All the transmissions have the same bandwidth requirement $b = 1/(\log p + 1)$.

First, we have a transmission going from 0 to p. Then a transmission from 0 to $p/2$ and a transmission from $p/2$ to p. Then four transmissions spanning one-fourth the distance, and so on. In class $i \in \{0, 1, \ldots, \log p\}$ there are $p/2^i$ transmissions $B(s_{ij}, d_{ij}) = b$ where $s_{ij} = j2^i, d_{ij} = (j + 1)2^i$ for all $j = 0, 1, \ldots, (p/2^i) - 1$. All other entries of B are 0. This is illustrated in Figure 4.3(a) for $p = 16$. Clearly the congestion of this pattern is uniformly 1.



(a) An example instance with congestion 1 at all links

(b) An optimal solution for the above example using 3 wavelengths

Figure 4.3: An Example Instance where Congestion Bound is Weak

Consider an optimal solution for the splittable case. There has to be a wavelength with a light-trail in which the transmission $[0, p]$ is scheduled. This light-trail might have additional transmissions besides $[0, p]$. Clearly, we can assume without loss of generality that the light-trail contains the longest transmissions. Suppose the transmissions from the longest l classes are completely contained in this light-trail. Thus we have a total of at most $1 + 2 + 4 + \cdots +$

$2^l = 2^{l+1} - 1$ transmissions assigned to this light-trail. Total bandwidth requirement of these transmissions should be at most 1. This gives us $(2^{l+1} - 1)(1/(\log p + 1)) \leqslant 1$ implying $l \leqslant \log(\log p + 2) - 1 = O(\log\log p)$.

The remaining transmissions do not cross processors with numbers $qp/2^{l-1}$, for all integers q. Thus, we have 2^{l-1} separate parallel problems, each having $p/2^{l-1} = \Omega(p/\log p)$ processors. Thus the total number of wavelengths $W(p)$ needed must satisfy the recurrence $W(p) = 1 + W(p/\log p)$. This solves to $W(p) = \Omega(\log p/\log\log p)$.

Suppose we add $\omega - 1$ transmissions of extent $[0, p]$ and bandwidth requirement 1 to this pattern of transmissions of congestion 1 uniformly at all links. We can similarly show that an optimal solution will require $\omega - 1 + \Omega(\log p/\log\log p)$ wavelengths. Thus we will get an instance of congestion ω uniformly at all links but which requires $\Omega(\omega + \log p/\log\log p)$ wavelengths, for any ω.

4.6 Summary

It can be shown that the nonsplittable stationary problem is NP-hard on general interval graphs, using a simple reduction from bin-packing. We do not know if the splittable problem is also NP-hard. We gave an algorithm for both variations of the stationary problem which takes $O(\omega + \log p)$ wavelengths.

For the example in Figure 1.1 we have seen that the algorithm in Chapter 3 and the algorithm in this chapter use the same number of wavelengths though the light-trail configurations differ. However, there are examples where the algorithm in this chapter takes significantly more wavelengths. Let us consider the following example. For some parameters $k < L$, let there be $k + 1$ transmissions $[i, i + L^i]$ where $0 \leqslant i \leqslant k$ and each transmission has bandwidth requirement $1/(k + 1)$. These intervals are proper intervals. The optimal algorithm in Chapter 3 uses a single wavelength. However, the algorithm in this chapter will require $k + 1$ wavelengths each containing a single transmission in a single light-trail.

It will be useful to improve the lower bound arguments; as Section 4.5 shows, congestion is not always a good lower bound. This may lead to a constant factor approximation algorithm for the problem.

Chapter 5

Online Problems on Interval and Circular-arc Graphs

In this chapter we discuss our algorithms for the online problem on general interval and circular-arc graphs. For simplicity, here also we describe the algorithm on the light-trail scheduling problem. The analogous algorithm for online component coloring problem can be similarly described.

We start with the description of the online light-trail scheduling problem in Section 5.1. For the online problem, we present two algorithms – (1) SEPARATECLASS having competitive ratio $\Theta(\log p)$ in Section 5.2 and (2) ALLCLASS, a simplification of SEPARATECLASS in Sections 5.3. We show in Section 5.4 that this simplified algorithm, ALLCLASS has a competitive ratio in between $\Omega(\log^2 p/ \log\log p)$ and $O(\log^2 p)$. In Section 5.5 we show that every online algorithm must have competitive ratio $\Omega(\log p)$. In Section 5.6 we give results of simulation of our online algorithms.

5.1 Online Light-trail Scheduling

In the online case, the transmissions arrive dynamically. An arrival event has parameters (s_i, d_i, r_i) respectively giving the origin, destination, and bandwidth requirement of an arriving transmission request. The algorithm must assign such a transmission to a light-trail L such that s_i, d_i belongs to the light-trail, and at any time the total bandwidth requirement of transmissions assigned to any light-trail is at most 1. A departure event marks the completion of a previously scheduled transmission. The corresponding bandwidth is released and becomes available for future transmissions. The algorithm must make assignments without knowing about subsequent events.

Unlike the stationary problem, congestion at any link may change over time. Let $\omega_{lt}(S)$ denote the congestion induced on a link l at time t by a set of transmissions S. This is simply the total bandwidth requirement of those transmissions from S requiring to cross link l at time t. The congestion lower bound $\omega(S)$ is $\max_l \max_t \omega_{lt}(S)$, the maximum congestion over all links over all time instants.

In both of our online algorithms, when a transmission request t arrives, we first determine its class i and trail-point x. Recall that the class of t is i if t has length in the range $(2^{i-1}, 2^i]$ and the *trail-point* (defined in Section 4.3) of t is the smallest processor $q2^{i-1}$ such that t crosses processor $(q+1)2^{i-1}$. In both algorithms, transmission t is allocated to some light-trail $[x, x+2^{i+1}]$. However, the algorithms differ in the way a light-trail is configured on some candidate wavelength.

5.2 Algorithm SeparateClass

In this algorithm, every allocated wavelength is assigned a class label i and a phase label δ, and has shutters OFF at processors $(4q+\delta)2^{i-1}$ for all q, *i.e.,* is configured to serve only transmissions of class i and phase δ. Whenever a transmission t of class i and phase δ is to be served, it is only served by a wavelength with the same labels. If such a wavelength w is found, and a light-trail L on w starting at the trail-point of t has space, then t is assigned to the light-trail L. If no such wavelength is found, then a new wavelength w' is allocated, it is labeled and configured for class i and phase δ as above and t is assigned to the light-trail on w' that starts at the trail-point of t.

When a transmission finishes, it is removed from its associated light-trail. When all transmissions in a wavelength finish, then its labels are removed, and it can subsequently be used for other classes or phases.

Time complexity: for a class i and phase δ there are $p/2^i$ possible light-trails. For each of these light-trails, we can maintain a list of wavelengths on which the light-trail is present. So on an arrival, searching for a candidate light-trail takes $O(w)$ time where w is the number of wavelengths used. On departure also, it takes $O(w)$ time.

Lemma 5.1. *Suppose, at some instant of time, among the wavelengths allocated by* SEPARATE-

CLASS, x wavelengths had non-empty light-trails of the same class and phase across a link l'. Then there must be a link l having congestion $\Omega(x)$ at some instant of time.

Proof. Suppose at some instant of time, wavelengths $w_1, w_2, \ldots, w_x$, ordered according to the time of allocation, had non-empty light-trails $L_1, L_2, \ldots, L_x$, respectively, of same class and phase across link l'. Let u be the anchor (defined in Section 4.3) of the transmissions assigned on these light-trails and l be the link between processor u and processor $u + 1$.

Now suppose wavelength w_x was allocated due to a transmission t. This could only happen because t could not fit in the wavelengths w_j for all $j \leqslant x - 1$.

For the splittable case this can only happen if light-trails L_1 through L_{x-1} together contain transmissions of congestion at least $x - 1 - \omega(t) = \Omega(x)$ crossing the anchor u of t, when t arrived. Thus at that time l had congestion $\Omega(x)$, giving us the result.

