ALGORITHMS FOR WAVELENGTH ASSIGNMENT AND CALL CONTROL IN OPTICAL NETWORKS

by

Zhengbing Bian
B.Eng., Huazhong University of Science and Technology, 1996
M.A.Sc., The University of British Columbia, 2001
M.Sc., Simon Fraser University, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
Doctor of Philosophy
in the School
of
Computing Science

© Zhengbing Bian 2008
SIMON FRASER UNIVERSITY
Summer 2008

All rights reserved. This work may not be reproduced in whole or in part, by photocopy or other means, without the permission of the author.
APPROVAL

Name: Zhengbing Bian

Degree: Doctor of Philosophy

Title of thesis: Algorithms for Wavelength Assignment and Call Control in Optical Networks

Examining Committee: Dr. Arthur E. Kirkpatrick
Chair

____________________________________________
Dr. Qian-ping Gu, Senior Supervisor

____________________________________________
Dr. Joseph G. Peters, Supervisor

____________________________________________
Dr. Jiangchuan Liu, SFU Examiner

____________________________________________
Dr. Kshirasagar Naik, External Examiner
Associate Professor, Department of Electrical and Computer Engineering
University of Waterloo

Date Approved: August 5, 2008
Declaration of Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the "Institutional Repository" link of the SFU Library website <www.lib.sfu.ca> at: <http://ir.lib.sfu.ca/handle/1892/112>) and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author’s written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, BC, Canada
Abstract

Routing and channel assignment is a fundamental problem in computer/communication networks. In wavelength division multiplexing (WDM) optical networks, the problem is called routing and wavelength assignment or routing and path coloring (RPC) problem: given a set of connection requests, find a routing path to connect each request and assign each path a wavelength channel (often called a color) subject to certain constraints. One constraint is the distinct channel assignment: the colors (channels) of the paths in the same optical fiber must be distinct. Another common constraint is the channel continuity: a path is assigned a single color. When a path may be assigned different colors on different fibers, the RPC problem is known as the routing and call control (RCC) problem. When the routing paths are given as part of the problem input, the RPC and RCC problems are called the path coloring and call control problems, respectively. Major optimization goals for the above problems include to minimize the number of colors for realizing a given set of requests and to maximize the number of accommodated requests using a given number of colors. Those optimization problems are NP-hard for most network topologies, even for simple networks like rings and trees of depth one. In this thesis, we make the following contributions. (1) We give better approximation algorithms which use at most $3L$ ($L$ is the maximum number of paths in a fiber) colors for the minimum path coloring problem in trees of rings. The $3L$ upper bound is tight since there are instances requiring $3L$ colors. We also give better approximation algorithms for the maximum RPC problem in rings. (2) We develop better algorithms for the minimum and maximum RPC problems on multi-fiber networks. (3) We develop better algorithms for the call control problem on simple topologies. (4) We develop carving-decomposition based exact algorithms for the maximum edge-disjoint paths problem in general topologies. We develop and implement tools for computing optimal branch/carving decompositions of planar graphs to provide a base for the branch/carving-decomposition based algorithms. These tools are of independent interests.
Keywords: communication networks; path coloring; call control; approximation algorithms; trees of rings; planar branch-decomposition; computational study
To my family
I would like to express my gratitude to my senior supervisor Dr. Qian-ping Gu. Without his support and guidance, this thesis would not have been possible. I would like to thank Dr. Xiao Zhou (Tohuku University, Japan) for co-authoring several papers and for the many discussions on the call control problems. I would like to thank Dr. Joseph G. Peters for being a supervisor during both my Master and Ph.D. studies. I would like to thank Dr. Jiangchuan Liu and Dr. Kshirasagar Naik for being my thesis examiners. My gratitude also goes to Dr. Ramesh Krishnamurti for being the chair of my depth examination and Dr. Arthur E. Kirkpatrick for being the chair of my thesis defense.

I am especially grateful to my parents and sisters for their love, support and encouragement over the years. I am equally grateful to my wife Zheqiong Wang for her understanding and support during my study, and for taking care of our son Steven who has brought so much joyfulness to our family during the past four years.
# Contents

Approval ii

Abstract iii

Dedication v

Acknowledgments vi

Contents vii

List of Tables x

List of Figures xi

1 Introduction 1

1.1 Routing and Wavelength Assignment in WDM Optical Networks 2

1.2 Contributions of the Thesis 4

1.2.1 Contributions on Special Topologies 4

1.2.2 Contributions on General Topologies 6

1.3 Thesis Outline 8

2 Preliminaries and Related Work 10

2.1 Preliminaries 10

2.2 Previous Work 15

2.2.1 The Minimum Path Coloring Problem 15

2.2.2 The Maximum Path Coloring Problem 25

2.2.3 The Path Multicoloring Problem 29
3 Path Coloring on Trees of Rings
3.1 The 3L Upper Bound ........................................... 39
3.2 Preparation for Improvement ................................... 44
  3.2.1 Efficient Path Coloring Schemes ......................... 44
  3.2.2 Edge-coloring of Multigraphs ............................. 47
3.3 A 2.75-approximation Algorithm .............................. 49
3.4 Algorithms for Bounded Degrees .............................. 57
  3.4.1 Algorithm for Degree Six ................................ 57
  3.4.2 Algorithm for Degree Eight ............................... 58
  3.4.3 Algorithm for Degree Ten ................................. 64
3.5 Summary ........................................................ 68

4 Call Control and Maximum Path Coloring ....................... 69
4.1 Call Control in Bounded Depth Trees ......................... 69
  4.1.1 Hardness of Call Control in Depth-2 Trees ............... 70
  4.1.2 Call Control in Double-stars ............................ 72
  4.1.3 Weighted Call Control in Depth-2 and Depth-3 Trees ..... 76
  4.1.4 Call Control in Spiders .................................. 78
  4.1.5 Call Control in Trees with Central Paths ............... 80
  4.1.6 Call Control with Length-2 Paths ....................... 83
  4.1.7 Remarks .................................................. 84
4.2 Path Multicoloring in Multifiber Star and Spider Networks .... 85
  4.2.1 Hardness of the PMC Problem in Stars .................... 85
  4.2.2 Max-PMC Problems in Stars and Spiders .................. 92
4.3 The Weighted Max-RPC Problem on Rings ....................... 94
4.4 Summary ........................................................ 100

5 Branch/Carving Decomposition Based Algorithms ................ 102
5.1 Previous Work ................................................ 104
5.2 Optimal Branch Decomposition of Planar Graphs ................ 105
5.3 Empirical Study on Branchwidth of Planar Graphs ............... 107
## List of Tables

5.1 Computation time for Class (1) instances ........................................... 121
5.2 Memory usage for Class (1) instances ................................................ 122
5.3 Computation time and memory of Class (2) instances ................................. 125
5.4 Computation time and memory of Class (3) instances ................................. 126
5.5 Computation time of several implementations for $k$ close to $2bw$ ............... 128
5.6 Computation time of $rat$, $comrat$, and $memrat$ quoted from [70] ........ .... 128
5.7 Computation time of decomposition algorithms for Class (1) instances ........... 138
5.8 Computation time of decomposition algorithms for Class (2) instances ........... 140
5.9 Computation time of decomposition algorithms for Class (3) instances ........... 141

6.1 Computation results for a 16-node NSFNET backbone ................................ 150
6.2 Computation results for a 20-node European Optical Network ....................... 152
6.3 Computation results for a 24-node ARPANET-like network .......................... 153
6.4 Computation results for a 30-node UK Network ...................................... 153
6.5 Computation results for instances generated by LEDA. ............................. 154
6.6 Computation results for instances generated by PIGALE. ............................ 155
List of Figures

2.1 Some well-used topologies. .................................................. 12
2.2 Lower bounds for path coloring in trees .................................. 17
2.3 Lower bounds for path coloring in undirected rings ...................... 20
2.4 A $3L$ lower bound on undirected trees of rings. ......................... 23
2.5 A single fiber undirected star and a 2-fiber undirected star. .......... 30

3.1 Illustration of some terms defined on a tree of rings $TR$. ............... 39
3.2 A framework of algorithms for the Min-PC problem on trees of rings. .... 40
3.3 The sets of paths related to Schemes S31 and S32. ....................... 45
3.4 A multigraph $G_u$ and its contracted graph $F_u$. ....................... 49
3.5 Paths in and incident to subgraphs $H$ with $L(H) > [2.5L]$. ............... 52
3.6 Paths in and incident to the subgraph $H$ with four vertices and $L(H) > 2L$. 60
3.7 Paths in and incident to the subgraph $H$ with five vertices and $L(H) > 2L$. 65

4.1 (a) A double-star, and (b) the double-star after pre-processing. ........ 71
4.2 The depth-2 tree for the NP-hardness proof of the call control problem. 71
4.3 The call control problem in double-stars .................................. 75
4.4 A spider network. .............................................................. 78
4.5 Stars $G_1$ and $G_w$. ....................................................... 88
4.6 Parallel (compatible) and crossing (incompatible) requests. ........... 95
4.7 The algorithm for the weighted Max-RPC .................................. 96
4.8 Different sets of paths in OPT. ........................................... 96
4.9 Illustration of the idea behind the proof of Lemma 4.3.2. .................. 97
4.10 The procedure for identifying two edges on the ring. ...................... 99

5.1 Partition graph $H$ into subgraphs by a cycle in $H^*$. .................. 133
6.1 A 16-node NSFNET backbone. ........................................ 150
6.2 A 20-node European Optical Network. .......................... 151
6.3 A 24-node ARPANET-like network. ............................... 151
6.4 A 30-node UK Network. ............................................ 152
Chapter 1

Introduction

In recent years, Internet traffic has increased enormously due to the various bandwidth-intensive applications, such as video-conferencing, video-on-demand [106] and peer-to-peer (P2P) applications [116]. Optical networks, which employ optical fibers as the carrier, can provide the huge bandwidth needed. The bandwidth of optical fibers is about 50THz, which is much higher than that of conventional carriers such as copper wires. The huge bandwidth of optical fibers is usually utilized through multiplexing. Wavelength Division Multiplexing (WDM) is a multiplexing technology widely used in optical networks. It allows multiple channels to be carried on the same fiber by assigning a different wavelength to each channel.