For the nonsplittable case, suppose that $\omega(t) \leqslant 0.5$. Then the transmissions in each of the light-trails $L_j, 1 \leqslant j \leqslant x - 1$, must have congestion of at least 0.5 at l when t arrived, giving congestion $\Omega(x)$. So suppose $\omega(t) > 0.5$. Let k be the largest such that light-trail L_k contains a transmission t' with $\omega(t') \leqslant 0.5$ when t arrived. If no such k exists, then clearly the congestion at l when t arrived is $\Omega(x)$. If k exists, then all the light-trails L_j, $j > k$ have transmissions of congestion at least 0.5 at l when t arrived. And the light-trails L_j, $j \leqslant k$ had transmissions of congestion at least 0.5 at l when t' arrived. So at one of the two time instants the congestion at l must have been $\Omega(x)$. $\square$

Theorem 5.2. SEPARATECLASS *is $\Theta(\log p)$ competitive.*

Proof. Suppose that SEPARATECLASS uses w wavelengths. We will show that the best possible algorithm (including off-line algorithms) must use at least $\Omega(w/\log p)$ wavelengths. That will prove that SEPARATECLASS is $O(\log p)$ competitive.

Consider the time at which the wth wavelength was allocated by SEPARATECLASS. At this time $w - 1$ wavelengths are already in use, and of these at least $w' = (w - 1)/(4 \log p)$ must have the same class and phase. Among these w' wavelengths consider the one which was allocated last to accommodate some light-trail L serving some newly arrived transmission. At that time, each of the previously allocated $w' - 1$ wavelengths was nonempty in the extent of L. By Lemma 5.1, there is a link that had congestion $\Omega(w' - 1) = \Omega(((w - 1)/(4 \log p)) - 1) = \Omega(w/\log p)$ at some time instant. This is a lower bound on any algorithm, even off-line. Thus the competitive ratio of SEPARATECLASS is $O(\log p)$.

We show the lower bound $\Omega(\log p)$ using the following example. Let $p = 2^k + 1$. At each time $t = 0, 1, \ldots, k$, a transmission $[0, 2^t]$ arrives. All transmissions have bandwidth requirement $1/(k+1)$. At time $k+1$ all transmissions depart together. SEPARATECLASS takes k wavelengths because each transmission is of a different class. The optimal off-line algorithm assigns all of them to a single light-trail spanning the entire network and hence takes only one wavelength. □

5.3 Algorithm ALLCLASS

This is a simplification of SEPARATECLASS in that the allocated wavelengths are not labeled. When a transmission t of class i and trail-point x arrives, we search the wavelengths in the order they were allocated for a light-trail L of extent $[x, x + 2^{i+1}]$ such that L has enough space to serve t. If such a light-trail L is found, then t is assigned to L. If no such light-trail is found, then an attempt is made to create a light-trail $[x, x+2^{i+1}]$ from the unused portions of one of the existing wavelengths in a first-fit manner in the order they were allocated. If such a light-trail L can be created, then L is created and t is assigned to L. Otherwise a new wavelength w is allocated, the required light-trail L of extent $[x, x + 2^{i+1}]$ is created on w, and t is assigned to L. The portion of the wavelength w outside the extent of L is marked unused.

When a transmission finishes, it is removed from its associated light-trail. If this makes the light-trail empty then we mark its extent on the corresponding wavelength as unused.

Time complexity: using a binary search tree based data structure for the light-trails and the transmissions, the algorithm can be implemented in $O(\log n + w)$ time on each arrival and in time $O(\log n)$ on each departure where n is the number of active requests and w is the number of wavelengths used.

Theorem 5.3. ALLCLASS *is* $O(\log^2 p)$ *competitive.*

Proof. Suppose ALLCLASS uses w wavelengths. Since the optimal must use at least one, we only need consider the case $w = \Omega(\log^2 p)$.

The key idea is to argue that at some time during the execution of ALLCLASS there will be least $w/(4 \log p)$ non-empty light-trails (not necessarily of the same class and phase) crossing the same link. If this holds, then of these light-trails, at least $w/(16 \log^2 p)$ must have the same class and phase. But it can be shown that Lemma 5.1 is also true for ALLCLASS, and hence there is a link having congestion $\Omega(w/(16 \log^2 p))$ at some time instant. But this is a lower bound on the number of wavelengths required by any algorithm, including an off-line algorithm. Thus the competitive ratio of AllClass is at most $O(\log^2 p)$.

Number the wavelengths in the order of allocation. Consider the transmission t for which the wth wavelength was allocated for the first time. Let L be the light-trail used for t. Clearly, the wth wavelength had to be allocated because at that time the $w-1$ previously allocated wavelengths contained light-trails overlapping with L. Let S' denote this set of light-trails, each from a different wavelength, but overlapping with L.

If S' contains at least $w/(4\log p)$ light-trails which cross the leftmost link in L or the rightmost link, we are done. So assume the contrary. Thus there must be at least $w' = w - 1 - 2w/(4\log p) = w - 1 - w/(2\log p)$ in S' whose extent is completely contained in the extent of L. Among these light-trails, let L' be the largest numbered. Note that L' is strictly smaller than L. Thus we can repeat the above argument by using L' and w' in place of L and w respectively, only $\log p$ times, and if we fail each time to find at least $w/(4\log p)$ light-trails crossing a link, we will end up with a light-trail L'' such that there are at least w'' wavelengths having light-trails conflicting with L'', where $w'' = w - \log p - \log p(w/(2\log p)) = w/2 - \log p \geqslant w/(4\log p)$ for $w = \Omega(\log^2 p)$. But L'' is a single link and so we are done. □

5.4 Lower Bound for AllClass

We give a sequence of transmissions for which AllClass takes $\Omega(\log^2 p/\log\log p)$ wavelengths but an optimal off-line algorithm, Opt, requires only one wavelength.

Theorem 5.4. AllClass *is* $\Omega(\log^2 p/\log\log p)$ *competitive.*

Our transmission sequence consists of several (the exact count will be shown later) subsequences, which we call stages. In all stages, all transmissions have a height (*i.e.,* bandwidth requirement) of $1/p^2$. Our transmission sequence is such that, at any point of time, there are less than p^2 active transmissions. Opt will put all transmissions in a single light-trail using the full length of a wavelength. On the other hand, it will be seen that AllClass will allocate $\Omega(\log^2 p/\log\log p)$ wavelengths in total for all stages. We describe the first stage only; the other stages are scaled versions of the first stage. The goal of the first stage is to force AllClass to allocate wavelengths with light-trail patterns given in the following lemma.

Lemma 5.5. *Let the network have* $q+1$ *processors numbered* $0,\ldots,q$. *Then there is a transmission sequence for which* AllClass *allocates* $k = \lfloor \log q \rfloor$ *wavelengths numbered* $0,\ldots,k-1$, *with the following* staircase *pattern: each wavelength i has $\lfloor p/k \rfloor$ unit-length light-trails $[jk + i, jk + i + 1]$, each containing a single transmission, for all $j = 0, \ldots, \lfloor p/k \rfloor - 1$.*

Proof. For simplicity we assume $q = 2^k$, *i.e.,* k is exactly equal to $\log q$. The general case can be similarly proved.

We first describe how to create a unit-length light-trail $[x, x + 1]$ on any wavelength h. We will repeatedly use this procedure to create our pattern. Define $Hill(h, x)$ to be an ordered sequence of h transmissions as follows. For each $i = 0, 1, \ldots, h - 1$, $Hill(h, x)$ contains a transmission that uses the link $[x, x + 1]$ and has class $k - 1 - i$, and some suitable phase. The key point is that all the transmissions in a hill overlap but have different classes, and hence AllClass must assign them in distinct light-trails on different wavelengths. Thus starting from scratch, the arrival of the transmissions in a hill will cause h wavelengths to be allocated. For example, we show $Hill(h = 4, 31)$ on right half of Figure 5.1(a). Further, if a new transmission $[x, x + 1]$ arrives, it will cause one more wavelength to be allocated. From now on, by creating (deleting) a hill we mean the arrival (departure) of transmissions in a hill.

Now we describe how to generate the staircase using several hills. The idea is to build the staircase one wavelength at a time from top to bottom, *i.e.,* first create all light-trails of the staircase on wavelength $k - 1$, then all light-trails on wavelength $k - 2$ and so on. Each unit-length light-trail $[x, x + 1]$ on an wavelength is created by temporarily creating an appropriate hill underneath it, then creating the transmission $[x, x + 1]$ and finally deleting the temporary hill. The left half set of staircases is created first, and then the right half set.