A network is called all-optical (or single-hop) network if optical signals remain in optical form (without conversion to electrical form) from source to destination. A fundamental problem in all-optical WDM networks (and in circuit-switched networks in general) is that given a set of connection requests (source-destination pairs) in a network, find a path for each request (routing) and assign each path a channel such that the paths with the same channel do not share any communication link in the network (channel assignment). The problem is also known as the off-line routing and channel assignment problem. In this thesis, we study the off-line routing and channel assignment problem and related problems in optical networks.
1.1 Routing and Wavelength Assignment in WDM Optical Networks

In all-optical WDM networks, each channel is supported by a wavelength or a color and the routing and channel assignment is known as routing and wavelength assignment or routing and path coloring (RPC). We will use the two terms interchangeably. The wavelength assignment sub-problem has two basic constraints. The first constraint, called distinctive channel assignment constraint, requires that the paths must be assigned different colors if they are on the same fiber. The second constraint, called the channel continuity constraint, requires that each path must be assigned the same color on every link from source to destination. There are two natural optimization problems. One is to minimize the total number of colors for accepting and coloring all the given routing requests in a network. This problem is called the minimum routing and path coloring (Min-RPC) problem. The other is to select a maximum subset of requests, route and color the selected requests with a given number of colors. The problem is called the maximum routing and path coloring (Max-RPC) problem. Algorithms for the Min-RPC problem are important to reduce the resource required for realizing a set of requests, and they are widely used in network planning and design stages. Algorithms for the Max-RPC problem are critical for improving the performances of a network when the resources in the network are not enough to support all connection requests, and they are widely used in network design and operating stages. In the Max-RPC problem, each request may be assigned a weight (e.g., the profit obtained if the request is accepted). The goal is to maximize the total weight of accepted requests. This generalized problem is called the maximum weight routing and path coloring problem (weighted Max-RPC). In all the problems defined above, routing paths may be given as part of the input (i.e., a set of pre-specified paths is given instead of a set of requests), and only the path coloring sub-problem needs to be solved (we will use “PC” instead of “RPC” in the abbreviations). All these problems are NP-hard for general networks [92, 105].

The routing and path coloring problems are first studied in single fiber optical networks, i.e., each link has a single fiber (thus paths with the same color must be edge-disjoint). Recently, there are renewed interests in the multifiber optical networks. In these networks, each link $e$ has $\mu(e)$ parallel fibers, and at most $\mu(e)$ paths can use the same color on link $e$. Thus, in multifiber optical networks, more than one path may use the same color on a link. All the problems defined above can be studied in the multifiber environment. The path
The routing and call control problem is closely related to the routing and path coloring problem, and can be defined as follows: Given a set $R$ of requests in a graph $G = (V, E)$ with every edge $e \in E(G)$ assigned a non-negative capacity $c(e)$, find a maximum subset of $R$ such that each request in the subset is assigned a path, and the number of paths on any edge $e$ is at most $c(e)$. Each request may be given a positive weight, and the goal is to accept a subset of requests with maximum total weight. When routing paths are given as part of the input, the problem is simply called the call control problem. The call control problem can be considered as a variant of the path coloring problem in which a path may use different colors on different links (the channel continuity constraint is relaxed) and the number of available colors on different links may be different. When $c(e) = 1$ for every edge $e \in E(G)$, the routing and call control problem is called the maximum edge-disjoint paths (MEDP) problem. The Max-RPC problem with one available color is the same as the MEDP problem. The maximum edge-disjoint paths with pre-specified paths (MEDPwPP) problem is the same as the call control problem with $c(e) = 1$ for every edge $e \in E(G)$, or the Max-PC problem with one available color.

The call control problem is also closely related to the maximum path multicoloring (Max-PMC) problem in multifiber optical networks. The Max-PMC problem asks to maximize the number of paths that can be colored by a given number $w$ of colors in a multifiber optical network. When $w = 1$, this is simply a maximum edge-disjoint paths problem in a single fiber network, and is a call control problem (with edge capacity $c(e) = \mu(e)$ on edge $e$) in a multifiber network. Solving the call control problem is an important step towards solving the Max-PMC problem in the iterative greedy approach.

The routing and path coloring problem and the related problems have been extensively studied in general topologies and in various special topologies, including the tree, a connected graph without cycles; the ring, a cycle with at least three nodes; and the tree of rings, a set of rings connected through a tree structure. The ring and tree of rings are popular topologies in optical networks. For example, a 140-node metropolitan area network of 12 rings with a conceptual tree of rings topology has been deployed in Jacksonville, Florida, based on the Optical Metro 3500 Multiservice Platform of Nortel (see [2, 3] for the conceptual topology). In this thesis, we focus on the path coloring and call control problems. We study the Min-PC problem on trees of rings, the Max-RPC problem on rings, the path multicoloring and call control problems on trees with special properties. We also study the MEDP and Max-PC
problems on more general topologies. We give details of our studies in the following section.

1.2 Contributions of the Thesis

In this section, we give an outline of the problems that we solved in this thesis. For each problem considered, we first give the previously known best result, and then show our contributions. In what follows, a network is a single fiber network (i.e., each link has one fiber) unless it is explicitly referred to as a multifiber network. An undirected network is expressed by an undirected graph, and a directed network is expressed by a directed graph. Optical networks are directed because the optical amplifier, a key element to boost the optical signals in a fiber, is directed. However, the undirected network model is an abstract theoretical model, and has been used in many previous studies. In the following discussion, graphs are undirected unless explicitly stated as directed when we give a specific result on the problems we are interested in.

1.2.1 Contributions on Special Topologies

We first give the results we obtained on special topologies. Given a set of paths in a network, the maximum number \( L \) of paths on any link of the network is a lower bound on the number of colors required for coloring the set of paths. For the Min-PC problem, the number of colors required is often evaluated in the unit of \( L \).

**The path coloring problem on trees of rings**

A tree of rings is an important topology for optical networks, with several subrings connected to a main ring, sub-subrings to subrings, and so on, to form a larger network. An algorithm using at most \( 4L \) colors is known for the Min-PC problem on the undirected trees of rings with arbitrary node degrees [47]. The algorithm has an approximation ratio of 4. A 2-approximation algorithm is known for a restricted class of trees of rings with maximum node degree four [44]. A \( 3L \) and 2-approximation (resp. 2.5-approximation) algorithm is given for a restricted class of trees of rings with maximum node degree four (resp. six) in [25]. It is also shown in [25] that at least \( 3L \) colors are needed for some instances in the trees of rings with degree four. It has been an interesting open problem whether \( 3L \) is the upper bound for the Min-PC problem on the entire class of trees of rings. We settle this
problem by giving a 3L algorithm for trees of rings with arbitrary node degrees. We further give a 2.75-approximation algorithm which improves the previous approximation ratio of 4. For the restricted class of trees of rings with node degree six (resp. eight and ten), we give a 3L and 2-approximation (resp. 2.5-approximation) algorithm which improves the previous results. Our algorithms are based on novel applications of edge-coloring of multigraphs and efficient local greedy path coloring schemes.

Our algorithms work for directed trees of rings as well, with an upper bound of 6L (this improves the 8L upper bound of [47]).

The Min-RPC problem in trees of rings has been studied in [92, 105]. Using the cut-one-link approach, they obtained 3-approximation and 10/3-approximation algorithms for the Min-RPC problem in undirected and directed trees of rings, respectively. For undirected trees of rings with arbitrary degree, our algorithms imply a 3-approximation algorithm without using the cut-one-link approach.

The maximum routing and path coloring problem on rings

The Max-PC, Max-RPC, weighted Max-PC and weighted Max-RPC problems in rings can all be approximated with a ratio of 1.58 using the iterative greedy method. Various algorithms aim to improve this ratio. In particular, the Max-PC, Max-RPC and weighted Max-PC problems in rings can all be approximated with ratios better than 1.5 [33]. We show that the weighted Max-RPC problem in rings can be approximated with a ratio of 1.5, improving the previous 1.58-approximation result. Our approach is to use a combination of the cut-one-link method and the maximum matching method introduced in [98] for approximating the (unweighted) Max-RPC problem in rings.

The path multicoloring problem in generalized stars

A multifiber optical network has more than one fiber per link. If every link has the same number k of fibers, the network is called a k-fiber network. Otherwise, the network is called a non-uniform network. For every even \( k > 1 \), the path multicoloring problem is known solvable in polynomial time in a k-fiber undirected star (also known as the depth-1 tree), a tree in which one node has degree greater than one and all other nodes have degree exactly one [87, 88]. This should be contrasted to the single fiber case \( (k = 1) \), which is NP-hard. The complexity of the problem for arbitrary odd \( k \) \( (k \geq 3) \) was not known. We show that
for every odd \( k \geq 3 \), the Min-PMC and Max-PMC problems in a \( k \)-fiber star are NP-hard. The NP-hardness results also hold for a spider (also known as the generalized star), a tree in which one node has degree more than two and all other nodes have degree one or two. We give efficient algorithms for the Min-PMC problem in non-uniform stars with even number of fibers in every link and \( k \)-fiber (\( k \) even) spiders. We also give a \((1 + \frac{1}{k-1})\)-approximation algorithm for the Min-PMC problem in \( k \)-fiber spiders for every odd \( k \geq 3 \).

We have some algorithmic results for the Max-PMC problem. We give an optimal algorithm for the Max-PMC problem in non-uniform stars with even fibers in every link. We also give an optimal algorithm and a 1.58-approximation algorithm for the Max-PMC problem in \( k \)-fiber (\( k \) even) spiders and non-uniform spiders, respectively. The algorithms for spiders rely on an optimal algorithm for the call control problem.

### The call control problem in trees

The call control problem is solvable in undirected stars (depth-1 trees) with arbitrary capacities, but is NP-hard and Max SNP-hard in depth-3 trees even if the capacities are either one or two [63]. An interesting question is, what is the boundary to separate the class of trees for which the call control problem is in P from those for which the problem is NP-hard.

For this question, we have obtained the following results. We show that the call control problem is NP-hard and MAX SNP-hard even in depth-2 trees with capacities 1 or 2. We give a polynomial time algorithm for the call control problem in double-stars which are special depth-2 trees. These results suggest that the boundary is in depth-2 trees, and the call control problem is in P or NP-hard, depending on the node degrees of the trees. We also give 2-approximation and 3-approximation algorithms for the weighted call control problem on depth-2 and depth-3 trees, respectively. This improves the previous 4-approximation algorithm for the problem on those trees. We show that the call control problem in spiders can be solved optimally. All of our algorithms depend on a subroutine which optimally solves the restricted weighted call control problem on arbitrary trees in which all paths contain a same node of the tree.

### 1.2.2 Contributions on General Topologies

Results given in Section 1.2.1 are obtained on special topologies. In this section, we first briefly review the maximum edge-disjoint paths problem, then we show an approach for
solving the problem in more general topologies.

**The maximum edge disjoint paths problem**

Given a set of \( k \) source-destination pairs in a graph \( G = (V, E) \), the maximum edge-disjoint paths (MEDP) problem is to connect as many of these pairs as possible using edge disjoint paths. The MEDP problem has been known to be NP-hard in directed graphs with only two terminal pairs [58]. In directed graphs, the MEDP problem is NP-hard to approximate within a factor better than \( \Omega(|E|^{1/2}) \) unless \( P = NP [67] \), and can be approximated with ratio \( O(\min\{|V|^{2/3}, |E|^{1/2}\}) \) [40]. In undirected graphs, the MEDP problem is solvable for any fixed \( k \) [111], but is NP-hard for general value \( k \). The MEDP problem is hard to approximate within ratio \( \log^{3-\epsilon}|V| \) for any fixed \( \epsilon > 0 \) unless \( NP \subseteq ZPTIME(|V|^{\text{poly}(\log |V|)}) \) [9]. Polylogarithmic approximation algorithms are only known for some special topologies. Although these results are very important theoretically, the hidden constants behind are usually large and the practical performance of them is unknown but is likely far from optimal. We design exact algorithms for the MEDP problem in planar graphs. Our algorithms are based on carving decompositions of the planar graphs.

**Branch/Carving decomposition based algorithms**

Recently, there has been increasing interest in the tree/branch/carving-decomposition based method for solving optimization problems. A tree/branch/carving-decomposition of a graph is a way to partition the graph into pieces recursively. There are two major steps in a tree/branch/carving-decomposition based algorithm for solving a problem: (1) computing a tree/branch/carving-decomposition with a small width and (2) applying a dynamic programming algorithm based on the decomposition to solve the problem.

To develop a tree/branch/carving decomposition based algorithm, we need a tool to find a tree/branch/carving decomposition. Step (2) usually runs in time exponential in the width of the decomposition computed in Step (1). Thus, it is important to compute a decomposition with small width. Developing such a tool is non-trivial, since deciding the treewidth/branchwidth/carvingwidth of a general graph is NP-hard [113]. There are good news for planar graphs. Given a planar graph \( G \) with \( n \) vertices and an integer \( \beta \), Seymour and Thomas give a decision procedure which decides if \( G \) has a branchwidth/carvingwidth at least \( \beta \) in \( O(n^2) \) time [113].
CHAPTER 1. INTRODUCTION

We propose efficient implementations of the decision procedure of [113]. Our implementations run much faster and use less memory than a straightforward implementation reported in [70], and can compute the branchwidth of some planar graphs with one hundred thousand edges. We further develop divide-and-conquer algorithms for finding the optimal branch/carving decomposition of planar graphs. Our decomposition finding algorithms are much faster than those reported in previous studies [71], and can compute the optimal branch decomposition of some planar graphs with 50,000 edges. This provides the base for using the branch/carving decomposition based method to solve optimization problems.

Carving decomposition based algorithm for the MEDP problem

The MEDP problem in directed trees of bounded degree, and the MEDPwPP problem in bounded degree trees of rings are both optimally solvable. The optimal algorithms are based on dynamic programming as well. Note that bounded degree trees and bounded degree trees of rings both have bounded carvingwidth. Furthermore, trees and trees of rings are both planar. A nature question is, whether the MEDP and MEDPwPP problems in planar graphs with bounded carvingwidth are optimally solvable? We show that the maximum edge-disjoint paths problem (with pre-specified paths) can be solved optimally in planar graphs with bounded carvingwidths. We give a dynamic programming algorithm based on an optimal carving decomposition of the planar graphs. (For the MEDP and MEDPwPP problems, and other problems related to path subsets, it seems that the carving-decomposition is a better choice than the tree/branch-decomposition.) Our experimental results show that the algorithm can compute a set of maximum edge-disjoint paths with reasonable load on graphs of practical size. We also show that the maximum path coloring problem is solvable in graphs with small carvingwidth if the number of available colors is also small, using carving-decomposition based method.

1.3 Thesis Outline

The rest of the thesis is organized as follows. We give the preliminaries of the thesis and review some previous work related to path coloring and call control in Chapter 2.

In Chapter 3, we study the path coloring problem on trees of rings. We give an efficient algorithm which solves the Min-PC problem on a tree of rings with an arbitrary (node) degree using at most 3L colors and achieves an approximation ratio of 2.75 asymptotically,
where $L$ is the maximum number of paths on any link in the network. The $3L$ upper bound is tight since there are instances of the Min-PC problem that require $3L$ colors even on a tree of rings with degree four. We also give approximation algorithms for the Min-PC problem on trees of rings with bounded degrees.

In Chapter 4, we first study the call control problem in trees. We show that the call control problem is NP-hard and MAX SNP-hard even in depth-2 trees with capacities 1 or 2. We then give efficient algorithms for the call control problem in several classes of bounded depth trees, and the restricted call control problem on arbitrary trees in which all paths contain a same node of the tree. In the remaining part of Chapter 4, we study the related maximum path coloring problem. We mainly focus on multifiber optical networks. We show that for every odd integer $k \geq 3$, the Min-PMC and Max-PMC problems in $k$-fiber stars are NP-hard. We give efficient algorithms for the Min-PMC problem in multifiber stars and spiders with even number of fibers in every link. We also give several algorithms for the Max-PMC problems in multifiber stars and spiders. We show that the maximum weight routing and path coloring problem in rings can be approximated with a ratio of 1.5.

In Chapter 5, we develop efficient algorithms for computing optimal branch/carving decompositions of planar graphs. We first give several efficient implementations of Seymour and Thomas decision procedure which, given an integer $\beta$, decides whether an input graph has the branchwidth at least $\beta$ or not. Our implementations are faster and use less memory than previous implementations, and can compute the branchwidth of some instances with one hundred thousand edges. Using the decision procedure as a subroutine, we further give several divide-and-conquer algorithms for computing optimal branch decompositions of planar graphs. Our implementations of the divide-and-conquer algorithms are fast and can compute an optimal branch decomposition of some instances with 50,000 edges.

Exact algorithms for the maximum edge-disjoint paths problem in planar graphs are given in Chapter 6. We first show that the maximum edge-disjoint paths problem can be solved optimally in planar graphs with bounded carvingwidths. We give a dynamic programming algorithm based on an optimal carving decomposition of the planar graphs. Experimental results show that the algorithm can compute a set of maximum edge-disjoint paths with reasonable load on graphs with practical size.

We summarize the thesis and discuss directions for future work in the final chapter.
Chapter 2

Preliminaries and Related Work

In this chapter, we will introduce the preliminaries of the thesis and review some related work. In Section 2.1, we give the graph notation and the definitions for bipartite graph, matching and graph coloring. These concepts can be found in most graph theory books, such as [21] and [121]. We also give some well known results on edge-coloring of multigraphs, and on matching and its generalization. We will define several important parameters related to the routing and path coloring problem. We will also introduce several popular network topologies that are frequently used in WDM optical networks. In Section 2.2, we review the previous results for the routing and path coloring problem and its variants in single-fiber and multifiber optical networks, and for the routing and call control problem. The network topologies considered include chains, rings, trees, trees of rings, and meshes.