Before creating the left half, we first create hill $H = Hill(k - 1, q - 1)$. This hill will survive until the left half is completely created. Its sole purpose is to ensure that the numbering of its $k - 1$ wavelengths does not change as the left half is created. The left half is created top to bottom as given in Algorithm 5. Consider the first execution of the insertion marked as belonging to the staircase. Because of the hill H', this transmission will clearly be assigned to a light-trail on wavelength i. Note further that when transmissions in H' depart, the wavelengths $0, \ldots, i - 1$ do not become empty because of the presence of hill H in the right half. Thus the subsequent iterations also force the transmissions to be assigned in wavelength i, and so on.

At the end of the above, we will have created a pattern as shown in Figure 5.1(a). Since each light-trail contains only one transmission, we just show the transmissions instead of the light-trails.

Next we remove H, and execute the same code to create the right half of the staircase on

Algorithm 5: Create left half of the staircase

```
for $i = k - 1$ downto 0 do
  for $j = 0$ to $q/2k$ do
    Create a hill $H' = Hill(i, jk + i)$
    Insert an arrival event for transmission $[jk + i, jk + i + 1]$ {belongs to staircase}
    Remove hill $H'$
  end for
end for
```

the k wavelengths already allocated. Note that the light-trails created in the left half now serve the purpose that H did earlier. At this point we will have the complete staircase. □

The first stage is created by using Lemma 5.5 with $q = p$. In the second stage, we can treat every $k = \lfloor \log p \rfloor$ processors as a single processor, and think of the network as having $p' = \lfloor p/k \rfloor$ processors. We create a staircase of height $\lfloor \log p' \rfloor$ but with light-trails of length k using Lemma 5.5 with $q = p'$. Since these light-trails are longer than the light-trails in the previous stage, we can stack up the new pattern on top of the previous pattern. We can keep doing this until p' becomes less than 2. Thus the number of stages is $\Omega(\log_{\log p} p) = \Omega(\log p / \log \log p)$.

Let $T(p)$ denote the total height of the patterns thus created for $p+1$ processors, then $T(p)$ is computed using the following recurrence:

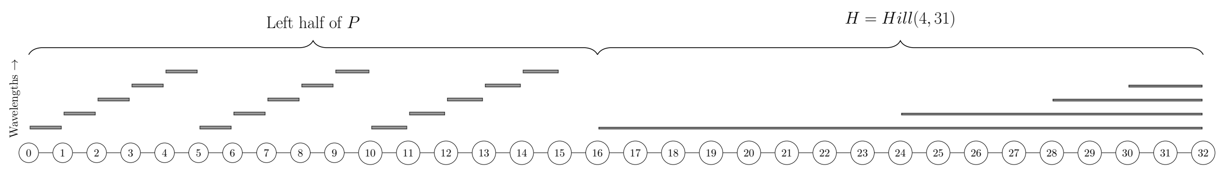
$$T(p) = \lfloor \log p \rfloor + T(\lfloor p/\lfloor \log p \rfloor \rfloor) \qquad \text{or simply} \qquad T(p) = \log p + T(p/\log p) \tag{5.1}$$

with the base condition $T(p) = 0$ for $p \leqslant 1$. It can be seen that the recurrence has solution $T(p) = \Omega(\log^2 p / \log \log p)$.

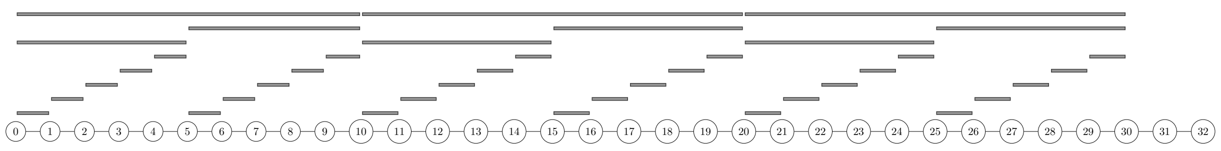
Thus AllClass will use $\Omega(\log^2 p / \log \log p)$ wavelengths for the patterns created. Figure 5.1(b) shows all the transmissions active at the end of all stages, for the example considered in Figure 5.1(a).

5.4.1 Remarks

It is interesting to note that AllClass is more flexible than SeparateClass, and it is this flexibility that is exploited in the lower bound argument to show a worse ratio for AllClass than SeparateClass.



(a) Pattern Created Midway of the First Stage



(b) Final Pattern

Figure 5.1: An Instance for which ALLCLASS is $\Omega(\log^2 p/\log\log p)$-competitive

Indeed, the more flexibility we give, the worse it seems the ratio will become. In ALLCLASS, if we have a transmission of length $L = 2^k$ we assign it to a light-trail of length $2L$. This seems wasteful. But this is done to accommodate transmissions that do not start at a multiple of $L/4$, using only 4 phases. Suppose we decide to be more flexible, and allow light-trails to start anywhere (so long as their length is 2^k for some k) using 2^k phases. Although this strategy will handle the above transmission better, in general it is worse in that its competitive ratio can be shown to be $\Omega(\log^2 p)$.

5.5 Problem Lower Bound – $\Omega(\log p)$

Theorem 5.6. *Every online algorithm has competitive ratio* $\Omega(\log p)$.

Let ALG be any algorithm for the online problem and OPT be an optimal off-line algorithm. By observing the behavior of ALG we can create a sequence of transmissions for which ALG takes $\Omega(\log p)$ times as many wavelengths as OPT. This will prove the theorem.

For convenience we assume that the network has $p + 1$ processors numbered $0, 1, \ldots, p$ and $p = 2^k$ for some k. Our transmission sequence will have $k = \log p$ stages. For stage $i = 0, 1, \ldots, k - 1$, consider the network broken up into $p' = p/2^i$ intervals of length 2^i. Let this set of intervals be $Q_i = \{[q2^i, (q + 1)2^i]\}_{q=0}^{p'-1}$. At the beginning of the ith stage, for each interval $I \in Q_i$, k^2 transmissions having extent I arrive. We will denote this set of $p'k^2$ transmissions by A_i. All transmissions have a height (*i.e.,* bandwidth requirement) of $1/k$. At the end of the ith stage, all but a subset S_i of A_i depart. The set S_i is determined by observing the behavior of ALG.

Lemma 5.7. *Among the $p'k^2$ transmissions arriving at the beginning of stage i, we can find a set S_i of $p'k$ transmissions such that (1) Exactly k transmissions from S_i are for a single interval $I \in Q_i$, (2)* ALG *assigns each transmission in S_i to a distinct light-trail.*

Proof. We have k^2 transmissions for each interval $I \in Q_i$. Partition these k^2 transmissions arbitrarily into k groups of k transmissions each. So overall we have $p'k$ groups each containing k transmissions. Now form a bipartite graph (U, V, E) as follows.

1. U has $p'k$ vertices, each vertex corresponding to a group of k transmissions as formed above. Note that there are k groups for each interval, and hence we can consider a distinct group of k vertices of U to be associated with each $I \in Q_i$.

2. V has a vertex corresponding to each light-trail used by ALG for serving the transmissions of this stage.

3. E has following edges. Suppose a transmission t from the group associated with a vertex $u \in U$ is placed by ALG in the light-trail L associated with a vertex $v \in V$. Then for each such t there will be an edge (u, v) in E. Note that this may produce parallel edges if several transmissions in the group of u are placed in L.

The degree of each vertex in U is exactly k, one edge for each transmission in the associated group. Consider any vertex $v \in V$. Since its associated light-trail can accommodate at most k transmissions of height $1/k$, its degree must be at most k.

Now consider any subset S of vertices from U and its neighborhood T in V. Because vertices in U have degree exactly k there must be exactly $|S|k$ edges leaving S. These must be a subset of the edges entering T. But vertices in V have degree at most k. So there can be at most $|T|k$ edges entering T. Thus we have $|S|k \leqslant |T|k$, *i.e.,* $|T| \geqslant |S|$, *i.e.,* S has at least as many neighbors as its own cardinality. But this is true for any S. Thus by Hall's theorem [96, pp-110], there must be a matching M that includes an edge from every vertex of U to a distinct vertex in V.