2.1 Preliminaries

We use standard graph notation which can be found in a graph theory book such as [121]. An undirected network is expressed by a simple undirected graph $G$ with node set $V(G)$ and edge set $E(G)$, where an edge $\{u, v\} \in E(G)$ expresses a link between $u$ and $v$. We use $n$ for $|V|$ and $m$ for $|E|$. We use $\delta_G(u)$ for the node degree of $u \in V(G)$, and $\Delta(G) = \max\{\delta_G(u) | u \in V(G)\}$ for the maximum node degree of $G$. A vertex coloring of a graph $G$ is an assignment of colors to the vertices of $G$ such that adjacent vertices are given different colors. The chromatic number of $G$ is the minimum number of colors required in a vertex coloring of $G$.

A circuit of a graph $G$ is a sequence of consecutive edges that begins and ends at the
same node with no repeated edge. A component of a graph $G$ is a maximal connected subgraph of $G$. An *Euler circuit* of a component of $G$ is a circuit which contains all edges of the component. A component of $G$ has an Euler circuit if every node of the component has even degree.

We use a *path* for a simple path in a graph (i.e. a sequence of consecutive edges with no repeated node). An undirected path between node $u$ and node $v$ is denoted by $u - v$. The *length* of a path is the number of edges in the path. We use length-$i$ path to denote a path with $i$ edges.

We define some well used topologies here. A *chain* of $n$ nodes is a path of $n - 1$ edges. A *tree* of $n$ nodes is a connected graph with $n - 1$ edges. A *star* is a tree with one node (called center) having degree greater than one and all other nodes having degree one. A *spider* is a graph obtained by replacing every edge in a star by a chain. These chains are called *legs* of the spider. Spiders are also called *generalized stars*. A *ring* is a cycle with at least three nodes. The ring is the simplest graph in which routing is an issue. A tree of rings can be defined recursively as follows: a single ring is a tree of rings; the graph obtained by connecting a new ring and an existing tree of rings at exactly one node is also a tree of rings. See Figure 2.1 for examples of the topologies.

Given a path $p$ in a graph $G$, we say $p$ is on an edge $e$ (resp. a node $v$) if $p$ contains $e$ (resp. $v$). Given a set $P$ of paths in a graph $G$, for an edge $e \in E(G)$ and a subset $Q \subseteq P$ we use $L_Q(e) = |\{p \in Q \text{ and } p \text{ is on } e\}|$ and $L_Q = \max_{e \in E(G)} L_Q(e)$ to denote the load of $Q$ on edge $e$ and the load of $Q$ in $G$, respectively. We will simply use $L(e)$ and $L$ for $L_P(e)$ and $L_P$, respectively, and call $L$ the load. We use $\omega(P)$ to denote the *clique number* of $P$, i.e., the maximum number of pairwise intersecting paths in $P$. For a set $P$ of paths, both $L$ and $\omega(P)$ are lower bounds on the number of colors required to color $P$. For a set $P$ of paths in a graph $G$, we say $P$ is *$w$-colorable* if each path of $P$ can be assigned one of the $w$ colors and paths with the same color are edge-disjoint. Such a coloring is referred to as a *valid $w$-coloring*. Similarly, a set of routing requests $R$ is said $w$-colorable if there exists a set $P$ of paths such that each request in $R$ is assigned one of the paths in $P$ and $P$ is $w$-colorable.

Given a set $W = \{\lambda_1, \lambda_2, \ldots\}$ of colors and a set $P$ of paths, a color assignment from $W$ to $P$ is called a *valid coloring* if each path in $P$ is assigned a single color from $W$ and the paths with the same color are edge-disjoint. Finding a valid coloring for $P$ is also called *coloring $P$*. 
A well used strategy for the Min-PC problem is the first-fit coloring: Given a set $W = \{\lambda_1, \lambda_2, \ldots\}$ of colors and a set $P$ of paths, the paths in $P$ are colored one by one in arbitrary order, and a path $p \in P$ is assigned a color $\lambda_i$ with the smallest index $i$ such that no path of $P \setminus \{p\}$ already colored by $\lambda_i$ intersects with $p$.

Given an undirected graph $G = (V, E)$, we can obtain a symmetric directed graph $D = (V, A)$ by replacing every edge $e \in E(G)$ with two arcs with opposite directions. We use $(u, v)$ to denote an arc with tail $u$ and head $v$. We use $m$ for $|A|$, the number of arcs in $D$. We use directed graphs for symmetric directed graphs in this thesis, unless otherwise stated. A directed path is a sequence of consecutive arcs with no repeated node. We use $u \rightarrow v$ to denote a directed path from node $u$ to node $v$. Given a set $P$ of directed paths in a directed graph $D$, we can define the load of $P$ on an arc and the load of $P$ in $D$ in a similar way as in the undirected case. In the following discussion, we will state whether the
graph under consideration is undirected or directed when we give a specific result on the
problems we are interested in. Other graphs constructed (to help solve these problems) are
undirected unless otherwise stated.

Given a set \( P \) of paths in an undirected graph \( G \) (resp. directed graph \( D \)), the conflict
graph associated with \( P \) is the undirected graph \( G_c(P, E_c) \) with the node set \( P \) such that
each node of \( G_c \) corresponds to a path in \( P \) and two nodes of \( G_c \) are adjacent if and only
if the corresponding paths in \( P \) share an edge of \( G \) (resp. an arc of \( D \)). The path coloring
problem for \( P \) in \( G \) is equivalent to the vertex coloring problem in the corresponding conflict
graph \( G_c(P, E_c) \).

An independent set in a graph \( G \) is a set of pairwise non-adjacent vertices. A graph \( G \)
is bipartite if the node set \( V(G) \) is the union of two disjoint independent sets of \( G \). The clique number of a graph \( G \) is the maximum size of a set of pairwise adjacent vertices (called clique) in \( G \). A matching in a graph \( G \) is a set of edges with no shared endpoints. A maximal matching in a graph is a matching that cannot be enlarged by adding an edge. A maximum matching is a matching of maximum size among all matchings in the graph. A maximum matching can be computed in polynomial time, both in bipartite graphs \([74]\) and in general graphs \([91]\). The maximum weight cardinality \( k \) matching is a maximum weighted matching
of size \( k \). It can also be found in polynomial time \([60]\).

Given a (multi)graph \( G \) with vertex set \( V(G) \) and edge set \( E(G) \), an edge-coloring of \( G \)
is an assignment of colors to the edges of \( G \) such that every pair of edges incident to the
same vertex are given different colors. We call such an assignment a valid edge-coloring of
\( G \). It is NP-hard to find a minimum edge-coloring of a graph \([73]\). Recall that \( \Delta(G) \) is the
maximum degree of \( G \). The following results on edge-coloring are well known.

**Proposition 2.1.1** \([114]\) A valid edge-coloring of a multigraph \( G \) using at most \([3\Delta(G)/2]\]
colors can be found in \( O(|E(G)|(|\Delta(G) + |V(G)|)) \) time.

**Proposition 2.1.2** \([95]\) A valid edge-coloring of a multigraph \( G \) using at most \( \max\{[(11\Delta(G)+
8)/10], l(G)\} \) colors can be found in \( O(|E(G)|(|\Delta(G) + |V(G)|)) \) time, where

\[
l(G) = \max \left\{ L(H) = \left\lfloor \frac{|E(H)|}{|V(H)|/2} \right\rfloor \mid H \text{ is a subgraph of } G \text{ with } |V(H)| \geq 3 \right\}
\]
is a lower bound on the number of colors for the edge-coloring of \( G \).

If the number of nodes of a multigraph \( G \) is bounded by a constant, then an optimal
CHAPTER 2. PRELIMINARIES AND RELATED WORK

edge-coloring of $G$ can be computed in polynomial time, using a dynamic programming approach [46].

Given a multigraph graph $G$ and an integer $t$, the maximum edge $t$-coloring problem is to edge-color a maximum subset of edges of $G$ using at most $t$ colors.

The degree-constrained subgraph (DCS) is a generalization of the matching, and can be defined as follows. Given a graph $G$ and functions $b_1, b_2 : V(G) \to \mathbb{N}$ (set of non-negative integers), a DCS is a subgraph $M$ of $G$ such that for each node $v \in V(G)$, $b_1(v) \leq \delta_M(v) \leq b_2(v)$. We call $b_1(v)$ and $b_2(v)$ the lower bound capacity and upper bound capacity of $v$, respectively. We say a node $v$ is matched by edge $e$ in a DCS if $v$ is an end-node of the edge $e$ in the DCS. A DCS with the maximum number of edges can be found (or reported non-existing) in $O(\sqrt{\sum_{v \in V(G)} b_2(v)} |E(G)|)$ time [59]. In the weighted DCS problem, each edge has a real-valued weight. The goal is to maximize the total weight of the edges in the subgraph $M$. A DCS of maximum weight can be found in $O((\sum_{v \in V(G)} b_2(v)) \min\{|E(G)| \log |V(G)|, |V(G)|^2\})$ time [59]. The algorithms of [59] work for multigraphs as well, and a DCS of maximum weight in multigraphs can be found in $O(|E(G)|^2 (\log |V(G)|)(\log \mu_{\text{max}}))$ time, where $\mu_{\text{max}}$ is the maximum edge multiplicity. The DCS problem is similar to the well-known $b$-matching problem. For a given function $b : V(G) \to \mathbb{N}$, a $b$-matching is a set $M$ of edges such that the number of edges in $M$ incident to any node $v$ is at most $b(v)$. See pages 257-259 of [65] for details.