Consider the set S_i of transmissions associated with each edge of M. Since there is exactly one edge in M for each processor in U, S_i has one transmission per group of transmissions for each interval. Hence S_i has exactly k transmissions for each interval. Since M has exactly one edge per vertex in V, we know that each transmission in S_i is assigned to a distinct light-trail by ALG. □

We have now completely described the transmission sequence. At the end all transmissions have departed except those in some S_i. We will use D_i to denote the transmissions which depart in stage i. Clearly $A_i = S_i \cup D_i$.

Lemma 5.8. OPT *uses overall* $2k-1$ *wavelengths while processing the transmission sequence for all stages.*

Proof. Consider stage i. The set A_i has k^2 transmissions for each interval $I \in Q_i$. To serve these transmissions A_i, OPT uses k wavelengths configured as follows. Each wavelength is configured into light-trails as per Q_i, *i.e.,* each interval $I \in Q_i$ forms one light-trail. Now the key point is that OPT places all transmissions in S_i into light-trails on a single wavelength. This can be done because the set S_i indeed has k transmissions for each $I \in Q_i$. The remaining transmissions D_i can be accommodated into $k-1$ additional wavelengths. Note now that at the end of the stage, the transmissions D_i depart. Hence although the stage used k wavelengths transiently, at the end $k-1$ of these are released.

Thus, at the end of stage i, there will be $i+1$ wavelengths in use, one for transmissions in each S_j, $j = 0, \ldots, i$. When A_{i+1} arrives, OPT will allocate k new wavelengths. So while processing A_{i+1} there will be $i+1+k$ wavelengths in use. These will drop down to $i+2$ at the end of stage $i+1$. Thus, over all the stages the maximum number of wavelengths used will be at most $\max_{i=0,\ldots,k-1}(i+k)$, *i.e.,* $2k-1$. □

Lemma 5.9. ALG *uses at least* $k^2/2$ *wavelengths while processing the transmission sequence.*

Proof. Consider the light-trails used by ALG which are active at the end of the stage $k-1$. Each of these light-trails may contain several transmissions but only one transmission from each S_i. Since transmissions from each S_i have different lengths, each light-trail must hold transmissions of different lengths. Thus, each light-trail can have at most one transmission of length 1, one of length 2, and so on. The sum of the lengths of the transmissions assigned to a single light-trail of length l is thus at most $1+2+4+\cdots+l = 2l-1 \leqslant 2l$. But this applies to all light-trails in any wavelength, and hence the total length of the transmissions assigned to a single wavelength is at most $2p$. However, the transmissions that survive at the end consist of nk transmissions of length 1, $nk/2$ transmissions of length 2, and so on to $2k$ transmissions of length $p/2$. Thus the total length is nk^2. Thus ALG needs at least $(nk^2)/(2p) = k^2/2$ wavelengths at the end. □

But OPT requires at most $2k-1 < 2k$ wavelengths. Hence the competitive ratio is at least $(k^2/2)/2k = k/4 = \Omega(\log p)$. This completes the proof of Theorem 5.6.

5.6 Simulations

We simulate our two online algorithms and a baseline algorithm on a pair of oppositely directed rings, with processors numbered 0 through $p-1$ clockwise.

We use slightly simplified versions of the algorithms described in Sections 5.2 and 5.3 (but easily seen to have the same bounds): basically we only use phases 0 and 2. Any transmissions that would go into class i phase 1 (or phase 3) light-trail are contained in some class $i+1$ light-trail (of phase 0 or 2 only), and are put there. We define a class i and phase 0 light-trail to be one that is created by putting OFF shutters at processors $jp/2^i$ for different j, suitably rounding when p is not a power of 2. A light-trail with class i and phase 2 is created by putting OFF shutters at processors $(jp/2^i + p/2^{i+1})$, again rounding suitably. The class and phase of a transmission is determined by the light-trail of maximum class (note that now larger classes have shorter light-trails) and minimum phase that can completely accommodate it. For ALLCLASS, there is a similar simplification. Basically, we use light-trails having end processors at $jp/2^i$ and $(j+1)p/2^i$ or at $jp/2^i + p/2^{i+1}$ and $(j+1)p/2^i + p/2^{i+1}$. As before, in SEPARATECLASS, we require any wavelength to contain light-trails of only one class and phase, whereas in ALLCLASS, a wavelength may contain light-trails of different classes and phases.

For the baseline algorithm in each ring we use a single OFF shutter at processor 0. Transmissions from lower numbered processors to higher numbered processors use the clockwise ring, and the others, the counterclockwise ring.

5.6.1 The Simulation Experiment

A single simulation experiment consists of running the algorithms on a certain load, characterized by parameters λ, D, r_{min} and α for 100 time steps. In our results, each data-point reported is the average of 150 simulation experiments with the same load parameters.

In each time step, all processors j that are not busy transmitting, generate a transmission (j, d_j, r_j) active for t_j time units. After that the processor is busy for t_j steps. After that it generates another transmission as before. The transmission duration t_j is drawn from a Poisson distribution with parameter λ. The destination d_j of a transmission is picked using the distribution D discussed later. The bandwidth is drawn from a modified Pareto distribution with scale parameter $= 100 \times r_{min}$ and shape parameter $= \alpha$. The modification is that if the generated bandwidth requirement exceeds the wavelength capacity 1, it is capped at 1.

We experimented with $\alpha = \{1.5, 2, 3\}$ and $\lambda = \{0.01, 0.1\}$ but report results for only $\alpha = 1.5$ and $\lambda = 0.01$; results for other values are similar. We tried four values $0.01, 0.1, 0.25$ and 0.5 for r_{min}. We considered four different distributions D for selecting the destination processor of a transmission.

1. *Uniform*: we select a destination uniformly randomly from the $p-1$ processors other than the source processor.

2. *UniformClass*: we first choose a class uniformly from the $\lceil \log p/2 \rceil + 1$ possible classes and then choose a destination uniformly from the processors possible for that class. It should be noted that there can be a destination at a distance at most $p/2$ in any direction since we schedule along the direction requiring the shortest path.

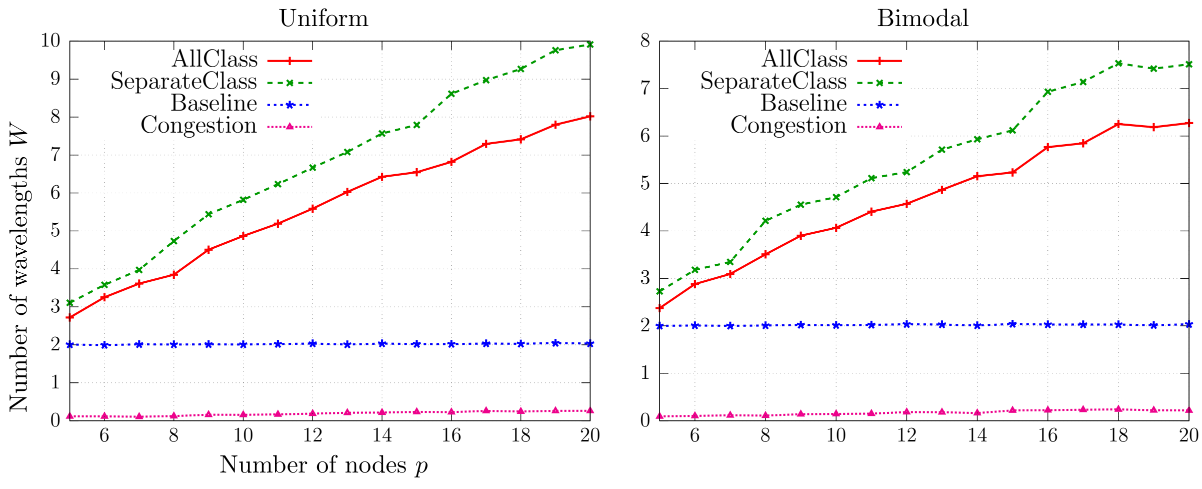
3. *Bimodal*: first we randomly choose one of two possible modes. In mode 1, a destination from the two immediate neighbors is selected and in mode 2, a destination from the processors other than the two immediate processors is chosen uniformly. For applications where transmissions are generated by structured algorithms, local traffic, *i.e.,* unit or short distances (e.g. $\sqrt{p}$ for mesh-like communications) would dominate. Here, for simplicity, we create a bimodal traffic which is a mixture of completely local and completely global.

4. *ShortPreferred*: we select destinations at shorter distance with higher probability. In fact, we first choose a class i in the range $0, \ldots, \lceil \log p/2 \rceil$ with probability $\frac{1}{2^{i+1}}$ and then select a destination uniformly from the possible destinations in that class.

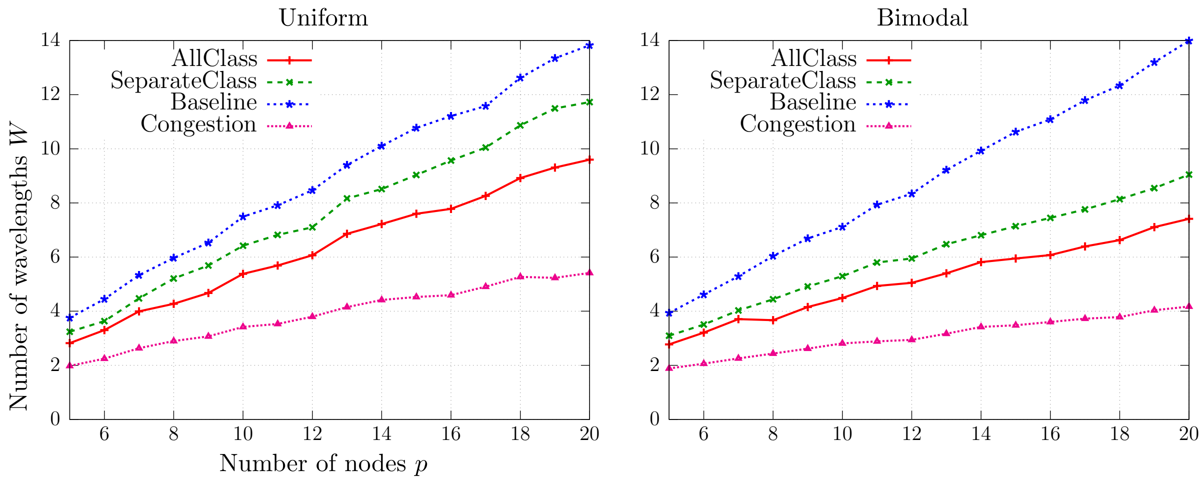
We report the results only for the distributions *Uniform* and *Bimodal* and for $r_{min} = 0.01, 0.5$, *i.e.,* a total of 4 load scenarios. Results for other scenarios follow a similar pattern.

5.6.2 Results

Figure 5.2 shows the results for the 4 load scenarios. For each scenario, we report the number of wavelengths required by the 3 algorithms and the measured congestion as defined in Section 5.1. Each data-point is the average of 150 simulations (each of 100 time steps) for the same parameters on rings having $p = 5, 6, \ldots, 20$ processors. We say that the two scenarios corresponding to $r_{min} = 0.01$ have *low load* and the remaining two scenarios ($r_{min} = 0.5$) have *high load*.



(a) Low Load



(b) High Load

Figure 5.2: Simulation Results

For low load, the baseline algorithm outperforms our algorithms. At this level of traffic, it does not make sense to reserve different light-trails for different classes. However, as load increases our algorithms outperform the baseline algorithm.

For the same load, it is also seen that our algorithms become more effective as we change from the completely global *Uniform* distribution to the more local *Bimodal* distribution. This trend was also seen with the other distributions we experimented with.

It is also to be noted that ALLCLASS performs better than SEPARATECLASS in our simulations. This is perhaps surprising because in Section 5.4 we showed that SEPARATECLASS has a better competitive ratio. Indeed, in that section we presented an input instance on which ALLCLASS performs substantially worse than SEPARATECLASS. But there is no contradiction here. The simulation results merely indicate that instances like the one we presented do not appear in our workload. For our workload, perhaps the extra flexibility of ALLCLASS is very useful. So we feel that in practice the algorithm ALLCLASS is an important candidate.

5.7 Summary

For the online problem we proved that the lower bound on the competitive ratio of any algorithm is $\Omega(\log p)$ and gave a matching algorithm which we proved to have competitive ratio $\Theta(\log p)$. We also gave a second algorithm which seems to work better in practice but can be as bad as $\Omega(\log^2 p/\log\log p)$ factor worse than an optimal off-line algorithm on some pathological examples as we have shown. We also proved an upper bound of $O(\log^2 p)$ for the algorithm but it will be an interesting problem to close the gap between the two bounds.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Our work on the component coloring problem is summarized in Table 6.1. Except for the stationary unweighted and nonsplittable weighted problem on PIGs, to the best of our knowledge, all our results are the first theoretical results in the respective category. For PIGs our results are superior than the existing results.

The simplest case of the component coloring problem that we considered is the unweighted stationary version on PIGs. We showed that this problem is equivalent to solving a vertex partitioning problem which turned out to be simpler to solve. We gave an $O(n)$ time algorithm for this partitioning problem. A slight modification of the partitioning procedure gave an $O(n^2)$ time algorithm for the splittable weighted problem on PIGs, assuming that the weights are at most C. This algorithm for splittable weighted problem was later extended to give a 2-approximation algorithm for the nonsplittable weighted problem which is NP-hard even on PIGs.

Unfortunately, on general interval graphs, solving the simpler vertex partitioning problem is not enough to solve the component coloring problem and hence we needed to devise a new technique. Here we first classified the vertices according to the length of the transmissions in the corresponding light-trail scheduling problem where class i contained all transmissions of length in the range $(2^{i-1}, 2^i]$ and solved each class separately using at most a constant times the optimal number of wavelengths. As there are $\log p$ classes where p denotes the number of processors in the light-trail scheduling problem, this gave a $O(\log p)$ approximation algorithm. However, a clever technique of merging schedules from different classes gave a constant factor

Table 6.1: Current Status of Component Coloring. Here n, ω are the number of vertices and the clique number, respectively, of the input graph, p is the number of distinct endpoints in an interval or circular-arc representation of the input graph, and in general, p is independent of n.

Input	Graph	Unweighted	Splittable	Nonsplittable
stationary	PIG	• $O(n)$ time algorithm	• $O(n^2)$ time algorithm	• NP-hard • 2-approximation algorithm
	interval	• complexity unknown • approximation algorithm with bound $O(\omega + \log p)$		• NP-hard • approximation algorithm with bound $O(\omega + \log p)$
	circular-arc	• NP-hard • approximation algorithm with bound $O(\omega + \log p)$		
online	all three classes	• algorithm SEPARATECLASS with competitive ratio $\Theta(\log p)$ • algorithm ALLCLASS with competitive ratio in between $\Omega(\log^2 p/\log\log p)$ and $O(\log^2 p)$ • problem lower bound $\Omega(\log p)$ proving optimality of SEPARATECLASS		

algorithm up to an additive term of $O(\log p)$. However, we do not know if there is a way to do away with this additive factor or if there is a polynomial time algorithm for the unweighted stationary problem on interval graphs.

In our first algorithm SEPARATECLASS for the online problem we used the same strategy of dividing the transmissions into classes according to their lengths and scheduling each class separately in a simple first-fit order. It turned out that this strategy is efficient enough to give an optimal algorithm as we showed that SEPARATECLASS is $\Theta(\log p)$ competitive and no online algorithm can have better competitive ratio than $\Omega(\log p)$.

Inspired by our stationary algorithm where merging schedules from different classes gave better performance, we also considered a second algorithm ALLCLASS, a variation of SEPARATECLASS, where light-trails of different classes are created from the same pool of wavelengths. Though ALLCLASS performs better in the traffic loads that we simulated, in general, ALLCLASS performs worse than SEPARATECLASS. We proved that the competitive ratio of ALLCLASS is between $\Omega(\log^2 p/\log\log p)$ and $O(\log^2 p)$.