We say an algorithm is a $\rho$-approximation algorithm for a minimization (resp. maximization) problem if the algorithm runs in polynomial time and the value $SOL$ of any solution produced by the algorithm satisfies $\frac{SOL}{OPT} \leq \rho$ (resp. $\frac{OPT}{SOL} \leq \rho$), where $OPT$ is the value of an optimal solution. If such an algorithm exists, we say the problem is $\rho$-approximable. If one can show that such an algorithm does not exist, under the commonly believed $P \neq NP$ or similar conjecture, then the problem is inapproximable up to factor $\rho$. In this thesis, if an algorithm achieves an approximation ratio of $\alpha + \epsilon$, for some constant $\alpha \geq 1$ and a small additive constant $\epsilon > 0$, the running time is usually exponential in $1/\epsilon$. In particular, a polynomial time approximation scheme or PTAS is an algorithm that can achieve an approximation ratio $1 + \epsilon$, for any fixed $\epsilon > 0$, in time that is polynomial in the input size. The notion of MAX SNP was introduced in [100]. An optimization problem does not have PTAS if it is MAX SNP-hard.
2.2 Previous Work

The (off-line) path coloring and call control problems have been extensively studied in both communication and graph theory communities. There have been rich literatures on these problems. In this section, we review previous work related to the path coloring and call control problems. It should be noted that for all problems studied in this thesis, if the graph considered is a chain, then any algorithm for the undirected chain works for the directed chain, by considering each direction separately. Similarly, if the graph considered is a ring and the paths are fixed, then any algorithm for the undirected ring works for directed ring as well, by considering the clockwise and counter-clockwise directions separately.

2.2.1 The Minimum Path Coloring Problem

The path coloring problem is NP-hard in general graphs. In fact, for every graph $G$, there is a set $P$ of paths in a grid-like graph such that the conflict graph of $P$ is isomorphic to $G$. Thus, the path coloring problem is in general as hard as the vertex coloring problem. The vertex coloring problem is hard to approximate with ratio $n^{1-\epsilon}$, for any fixed $\epsilon > 0$ [68]. Thus, the path coloring problem is hard to approximate within ratio $|P|^{1-\epsilon}$ as well. It makes sense to develop approximation algorithms for the path coloring problem in special graphs often found in communications networks. In this section, we review results for the path coloring problem in chains, trees, rings, trees of rings, and meshes. For trees (chains are also trees), the PC and RPC problems are the same, since there is a unique path between any two nodes. For graphs containing cycles, the PC and RPC problems are different because there might exist more than one way to route a connection request. We will also review the RPC problem in such topologies.

Min-PC in chains.

Given a set $P$ of paths in an undirected chain, the conflict graph $G_c$ of $P$ is an interval graph (an interval graph is a graph whose nodes correspond to a set of intervals on a chain, and two nodes are adjacent if and only if the corresponding intervals overlap). Since an interval graph is a perfect graph, the chromatic number of $G_c$ is equal to the clique number of $G_c$ [121]. Thus, $P$ has a valid coloring using $L$ colors. There is a simple greedy algorithm that colors $P$ using $L$ colors: process the nodes in the chain one by one from left to right; when processing a node $v$, consider the paths with left endpoint $v$ in an arbitrary order and
assign a path \( p \) a color \( \lambda \) such that no path intersecting \( p \) is already colored by \( \lambda \) [38]. It is not hard to see that the above algorithm colors \( P \) using exactly \( L \) colors, since at the time a path \( p \) is colored, no path with left endpoint on the right of \( v \) is already colored and at most \( L - 1 \) colored paths intersect \( p \). The Min-PC problem in directed chains can be solved by considering each direction separately. The algorithm for path coloring in chains is often used as a subroutine for path coloring in more complex topologies.

**Min-PC in trees.**

The Min-PC problem in trees has received much attention in the past decade. The Min-PC problem in undirected stars is known to be equivalent to the edge-coloring in multigraphs [105], a well-studied problem in graph theory community. Since the edge-coloring problem in multigraphs is NP-hard, the Min-PC problem in undirected stars (thus undirected trees) is also NP-hard. The Min-PC problem in undirected stars remains NP-hard even if the given set of paths has load \( L = 3 \) [48, 82]. Similar result holds for the directed case: the Min-PC problem in directed trees (with depth at least 2) remains NP-hard even if the given set of paths has \( L = 3 \) [48, 82]. The proofs use reductions from the edge-coloring problem in graphs with maximum degree three [73].

Before we discuss various algorithms for the Min-PC problem in undirected and directed trees, we will give a general framework of these algorithms. All known deterministic algorithms for the Min-PC problem in trees are greedy in the sense that they process the nodes of the tree in a breadth first search (BFS) order, and paths are never re-colored once they are colored. The algorithms work as follows: do a breadth first search of the nodes starting from an arbitrary node of the tree, then process the nodes in this BFS order. When processing a node \( u \), paths on nodes with BFS number strictly smaller than \( u \) are already colored, and their colors will not be changed; all uncolored paths on \( u \) are colored and colors are re-used as much as possible, according to some scheme. This color reusing scheme is important and differs for different algorithms.

It was proved in [48] and [105] that the Min-PC problem in undirected trees can be reduced to the edge-coloring problem in multigraphs. There is a greedy algorithm that accomplishes such task. Suppose we process the nodes of the tree according to the BFS order defined above, and we are now processing node \( u \). Let \( v \) be the parent of \( u \). All paths on \( v \) are already colored. We are now coloring the paths on \( u \), and they can be colored by reducing to the edge-coloring problem. Let \( P' \) be the set of paths on \( u \) that are
already colored. Their colors cannot be changed. However, paths in $P'$ are on the edge $(u, v)$ and thus must have been assigned distinct colors. In any valid edge-coloring of the graph constructed from the paths on $u$, the edges corresponding to paths in $P'$ are assigned different colors. Thus, having pre-colored paths $P'$ does not complicate the edge-coloring.

From the above discussion, one can see that any approximation algorithm for the edge-coloring problem can be used as an algorithm for the Min-PC problem in undirected trees, with the same performance guarantee. In particular, an asymptotic 1.1-approximation algorithm follows from the edge-coloring algorithm of [95]. Note that the Min-PC problem in bounded degree undirected trees can be solved optimally since the edge-coloring problem in multigraphs with constant number of vertices can be solved optimally in polynomial time using a dynamic problem approach (see Theorem 3.1.7 of [46]). For a set $P$ of paths in an undirected tree, there is a valid path coloring of $P$ using at most $\lceil 3L/2 \rceil$ colors [105]. This result follows from Shannon's edge-coloring bound and the equivalence of path coloring in undirected trees and edge-coloring in multigraphs. The $\lceil 3L/2 \rceil$ bound is tight. Consider for example an undirected 3-star with three paths each of which connects a different pair of leaf nodes (see Figure 2.2(a)). The paths have load two but need three colors. The bound also holds for arbitrarily large load, by adding duplicate paths.

The Min-PC problem in directed trees is somewhat different. The Min-PC problem in directed spiders can be solved optimally in polynomial time. In fact, a stronger result is known: for a set $P$ of directed paths in a directed tree $D$, $P$ can be colored by $L$ colors if and only if $D$ is a directed spider [64, 122]. The Min-PC problem in directed binary trees is already NP-hard [48]. The reduction is from circular arc graph coloring. The Min-PC problem remains NP-hard in directed trees of depth-2 even if the given set of paths has load
three [48, 82]. The Min-PC problem remains NP-hard in directed trees of depth-3 even if the given set of paths has load two [47]. On the positive side, it was proved in [75] that in a directed tree, any set \( P \) of paths can be colored by at most \( 5L/3 \) colors. The algorithm has asymptotic approximation ratio \( 5/3 \). It works in a greedy way, as described before. When processing a node \( u \), it extends the partial coloring to include all the paths on \( u \). In doing so, it reduces the problem to the edge-coloring problem of a bipartite graph, in which some edges are already colored.

No local greedy algorithm can do better than \( 5L/3 \). The following result was given in [52]: Let \( A \) be a deterministic greedy Min-PC algorithm in directed trees. There exists an algorithm called \( \text{ADV} \) which, on any input \( \delta > 0 \) and integer \( L > 0 \), outputs a binary directed tree \( T \) and a pattern of communication requests \( P \) of load \( L \) on \( T \), such that \( A \) colors \( P \) with at least \( (5/3 - \delta)L \) colors.

For directed binary trees, a better randomized algorithm is known. It was proved in [16] that any set \( P \) of paths in a directed binary tree can be colored by at most \( 7L/5 \) colors with high probability (thus the algorithm has approximation ratio \( 7/5 \)). Better approximation algorithms are also known for bounded degree directed trees. It was proved in [35] that there exist randomized \( 1.511 + o(1) \)-approximation and \( 1.336 + o(1) \)-approximation algorithms for the Min-PC problem in bounded degree directed trees and binary directed trees, respectively, providing that the load is not small. The \( 1.511 + o(1) \)-approximation algorithm improves a previous \( 1.61 + o(1) \)-approximation algorithm of [34]. The algorithms in [34] and [35] first formulate the Min-PC problem as an integer linear program (ILP), then solve the relaxed linear program (LP) optimally, and get an integral solution through randomized rounding. This approach has also been used by other researchers to obtain good approximation algorithms for many other problems, such as the edge-disjoint paths problem [40, 41]. It works as follows. For a valid coloring of a set of paths, the paths with the same color are edge-disjoint. Thus, the Min-PC problem can be rephrased as covering all the paths with the minimum sets of edge-disjoint paths. It can be formulated as follows:

Minimize \( \sum_{I \in \mathcal{I}} x(I) \)

Subject to \( \sum_{I \in \mathcal{I}, p \in I} x(I) \geq 1, \quad p \in P \)
\( x(I) \in \{0, 1\}, \quad I \in \mathcal{I}. \)

where \( \mathcal{I} \) denotes the collection of all sets of edge-disjoint paths. This integer linear program is NP-hard to solve. One can relax the integral constraint to \( 0 \leq x(I) \leq 1 \). The relaxed
linear program is the formulation for the fractional path coloring problem, which is to color a set of paths with fractional colors such that the sum of the fractional colors assigned to any path is at least one. The fractional path coloring problem is NP-hard in general as well [65], but is solvable for some graphs (in particular, undirected rings and bounded degree directed trees). The fractional solution can then be rounded to get an integral solution using randomized rounding techniques introduced in [104].