6.2 Future Work

The most important open problem is finding an algorithm for the unweighted component coloring problem for general interval graphs, or showing that this is NP-hard. Also, we have given an approximation algorithm which has an additive $\log$ term; it would be good to remove this term and get a constant factor approximation algorithm.

Our online model is very conservative: once a transmission is allocated on a light-trail, it cannot be moved to another light-trail, nor can the light-trail grow or shrink. However, there are models [37] which allow light-trails to shrink/grow dynamically, and those in which it is possible to transfer active transmissions from one light-trail to another [38]. It will be useful to incorporate these (with some suitable penalty, perhaps) into our model.

In the online case it would be interesting to devise special algorithms that work well given the distribution of arrivals.

It would be interesting to get similar results for more complicated topologies such as trees, mesh networks etc.

Consider the following variation of the basic light-trail scheduling problem, some sort of a dual. We have a light-trail based WDM network with p processors, as usual on a ring. For a processor to be able to transmit on k wavelengths, it must have a separate optical transponder for each wavelength. To receive it must have one transponder, which can receive on any wavelength. Suppose that there are λ wavelengths available overall. We are given a matrix R in which $R[i,j]$ gives the bandwidth needed from i to j. We need to solve the following:

1. Is it possible to assign the required bandwidth to each processor (for its different transmissions) satisfying all constraints of a light-trail based network?

2. Given feasibility as above, what is the minimum number of transponders needed, and which processors should have them? This is sort of a network design question.

The light-trail problem is similar to the following simplified problem of vehicle routing problem. We call this as *Bus Planning Problem.* We have the real line on which we are given the starting points s_i and destinations d_i of the ith among some p passengers. We are to plan the movement of these passengers using buses. Each bus has a certain starting point and destination of our design, and may carry at most B passengers at any time. A passenger cannot be asked to change buses, i.e. starting point of the bus carrying him must be smaller than the starting point of the passenger, and the destination larger (assume all movement is left to right). The cost function is the sum of the distances traveled by buses. Note that a certain bus might carry fewer than B passengers, but its distance fully counts in the cost. We need to see if our algorithms can be used in this problem too.

Bibliography

[1] S. Albers. Online Algorithms: A Survey. *Mathematical Programming*, 97(1):3–26, 2003. 8, 20

[2] N. Alon, G. Ding, B. Oporowski, and D. Vertigan. Partitioning into Graphs with Only Small Components. *Journal of Combinatorial Theory, Series B*, 87(2):231–243, 2003. 13

[3] J.A. Andrews and M.S. Jacobson. On a Generalization of Chromatic Number and two Kinds of Ramsey Numbers. *Ars Combinatoria*, 23:97–102, 1987. 13, 14

[4] K. Appel, W. Haken, and J. Koch. Every Planar Map is Four Colorable. Part II: Reducibility. *Illinois Journal of Mathematics*, 21(3):491–567, 1977. 12, 13

[5] J. Araujo, J.C. Bermond, F. Giroire, F. Havet, D. Mazauric, and R. Modrzejewski. Weighted Improper Colouring. *Journal of Discrete Algorithms*, 2012. 13, 14

[6] A.S. Ayad, K.M.F. Elsayed, and S.H. Ahmed. Enhanced Optimal and Heuristic Solutions of the Routing Problem in Light-trail Networks. *Workshop on High Performance Switching and Routing (HPSR)*, pages 1–6, 2007. 17, 18, 19

[7] S. Balasubramanian, W. He, and A.K. Somani. Light-Trail Networks: Design and Survivability. *Thirtieth IEEE Conference on Local Computer Networks*, pages 174–181, 2005. 17, 18, 19

[8] S. Balasubramanian, A.E. Kamal, and A.K. Somani. Network design in IP-centric Light-trail networks. In *Second International Conference on Broadband Networks (Broadnets)*, pages 41–50, 2005. 17

[9] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster. The Power of Reconfiguration. *Journal of parallel and distributed computing*, 13(2):139–153, 1991. ISSN 0743-7315. 20

[10] R. Berke and T. Szabó. Relaxed Two-coloring of Cubic Graphs. *Journal of Combinatorial Theory, Series B*, 97(4):652–668, 2007. 13, 14

[11] K.A. Berman and J.L. Paul. Large Monochromatic Components in Vertex Colored k-trees. *Congressus numerantium*, 53:161–166, 1986. 13, 14

[12] J.C. Bermond, F. Havet, F. Huc, and C.L. Sales. Improper Coloring of Weighted Grid and Hexagonal Graphs. *Discrete Mathematics, Algorithms and Applications*, 2(03):395–411, 2010. 13, 14

[13] K. Bondalapati and V.K. Prasanna. Reconfigurable Meshes: Theory and Practice. In *Fourth Workshop on Reconfigurable Architectures (IPPS)*, 1997. 20

[14] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, New York, NY, USA, 1998. 8, 20

[15] I. Broere and M. Frick. On the Order of Color Critical Graphs. In *Congressus Numerantium*, volume 47, pages 125–130, 1985. 13, 14

[16] G. Chartrand, D.P. Geller, and S. Hedetniemi. A Generalization of the Chromatic Number. In *Proceedings of the Cambridge Philosophical Society*, volume 64, pages 265–271. Cambridge University Press, 1968. 13, 14

[17] G. Chartrand, H.V. Kronk, and C.E. Wall. The Point-arboricity of a Graph. *Israel Journal of Mathematics*, 6(2):169–175, 1968. 13, 14

[18] I. Chlamtac and A. Gumaste. Light-trails: A Solution to IP Centric Communication in the Optical Domain. *Lecture Notes in Computer Science*, pages 634–644, 2003. 1, 17

[19] E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing: a survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1997. 50, 51, 53

[20] K.C. Dargen and K. Fraughnaugh. Conditional Chromatic Numbers with Forbidden Cycles. *Linear Algebra and its Applications*, 217:53–66, 1995. 12, 14

[21] G. Ding, B. Oporowski, D.P. Sanders, and D. Vertigan. Partitioning Graphs of Bounded Tree-width. *Combinatorica*, 18(1):1–12, 1998. 12, 14

[22] Ajit Diwan, Soumitra Pal, and Abhiram Ranade. Component Coloring of Proper Interval and Split Graphs. *Submitted*. 10, 85

[23] K. Edwards and G. Farr. On Monochromatic Component Size for Improper Colourings. *Discrete Applied Mathematics*, 148(1):89–105, 2005. 13, 14

[24] H. ElGindy, H. Schroder, A. Spray, A.K. Somani, and H. Schmeck. RMB – A Reconfigurable Multiple Bus Network. In *Second International Symposium on High-Performance Computer Architecture*, pages 108–117. IEEE, 1996. 20

[25] Jing Fang, Wensheng He, and A.K. Somani. Optimal Light-trail Design in WDM Optical Networks. In *IEEE International Conference on Communications*, volume 3, pages 1699–1703, June 2004. 17, 18, 19

[26] A. Farrugia. Vertex-partitioning into Fixed Additive Induced-hereditary Properties is NP-hard. *Electronic Journal of Combinatorics*, 11(R46):1–9, 2004. 14

[27] Marietjie Frick. *Generalised Colourings of Graphs*. PhD thesis, Randse Afrikaanse Universiteit, 1986. 13, 14

[28] Marietjie Frick. A Survey of (m, k)-colorings. 55:45–58, 1993. 12

[29] R. Gandhi, B. Greening, Jr, S. Pemmaraju, and R. Raman. Sub-coloring and Hypo-coloring Interval Graphs. *Discrete Mathematics, Algorithms and Applications*, 2(03): 331–345, 2010. 13, 14

[30] M.R. Garey and D.S. Johnson. The Complexity of Near-optimal Graph Coloring. *Journal of the ACM (JACM)*, 23(1):43–49, 1976. 11

[31] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710447. 7

[32] M.R. Garey, D.S. Johnson, G.L. Miller, and C.H. Papadimitriou. The Complexity of Coloring Circular Arcs and Chords. *SIAM Journal on Algebraic and Discrete Methods*, 1:216, 1980. 12