On the lower bound side, it was proved in [85] that there is a set $P$ of paths with load $L$ in a directed binary tree such that any valid coloring of $P$ requires at least $5L/4$ colors. This lower bound is for all kinds of algorithms, deterministic or randomized, greedy or non-greedy. In Figure 2.2(b), each arrow represents $L/2$ paths. There are 5 arrows which represent a total of $5L/2$ paths. No more than two paths can be assigned the same color, thus at least $5L/4$ colors are needed. Closing the gap (between the $5L/4$ lower bound and the $5L/3$ upper bound) is an interesting and challenging open problem.

The special case of symmetric paths in directed trees was studied in [36]. For any two nodes $s$ and $t$, the two directed paths $s \to t$ and $t \to s$ are called symmetric. A set of directed paths is called symmetric if it can be partitioned into pairs of symmetric paths. If each pair of symmetric paths is assigned the same color, a set of symmetric paths can be colored using the $3L/2$ algorithm for undirected trees. In this way, a $3L/2$ upper bound can be obtained for the case of symmetric paths. If the two symmetric paths are not required to use the same color, then a $1.367L + o(L)$ upper bound is known for directed binary trees [37]. This is better than the $7L/5$ bound for directed binary trees in which paths are not symmetric. In general, however, it is not known whether the Min-PC problem is easier if the set of paths is restricted to be symmetric. In particular, no better lower bound is known in the symmetric case. Caragiannis et al. showed that for every $\delta > 0$, there exists an integer $l > 0$, a directed binary tree and a set $P$ of symmetric paths on the tree with load $L = 4l$ such that no algorithm can color $P$ using less than $(5/4 - \delta)L$ colors [36].

**Min-PC in rings.**

The Min-PC problem in undirected rings is NP-hard [61]. The Min-PC problem in undirected rings is also known as circular arc graph coloring, since the conflict graph is a circular arc graph (a circular arc graph is a graph whose nodes correspond to a set of paths on a ring, and two nodes are adjacent if and only if the corresponding paths overlap). It was shown in [118] that any set $P$ of paths with load $L$ in an undirected ring can be colored by
CHAPTER 2. PRELIMINARIES AND RELATED WORK

Figure 2.3: (a) A $2L - 1$ lower bound on undirected rings; (b) A $\frac{3}{2}\omega(P)$ lower bound on undirected rings.

$2L - 1$ colors. The algorithm is very simple: select one node $v$ which is an end-node of some paths, and color the paths with $v$ as an internal node using at most $L - 1$ colors, and the remaining paths (which are on an undirected chain) using at most $L$ colors. The algorithm has an approximation ratio of 2 since $L$ is a lower bound on the number of colors required for any optimal solution. The $2L - 1$ upper bound is tight, since there are instances that require this many colors. For example, consider five paths which are pairwise intersecting and cover the whole ring (see Figure 2.3(a)). The load is three, but five colors are needed. For any integer $L \geq 2$, it is easy to construct a set $P$ of paths with load $L$ such that $2L - 1$ colors are needed to color $P$.

A generalization of the $2L - 1$ upper bound is known. Let $l$ be the minimum number of paths in $P$ that are needed to cover the whole ring. Tucker showed that if $l \geq 4$, then $\left\lceil \frac{3}{2}L \right\rceil$ colors suffice to color $P$ [118]. This result is further generalized in [119]. It was shown that if $l \geq 5$, then $\left\lceil \left(\frac{5}{2}\right) L \right\rceil$ colors suffice to color $P$, and this bound is tight.

A better deterministic approximation algorithm is known. It was proved in [77] that any set $P$ of paths in an undirected ring can be colored by $\frac{3}{2}\omega(P)$ colors. The algorithm has approximation ratio 1.5, which is the current best ratio among all deterministic algorithms. Roughly speaking, the algorithm works as follows. Let $L$ be the load of the given set $P$ of paths. The algorithm first chooses $\left\lfloor \frac{L}{2} \right\rfloor$ sets of paths along the clockwise direction of the ring, and then chooses at most $L$ sets of paths along the counter-clockwise direction of the ring. Each of these sets contains a set of edge disjoint paths and thus can be colored by
a single color. If there is no path left during the two phases, then obviously at most \( \left\lfloor \frac{3}{2}L \right\rfloor \)
colors are used. Otherwise, more than \( \left\lfloor \frac{3}{2}L \right\rfloor \) colors are needed, but it was shown that the
clique number \( \omega(P) \) of the set \( P \) of paths is greater than \( L \) and at most \( \frac{3}{2} \omega(P) \) colors are
needed. The \( \frac{3}{2} \omega(P) \) upper bound is also tight. Consider a set of five paths as shown in
Figure 2.3(b). The conflict graph is a pentagon which has a maximum clique of size two,
but has chromatic number three.

The current best (randomized) approximation algorithm for the Min-PC problem in
undirected rings achieves an asymptotic approximation ratio of \( 1 + 1/e \) (about 1.37) with
high probability, if the load is not small [84]. The algorithm first transforms the problem
into an integral multi-commodity flow problem, then solves the relaxed linear programming
problem, and finally obtains a solution using a randomized rounding technique introduced
in [104]. Caragiannis et al. obtained the following generalized result: if the load is not
small, there exists a polynomial time randomized algorithm which has approximation ratio
\( 1 + \ln\left(\frac{1}{1-\frac{1}{e}}\right) + o(1) \) with high probability, where \( l \) is the minimum number of paths required
to cover the ring [35].

The Min-PC problem in directed rings can be solved similarly by considering each di­
rection separately, since the paths are fixed.

**Min-RPC in rings.**

As mentioned before, the ring is the simplest graph in which routing is an issue. The
routing and path coloring problem seems to be harder than the path coloring sub-problem
alone (for many topologies other than tree). In fact, it is not clear how to simultaneously
solve the routing problem and the path coloring problem. Most algorithms solve the two
sub-problems in two steps: first the requests are routed, then the obtained paths are colored.

The Min-RPC problem is NP-hard for both undirected and directed rings [48]. The
Min-RPC problem in undirected rings was studied in [105], and the problem in directed
rings was studied in [92]. They proposed the following method: remove an arbitrary link
from the ring and obtain an chain, then route (uniquely) all the requests on the chain, and
color the paths using the optimal path coloring algorithm for chains. This approach has
an approximation ratio of two (for both undirected and directed rings). Suppose the paths
routed on the chain have load \( L \) and thus are colored by \( L \) colors. Consider an edge \( e \) on the
chain with load \( L \). Any optimal algorithm (for the Min-RPC problem on rings) has to route
the requests corresponding to paths on \( e \) along one of the two possible routes on the ring.
This results in a load of at least $\lceil L/2 \rceil$ on the ring and thus requires at least $\lceil L/2 \rceil$ colors. Better approximation algorithms are known for undirected rings. In particular, Kumar gave an algorithm which uses at most $OPT \cdot (1.5 + 1/2e + o(1)) + O(\sqrt{OPT \cdot \ln n})$ colors with high probability [83]. The approximation ratio is close to $1.5 + 1/2e$ if $OPT$ is large, but may be greater than 2 if $OPT$ is not large enough. Cheng gave an approximation algorithm which has an approximation ratio of $2 - 1/\Theta(\log n)$ with high probability [42]. This value holds for all values of $OPT$, although asymptotically it is not better than Kumar’s. We are not aware of any deterministic algorithm which achieves a constant approximation ratio strictly better than 2.

**Min-PC in trees of rings.**

The Min-PC problem in trees of rings is clearly NP-hard, since a ring is the simplest tree of rings, and the Min-PC problem in rings is NP-hard, for both undirected and directed cases. The Min-PC problem in undirected trees of rings was first studied in [44]. They gave a 2-approximation algorithm for undirected trees of rings of maximum degree four (i.e., each node can be contained in at most two rings). Erlebach showed that a set $P$ of paths on an undirected tree of rings $TR$ can be colored by at most $4L$ colors [47]. The $4L$ algorithm can be extended to work for directed trees of rings, with an upper bound of $8L$. His algorithm works as follows.

**Algorithm** GreedyColoring ($TR, P$):

1. Initially, all paths are uncolored.

2. Process each node $u$ of $TR$ in a depth first search (DFS) order starting from an arbitrary node $s \in V$ as follows:

   Let $P_u$ be the set of uncolored paths on $u$. Assign every path $p$ of $P_u$, in arbitrary order, the color $\lambda$ with the smallest index (break tie arbitrarily) such that no path intersecting $p$ is already colored by $\lambda$.

   The coloring strategy used for $P_u$ in the algorithm is the *first-fit strategy*. Although the first-fit approach is simple (and effective sometimes), it is not always the best possible. It was shown in [25] that a set $P$ of paths on an undirected tree of rings with maximum degree eight can be colored by at most $3L$ colors. The $3L$ algorithm is based on a processing order first proposed by Erlebach [47], and a better color reusing strategy when processing each
node. They further gave a 2-approximation algorithm for undirected trees of rings with degree at most four and a 2.5-approximation algorithm for undirected trees of rings with degree at most six.

There are instances that require $3L$ colors even on undirected trees of rings with maximum degree four [25]. An example of such instances is as follows: Let $P = A \cup B \cup C \cup D \cup E \cup F$ be the set of paths, with each subset having $L/2$ ($L$ is even) paths, as shown in Figure 2.4. The maximum number of paths on any link in the tree of rings is $L$. Any two paths in $P$ must be assigned different colors since they intersect with each other. There are a total of $3L$ paths in $P$, thus $3L$ colors are needed. This lower bound shows that in the worst case one cannot do better than $3L$ even for undirected trees of rings with node degree four.

**Min-RPC in trees of rings.**

The Min-RPC problem in trees of rings is NP-hard, following from the NP-hardness of the problem in rings. The following method is known for the Min-RPC problem on trees of rings: cut one link from each ring in the given tree of rings and obtain a tree, then route (uniquely) the requests and solve the Min-PC problem in the obtained tree. It is known
that through this cut-one-link method, any algorithm for the Min-PC problem in undirected
trees using at most $\alpha L$ colors can be used to obtain a $2\alpha$-approximation algorithm for the
Min-RPC problem in undirected trees of rings [105]. The same result holds for the directed
case as well [92]. The best upper bounds (in terms of load) are $3L/2$ and $5L/3$ for the
Min-PC problem in undirected and directed trees, respectively. Thus, one can obtain 3-
approximation and $10/3$-approximation algorithms for the Min-RPC problem in undirected
and directed trees of rings, respectively.

For undirected trees of rings with degree at most eight, a 3-approximation algorithm not
based on cut-one-link can be obtained as follows: route optimally the requests such that the
load $L$ is minimized, then use the $3L$ algorithm of [25] to color the obtained paths. Since
$L$ is the optimal load and thus is a lower bound for the Min-RPC problem, the algorithm
has approximation ratio 3. It is interesting to design approximation algorithms with ratios
strictly better than 3 and $10/3$ for the Min-RPC problem in undirected and directed trees
of rings, respectively.

Min-RPC in meshes.

The Min-PC problem in meshes is equivalent to the vertex color problem in general graphs.
Thus, any hardness or algorithmic result on vertex coloring applies to the Min-PC problem
in meshes.

The Min-RPC problem in $N \times N$ undirected square mesh networks is NP-hard, since the
special case of edge-disjoint paths is already NP-hard [103]. It was shown in [103] that the
number of colors needed can be approximated with a constant factor. The author formalized
the problem as a sequence of integer linear programs. It was then shown that the solutions
to the relaxations provide a constant factor approximation to the number of colors needed.
The ratio was established through a complicated randomized rounding procedure, which
proceeds iteratively. In each iteration, the solution gets closer to an integral solution. After
a small number of iterations, the solution is near-integral. The author was then able to
convert it into an integral solution without much increasing in the number of colors. The
above procedure gives a constant factor approximation to the number of colors needed.
Unfortunately, the argument is non-constructive and does not give a routing and path
coloring. Nevertheless, the author showed that the Min-RPC problem can be approximated
within a factor of $\text{poly}(\ln \ln N)$. This improves a previously best ratio of $O(\ln N)$ [80] (which
is an improvement over an even earlier ratio of $O(\ln^2 N)$).
2.2.2 The Maximum Path Coloring Problem

In this section, we review results for the maximum (routing and) path coloring problem in chains, trees, rings, and trees of rings. The maximum routing and path coloring (Max-RPC) problem can be defined as follows: Given a set \( R \) of requests and an integer \( w > 0 \), find a maximum cardinality subset \( R' \subseteq R \) of requests that is \( w \)-colorable. The maximum path coloring (Max-PC) problem is similar to Max-RPC, but the paths are given as part of the input. For trees, Max-RPC and Max-PC coincide since the routing is unique. When \( w = 1 \), the Max-RPC (resp. Max-PC) problem is equivalent to the MEDP (resp. MEDPwPP, the maximum edge-disjoint paths with pre-specified paths). The MEDP problem has attracted much attention in theoretical computer science, and we will review it in more details in Section 2.2.5.

Before we introduce the individual algorithms for Max-PC in various graphs, we describe a generic approach (hereinafter will be called iterative greedy approach). Suppose we have a \( \rho \)-approximation algorithm for the MEDPwPP problem (equivalently, a \( \rho \)-approximation algorithm for the Max-PC problem with only one color). To solve the Max-PC problem, we call the algorithm for the MEDPwPP problem, select a maximum cardinality subset \( P' \subseteq P \) of pairwise disjoint paths, and remove the set \( P' \) of paths from \( P \) (i.e., \( P := P \setminus P' \)). The procedure is repeated \( w \) times. Each selected subset is colored by a distinct color. The union of the \( w \) chosen sets is taken as the solution for the Max-PC problem. It is known that if the MEDPwPP algorithm has approximation ratio \( \rho \), then this iterative algorithm for the Max-PC problem has approximation ratio \( \frac{1}{1 - e^{-1/\rho}} \) (which is at most \( \rho + 1 \)) [54]. This simple approach sometimes gives very good approximation for the maximum path coloring problem.

Max-PC in chains.

The Max-PC problem in undirected chains was studied in [38]. It was shown that a simple greedy algorithm can solve the problem optimally. They also gave a clever linear time implementation. The algorithm is very simple and works as follows: process the nodes of the chain one by one from left to right; when processing a node \( v \), consider greedily the paths with right endpoint \( v \), and include a path \( p \) if \( p \) can be assigned a color \( \lambda \) such that no path intersecting \( p \) is already colored by \( \lambda \). The weighted Max-PC problem in chains can also be solved optimally in polynomial time, by reducing to a minimum cost network flow...
problem. The same result holds for the directed chains. This algorithm is often used as a subroutine for solving the maximum path coloring problem in more complex topologies.

**Max-PC in trees.**

When $w = 1$, the Max-PC problem is simply the MEDP problem, and is known solvable in undirected trees [63]. Erlebach proved that the MEDP problem is NP-hard and MAX SNP-hard in directed trees [49]. The MAX SNP-hardness of the MEDP problem in directed trees implies that it does not have polynomial time approximation scheme (PTAS) unless $P = NP$. Erlebach also gave a $5/3 + \epsilon$-approximation algorithm for the MEDP problem in directed trees, where $\epsilon$ can be chosen arbitrarily small [49]. The Max-PC problem in undirected and directed trees with $w > 1$ was studied in [46]. It was showed that the Max-PC problem is solvable if $w$ and the degree of the (undirected or directed) tree are both bounded by a constant, using a dynamic programming approach. If either $w$ or the degree is unbounded, the problem is NP-hard. For arbitrary $w$, Erlebach used the iterative greedy approach to obtain 1.58-approximation algorithms for undirected trees and bounded degree directed trees, and a 2.22-approximation algorithm for directed trees with arbitrary degree. The iterative greedy approach seems to be one of the best known tools so far for dealing with the Max-PC problem in trees (and many other topologies).

The Max-PC problem in undirected stars is NP-hard [48, 105]. It is equivalent to the maximum edge $t$-coloring problem in multigraphs. The latter problem was studied by Feige et al. [56]. They showed that for every fixed $t \geq 2$ there is some $\epsilon > 0$ such that it is NP-hard to approximate maximum edge $t$-coloring within a ratio better than $1 - \frac{1}{2\epsilon t}$. They also gave an approximation algorithm whose ratio tends to $\alpha$ as $t \to \infty$, where $\alpha$ is the best approximation ratio for the edge-coloring in multigraphs. The current best approximation algorithm for the edge-coloring problem has asymptotic approximation ratio 1.1. Thus, the maximum edge $t$-coloring problem can be approximated with an asymptotic ratio 1.1 as $t \to \infty$. Accordingly, the Max-PC problem in undirected stars can be approximated with an asymptotic ratio 1.1 if the number of available colors $w$ approaches infinity.

The Max-PC problem in directed stars and spiders can both be solved optimally in polynomial time as follows: first select a maximum subset of paths with load at most $w$; then color the selected paths using $w$ colors. The first step is actually a call control problem (see Section 2.2.4). It can be solved optimally in polynomial time for directed stars, since it is equivalent to the $b$-matching problem in bipartite graphs, which is known to be solvable.
It can also be solved optimally for directed spiders, since Erlebach et al. showed that the call control problem in directed spider can be solved optimally in polynomial time by reducing to a network flow problem [55]. The second step can also be done in polynomial time, since in any directed spiders (thus stars), any set of paths with load $w$ can be colored by $w$ colors [64, 122].

Max-PC and Max-RPC in rings.

For general values of $w$, the Max-PC and Max-RPC problems (and the weighted cases) on rings are NP-hard, for both the undirected and directed cases [98, 99]. This follows easily from the NP-hardness of the corresponding minimization problems. Several methods are well used for solving these problems on rings. Many existing work use either a single method or a combination of several methods. The first method, called the *cut-one-link* method, simply ignores one link of the ring and solves the problem on the obtained chain using an algorithm of [38] or its variants. (We have already seen the use of the cut-one-link approach in Section 2.2.1.) The second method, called the *maximum matching* method, tries to use one color for two requests (or paths) according to a matching found in the auxiliary graph constructed. The third method is the *iterative greedy approach* already discussed, which solves the problem by calling $w$ times an approximation algorithm for the single color case. Intuitively, the iterative greedy method may be better than the maximum matching method, since the iterative greedy method may color more than two paths using a single color, while the maximum matching method always colors at most two paths using a single color.