[33] M. Geréb-Graus and T. Tsantilas. Efficient Optical Communication in Parallel Computers. In *Fourth annual ACM symposium on Parallel algorithms and architectures*, pages 41–48. ACM, 1992. ISBN 089791483X. 20

[34] P. Gokhale, R. Kumar, T. Das, and A. Gumaste. Cloud Computing over Metropolitan Area WDM networks: The Light-trails Approach. In *IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–6. IEEE, 2010. 18

[35] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*. North-Holland Publishing Co., Amsterdam, The Netherlands, 2004. ISBN 0444515305. 26

[36] A. Gumaste and I. Chlamtac. Mesh Implementation of Light-trails: A Solution to IP Centric Communication. *Twelfth International Conference on Computer Communications and Networks (ICCCN)*, pages 178–183, 2003. 17

[37] A. Gumaste and I. Chlamtac. Light-trails: An Optical Solution for IP Transport. *Journal of Optical Networking*, 3(5):261–281, 2004. 17, 73

[38] A. Gumaste and P. Palacharla. Heuristic and Optimal Techniques for Light-trail Assignment in Optical Ring WDM Networks. *Computer Communications*, 30(5):990–998, 2007. 17, 18, 19, 73

[39] A. Gumaste and S.Q. Zheng. Dual Auction (and Recourse) Opportunistic Protocol for Light-trail Network Design. In *IFIP International Conference on Wireless and Optical Communications Networks*, page 6, 2006. 19

[40] A. Gumaste, G. Kuper, and I. Chlamtac. Optimizing Light-trail Assignment to WDM Networks for Dynamic IP Centric Traffic. In *Thirteenth Workshop on Local and Metropolitan Area Networks (LANMAN)*, pages 113–118. IEEE, 2004. 17

[41] A. Gumaste, J. Wang, A. Karandikar, and N. Ghani. Multihop Light-trails (MLT): A Solution to Extended Metro Networks. In *International Conference on Communications (ICC)*, pages 1–6. IEEE, 2009. 17

[42] A. Gumaste, T. Das, A. Mathew, and A. Somani. An Autonomic Virtual Topology Design and Two-stage Scheduling Algorithm for Light-trail WDM Networks. *Journal of Optical Communications and Networking*, 3(4):372–389, 2011. 19

[43] M.M. Halldórsson. A Still Better Performance Guarantee for Approximate Graph Coloring. *Information Processing Letters*, 45(1):19–23, 1993. 11

[44] E. Hao, P.D. MacKenzie, and Q.F. Stout. Selection on the Reconfigurable Mesh. In *Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 38–45. IEEE, 1992. ISBN 0818627727. 20

[45] P. Haxell, T. Szabó, and G. Tardos. Bounded Size Components– Partitions and Transversals. *Journal of Combinatorial Theory, Series B*, 88(2):281–297, 2003. 13, 14

[46] J.W. Jang and V.K. Prasanna. An Optimal Sorting Algorithm on Reconfigurable Mesh. *Journal of Parallel and Distributed Computing*, 25(1):31–41, 1995. ISSN 0743-7315. 20

[47] T.R. Jensen and B. Toft. Graph Coloring Problems. *Wiley-Interscience Series in Discrete Mathematics and Optimization*, 1995. 11

[48] G. Johns and F. Saba. On the Path-chromatic Number of a Graph. *Annals of the New York Academy of Sciences*, 576(1):275–280, 1989. 13, 14

[49] R.J. Kang. *Improper Colourings of Graphs*. PhD thesis, University of Oxford, 2007. 13, 14

[50] R.M. Karp. Reducibility Among Combinatorial Problems. In JW Thatcher RE Miller, editor, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. 11

[51] K. Kawarabayashi and B. Mohar. A Relaxed Hadwiger's Conjecture for List Colorings. *Journal of Combinatorial Theory, Series B*, 97(4):647–651, 2007. 14

[52] J. Kleinberg, R. Motwani, P. Raghavan, and S. Venkatasubramanian. Storage Management for Evolving Databases. In *Proceedings of 38th Annual Symposium on Foundations of Computer Science*, pages 353 –362. IEEE, October 1997. doi: 10.1109/SFCS.1997.646124. 7, 8, 13, 14, 15, 16

[53] M. Kubale. *Graph Colorings*, volume 352. American Mathematical Society, 2004. 11

[54] H. Li and M. Maresca. Polymorphic-Torus Network. *IEEE Transactions on Computers*, 38(9):1345–1351, 1989. ISSN 0018-9340. 21

[55] K. Li, Y. Pan, and S.Q. Zheng. Parallel Matrix Computations using a Reconfigurable Pipelined Optical Bus. *Journal of Parallel and Distributed Computing*, 59(1):13–30, 1999. 21

[56] K. Li, Y. Pan, and S.Q. Zheng. Efficient Deterministic and Probabilistic Simulations of PRAMs on Linear Arrays with Reconfigurable Pipelined Bus Systems. *The Journal of Supercomputing*, 15(2):163–181, 2000. ISSN 0920-8542. 21

[57] R. Lin and S. Olariu. Reconfigurable Buses with Shift Switching: Concepts and Applications. *IEEE Transactions on Parallel and Distributed Systems*, 6(1):93–102, 1995. ISSN 1045-9219. 20

[58] N. Linial and M. Saks. Low Diameter Graph Decompositions. *Combinatorica*, 13(4): 441–454, 1993. 12, 14

[59] N. Linial, J.Ř.Í. Matoušek, O. Sheffet, and G. Tardos. Graph Colouring with no Large Monochromatic Components. *Combinatorics, Probability and Computing*, 17(04):577–589, 2008. 13, 14

[60] A. Lodha, A. Gumaste, P. Bafna, and N. Ghani. Stochastic Optimization of Light-trail WDM Ring Networks using Bender's Decomposition. In *Workshop on High Performance Switching and Routing (HPSR)*, pages 1–7, 2007. 19

[61] P.J. Looges and S. Olariu. Optimal Greedy Algorithms for Indifference Graphs. *Computers & Mathematics with Applications*, 25(7):15–25, 1993. 28

[62] C. Lund and M. Yannakakis. On the Hardness of Approximating Minimization Problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994. 12

[63] X. Luo and B. Wang. Integrated Scheduling of Grid Applications in WDM Optical Light-trail Networks. *Journal of Lightwave Technology*, 27(12):1785–1795, 2009. 18

[64] M. Maresca. Polymorphic Processor Arrays. *IEEE Transactions on Parallel and Distributed Systems*, 4(5):490–506, 1993. ISSN 1045-9219. 20

[65] J.Í. Matoušek and A. Prívetivỳ. Large Monochromatic Components in Two-colored Grids. *SIAM Journal on Discrete Mathematics*, 22(1):295–311, 2008. 13, 14

[66] R. Miller, V.K. Prasanna, D.I. Reisis, and Q.F. Stout. Parallel Computations on Reconfigurable Meshes. *IEEE Transactions on Computers*, 42(6):678–692, 1993. ISSN 0018-9340. 21

[67] K. Nakano. A Bibliography of Published Papers on Dynamically Reconfigurable Architectures. *Parallel Processing Letters*, 5(1):111–124, 1995. 20

[68] K. Nakano. Prefix-sums Algorithms on Reconfigurable Meshes. *Parallel processing letters*, 5(1):23–35, 1995. ISSN 0129-6264. 21

[69] G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience New York, NY, USA, 1988. 32

[70] I. Nieuwoudt. *On the Maximum Degree Chromatic Number of a Graph*. PhD thesis, University of Stellenbosch, 2007. 13, 14

[71] I. Nieuwoudt and J.H. van Vuuren. Algorithms for a Shared Resource Scheduling Problem in which Some Level of Conflict is Tolerable. *Journal of Scheduling*, pages 1–22, 2012. 13, 14

[72] S. Olariu. An Optimal Greedy Heuristic to Color Interval Graphs. *Information Processing Letters*, 37(1):21–25, 1991. 27, 28, 54

[73] Soumitra Pal and Abhiram Ranade. Scheduling Light-trails on WDM Rings. In *Proceedings of the 17th International Conference on Advanced Computing and Communications (ADCOM)*, pages 227–234. Advanced Computing and Communications Society, December 2009. 10, 85

[74] Soumitra Pal and Abhiram Ranade. Scheduling Light-trails on WDM Rings. *Journal of Parallel and Distributed Computing*, 72(10):1226 – 1236, 2012. ISSN 0743-7315. doi: 10.1016/j.jpdc.2012.05.010. 10, 85

[75] Y. Pan, M. Hamdi, and K. Li. Efficient and Scalable Quicksort on a Linear Array with a Reconfigurable Pipelined Bus System. *Future Generation Computer Systems*, 13(6): 501–513, 1998. 21

[76] Yi Pan and Keqin Li. Linear Array with a Reconfigurable Pipelined Bus System – Concepts and Applications. *Information Science*, 106(3-4):237–258, 1998. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/S0020-0255(97)10013-5. 20

[77] P.M. Pardalos, T. Mavridou, and J. Xue. The Graph Coloring Problem: A Bibliographic Survey. In Ding-Zhu Du and M. Pardalos, editors, *Handbook of combinatorial optimization*, volume 2, pages 331–395. Kluwer Academic Publishers, 1998. 11

[78] S. Pavel and S.G. Akl. Matrix Operations using Arrays with Reconfigurable Optical Buses. *International Journal of Parallel, Emergent and Distributed Systems*, 8(3):223–242, 1996. ISSN 1744-5760. 20

[79] S. Rajasekaran. Mesh Connected Computers with Fixed and Reconfigurable Buses: Packet Routing, Sorting, and Selection. In *First Annual European Symposium on Algorithms*, pages 309–320. Springer-Verlag, 1993. ISBN 3540572732. 20

[80] S. Rajasekaran and S. Sahni. Sorting, Selection, and Routing on the Array with Reconfigurable Optical Buses. *IEEE Transactions on Parallel and Distributed Systems*, 8(11): 1123–1132, 1997. 21

[81] A. Raspaud and W. Wang. On the Vertex Arboricity of Planar Graphs. *European Journal of Combinatorics*, 29(4):1064–1075, 2008. 13, 14

[82] J. Rothstein. Bus Automata, Brains, and Mental Models. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(4):522–531, 1988. ISSN 0018-9472. 20

[83] A. Roychoudhury and S. Sur-Kolay. Efficient Algorithms for Vertex Arboricity of Planar Graphs. In *Foundations of Software Technology and Theoretical Computer Science*, pages 37–51. Springer, 1995. 13, 14

[84] S. Sahni. Data Manipulation on the Distributed Memory Bus Computer. *Parallel Processing Letters*, 5(1):3–14, 1995. ISSN 0129-6264. 20

[85] E. Sampathkumar. Generalizations of Independence and Chromatic Numbers of a Graph. *Discrete mathematics*, 115(1):245–251, 1993. 13, 14

[86] E. Sampathkumar and G.D. Kamath. k-Size Chromatic Number of a Graph. *The Indian Journal of Statistics*, pages 393–397, 1992. 12, 14

[87] L. Snyder. Introduction to the Configurable, Highly Parallel Computer. *Computer*, 15 (1):47–64, 1982. ISSN 0018-9162. 20

[88] L. Stockmeyer. Planar 3-colorability is Polynomial Complete. *ACM Sigact News*, 5(3): 19–25, 1973. 12

[89] C.P. Subbaraman, J.L. Trahan, and R. Vaidyanathan. List Ranking and Graph Algorithms on the Reconfigurable Multiple Bus Machine. In *International Conference on Parallel Processing (ICPP)*, volume 3, 1993. 19, 21

[90] J.L. Trahan, R. Vaidyanathan, and R.K. Thiruchelvan. On the Power of Segmenting and Fusing Buses. *Journal of Parallel and Distributed Computing*, 34(1):82–94, 1996. ISSN 0743-7315. 20

[91] A. Tucker. Coloring a Family of Circular Arcs. *SIAM Journal on Applied Mathematics*, 29(3):493–502, 1975. 12

[92] B.F. Wang and G.H. Chen. Two-dimensional Processor Array with a Reconfigurable Bus System is at least as Powerful as CRCW Model. *Information Processing Letters*, 36(1): 31–36, 1990. ISSN 0020-0190. 21

[93] Y.R. Wang. An Efficient $O(1)$ Time 3D All Nearest Neighbor Algorithm from Image Processing Perspective. *Journal of Parallel and Distributed Computing*, 67(10):1082–1091, 2007. 21

[94] R. Wankar and R. Akerkar. Reconfigurable Architectures and Algorithms: A Research Survey. *International Journal of Computer Science & Applications*, 6(1):108–123, 2009. 20

[95] C.C. Weems, S.P. Levitan, A.R. Hanson, E.M. Riseman, D.B. Shu, and J.G. Nash. The Image Understanding Architecture. *International Journal of Computer Vision*, 2(3):251–282, 1989. ISSN 0920-5691. 20

[96] Douglas B. West. *Introduction to Graph Theory (2nd Edition)*. Prentice Hall, August 2000. 65

[97] D. Woodall. Improper Colourings of Graphs. In P. Degano and E. Sandewall, editors, *Graph Colourings*, volume 218, pages 45–63. Longman Scientific and Technical, 1990. 12, 13, 14

[98] B. Wu and K.L. Yeung. OPN03-5: Light-trail Assignment in WDM Optical Networks. In *IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–5, 2006. 17, 18, 19

[99] Y. Ye, H. Woesner, R. Grasso, T. Chen, and I. Chlamtac. Traffic Grooming in Light-trail Networks. In *IEEE Global Telecommunications Conference (GLOBECOM)*, 2005. 17

[100] W. Zhang, G. Xue, J. Tang, and K. Thulasiraman. Dynamic Light-trail Routing and Protection Issues in WDM Optical Networks. In *IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1963–1967, 2005. 19

Publications from this Thesis

[73] Soumitra Pal and Abhiram Ranade. Scheduling Light-trails on WDM Rings. In *Proceedings of the 17th International Conference on Advanced Computing and Communications (ADCOM)*, pages 227–234. Advanced Computing and Communications Society, December 2009

[74] Soumitra Pal and Abhiram Ranade. Scheduling Light-trails on WDM Rings. *Journal of Parallel and Distributed Computing*, 72(10):1226–1236, 2012. ISSN 0743-7315.

[22] Ajit Diwan, Soumitra Pal, and Abhiram Ranade. Component Coloring of Proper Interval and Split Graphs. *Submitted.*

Acknowledgments

I would like to express my sincere gratitude to my guide, Prof. Abhiram Ranade for his consistent motivation, guidance and support throughout this work. Without his generous and prompt help this work would not have been done.

I would like to thank my Research Progress Committee, Prof. Ajit Diwan for introducing the component colouring problem and providing me the ideas used in the thesis for solving the same and Prof. Ashwin Gumaste for encouragement, insightful discussions and patient clearing of our doubts related to light-trails. I am also thankful to other members of my RPC, Prof. Ganjendra K. Adil and Prof. Vishnu Narayan, who reviewed my research progress on a regular basis and gave valuable comments and encouragement.

I also want to thank the reviewers of my thesis, Prof. Geppino Pucci and particularly Prof. Sandeep Sen, who reviewed this thesis thoroughly and gave thoughtful comments on improving the writing.

I am ever grateful to the institute for supporting me by scholarship. Many thanks to the helpful staff at CSE office, specially Mrs. Athvankar and Vijay Ambre.

My stay in the institute was enriched due to many friends and seniors. I want to thank them all, especially Vishal Sevani, Jagadish M, Jinesh Machchhar, Abhisekh Shankaran, Ruta Mehta, Jugal Garg, Zahir Koradia, Ayush Choure, Uma Sawant, Prasanna K, Karthik Ramachandra, Srinivas Karthik, Sreyash Kenkre, Sandeep Deshmukh, Sobhan Babu and so on.

Many friends and well wishers from Shri Ram chandra Mission have encouraged me throughout, more importantly Dr. Sadhasivam K. I want to thank them whole heartedly.

I am also grateful to my parents, sisters, brother and other family members for being with me and providing me all that I need in my life. And above all, special thanks to Sunayna, my wife who came in my life around the end of my PhD and inspired me to finish it quickly.

Date: 6 August 2013 Soumitra Kumar Pal