The Max-PC problem on rings was first studied by Wan and Liu [120], who gave a polynomial time exact algorithm for the MEDPwPP problem in rings, and then used the iterative greedy approach to get a 1.58-approximation algorithm (for both the undirected and the directed cases). The approximation ratio was improved to 1.5 in undirected rings [99] (the algorithm works for directed rings with the same performance ratio). The 1.5-approximation algorithm is based on cut-one-link and maximum matching. Let $e$ be an arbitrary edge on the ring, $P_e$ be the set of paths on $e$ and $P_c$ be the set of paths not on $e$. The algorithm first cuts the ring at $e$ and then selects a maximum $w$-colorable subset of paths from $P_c$. The algorithm then tries to color some paths in $P_e$, using colors each of which is used to color only one paths in $P_c$. This latter step is done based on a maximum matching in a bipartite graph with bipartitions $P_e$ and $P_c$. 
Very recently, Caragiannis gave a 4/3-approximation algorithm for the Max-PC problem in undirected rings [33]. His algorithm is based on cut-one-link and a variant of the iterative greedy method. Essentially, he showed that if $OPT/w$ is large, then the cut-one-link approach ensures a good approximation ratio; otherwise ($OPT/w$ is small), the iterative greedy method has a good approximation ratio. The algorithm works for Max-PC in directed rings with the same approximation ratio. In the same paper, he also gave a randomized 1.49015-approximation algorithm for the weighted Max-PC problem in rings.

For the Max-RPC problem in undirected and directed rings, the iterative greedy algorithm also works and has approximation ratio 1.58. Nomikos et al. gave a 1.5-approximation algorithm for undirected rings and a 11/7-approximation algorithm for directed rings, both based on two separate algorithms, cut-one-link and maximum matching, and the best of the two is taken as the final solution [98]. Caragiannis improved the ratios to 4/3 and 1.41257 for undirected and directed rings, respectively, by using cut-one-link with iterative greedy algorithm [33].

**Max-PC and Max-RPC in trees of rings.**

When $w = 1$, the Max-PC and Max-RPC problems reduce to MEDPwPP and MEDP problems, respectively. The MEDPwPP problem is NP-hard and MAX SNP-hard and can be approximated with ratios 4 and 8 in undirected and directed trees of rings, respectively [47]. It was also shown in [47] that the MEDP problem is NP-hard in undirected and directed trees of rings, and any $\alpha$-approximation algorithm for MEDP in trees gives a $3\alpha$-approximation algorithm for MEDP in trees of rings, both in the undirected case and in the directed case. Since the MEDP problem in undirected and directed trees has exact algorithm and $5/3 + \epsilon$-approximation algorithm, respectively, the MEDP problem can be approximated with ratio 3 and $5 + \epsilon$ in undirected and directed trees of rings, respectively.

The Max-PC and Max-RPC problems with arbitrary $w$ are both NP-hard in trees of rings. Currently, there is no algorithm better than the iterative greedy approach. Simple calculations show that the iterative greedy approach has approximation ratios close to 4.5 (resp. 8.5) for Max-PC in undirected (resp. directed) trees of rings, and has approximation ratios close to 3.5 (resp. 5.5) for Max-RPC in undirected (resp. directed) trees of rings.
2.2.3 The Path Multicoloring Problem

Many optical networks have multiple parallel optical fibers in a link. Such a network is known as a multifiber optical network. For a link $e$ with $\mu(e)$ fibers, up to $\mu(e)$ routing paths can use the same color on link $e$. Note that in directed multifiber networks, both directions of a link should have the same number of fibers. Figure 2.5(a) shows a set of three paths in a single fiber undirected star. The paths need three colors, since they are pairwise intersecting. On the other hand, in Figure 2.5(b), the three paths need only one color in a 2-fiber undirected star. Thus, less colors are needed in multifiber optical networks, at the cost of extra fibers. The path coloring problem in multifiber optical networks is known as the path multicoloring (PMC) problem [97]. Recently, there are renewed interests in multifiber optical networks and the PMC problem [53, 54, 86, 87, 88, 89, 96, 97]. Inapproximable results were given in [10, 11, 12]. As in the single fiber case, there are two optimization problems in multifiber optical networks: the minimum path multicoloring problem (denoted as Min-PMC), and the maximum path multicoloring (Max-PMC) problem. Multifiber networks are distinguished into two types according to the number of fibers in every link. One is a uniform network in which every link has the same number of fibers. The other is a non-uniform network in which different links may have different number of fibers. We will use $k$-fiber network to denote a uniform network with $k$ fibers in every link in the undirected case. In a directed $k$-fiber network, both directions of a link have $k$ fibers. When $k = 1$, the Min-PMC and Max-PMC problems are simply the conventional Min-PC and Max-PC problems, respectively. In the single fiber case, the load $L$ is a lower bound on the number of colors needed. Similarly, in the multifiber case, $w_{lb} = \max_{e \in E} \left[ \frac{L(e)}{\mu(e)} \right]$ is a lower bound for the number of colors required. The maximization is taken over both directions of a link in the directed case.

The Max-PMC problem asks to maximize the number (or the weight) of paths that can be colored by a given number $w$ of colors in a multifiber optical network. When $w = 1$, this is simply a maximum edge-disjoint paths problem in a single fiber network, and is a call control problem (with edge capacity $c(e) = \mu(e)$ on edge $e$) in a multifiber network. Thus, a natural way to solve the Max-PMC problem is the iterative greedy approach (see Section 2.2.2). Erlebach et al. showed that if the call control algorithm has an approximation ratio of $\rho$, then the iterative greedy approach has an approximation ratio of $\frac{1}{1 - e^{-1/\rho}}$ (which is at most $\rho + 1$) [54]. The proof is similar to the one in [120].
CHAPTER 2. PRELIMINARIES AND RELATED WORK

Figure 2.5: A single fiber undirected star and a 2-fiber undirected star.

Min-PMC and Max-PMC in uniform $k$-fiber networks.

The Min-PMC problem in $k$-fiber undirected chains can be solved optimally in polynomial time [96]. In fact, the optimal algorithm uses exactly $w_{lb}$ colors. For the Max-PMC problem, suppose we have $w$ colors. One can reduce the Max-PMC problem to the following call control problem: set the capacity of an edge $e$ to be $w \times k$. Then solve the call control problem optimally. The paths in the solution to the call control problem can be colored by $w$ colors, since $w_{lb} = (w \times k)/k = w$.

For every even $k > 1$, the Min-PMC problem is known solvable in polynomial time in $k$-fiber undirected stars [87, 88]. The Max-PMC problem in even $k$-fiber undirected stars can be solved optimally by first reducing to a call control problem with every edge $e$ having capacity $k \times w$. The Min-PMC and Max-PMC problems are NP-hard for multifiber undirected and directed binary trees. This should be contrasted to the path coloring problem in bounded degree undirected trees, which is known solvable. If path lengths are restricted to at most three, the Min-PMC and Max-PMC problems in 2-fiber undirected trees are still NP-hard. If path lengths are restricted to four, the Min-PMC and Max-PMC problems in multifiber directed or undirected trees are NP-hard. An upper bound of $3L/(2k)$ is known for the Min-PMC problem in undirected $k$-fiber trees. This is a generalization of the $3L/2$ upper bound in the single fiber case, and the bound is tight.

The Min-PMC problem in $k$-fiber undirected rings is NP-hard, for every $k > 1$, as shown in [87]. An upper bound of $[(k + 1)/k \cdot L - 1]$ is also established in the same paper.
CHAPTER 2. PRELIMINARIES AND RELATED WORK

Min-PMC and Max-PMC in non-uniform multifiber networks.

The Min-PMC and Max-PMC problems in non-uniform multifiber undirected chains can both be solved optimally in polynomial time, in a similar way as in the uniform case [96]. The Min-PMC problem in undirected and directed rings can be approximated with a ratio of two, no matter whether the paths are fixed or not [96]. The Max-PMC problem can be approximated with a ratio of 1.58 in undirected and directed rings if the paths are fixed, and with a ratio slightly worse than 1.58 in undirected rings if the paths are not fixed, using the iterative greedy approach.

The Min-PMC problem can be solved optimally in directed stars, and can be approximated with ratio 1.5 in undirected stars [96]. The Min-PMC problem in undirected and directed spiders can be approximated with ratios 2.5 and 2, respectively [96]. The Min-PMC problem in undirected and directed trees can be approximated with a ratio of 4, a by-product of [41]. The Max-PMC problem in undirected trees can be approximated with a ratio of 2.542, using the iterative greedy approach.

2.2.4 The Routing and Call Control Problem

The routing and call control (RCC) problem can be defined as follows: Given a set \( R = \{(s_i, t_i) \mid i = 1, \ldots, k\} \) of requests in a graph \( G = (V, E) \) where each edge \( e \in E \) is assigned a non-negative integer capacity \( c(e) \), find a maximum subset \( R' \subseteq R \) such that the requests in \( R' \) can be routed without violating the capacity constraint, i.e., at most \( c(e) \) requests use edge \( e \). Such a subset of requests is called a routable set. In the directed case, each direction of a link is assigned a capacity, and the capacity constraint is violated only if the number of paths on the same direction of a link exceeds the capacity in that direction. The problem is known as the call control problem if the paths are given. Each request \( (s_i, t_i) \) may be assigned a positive weight \( w_i \), and the goal is to find a routable set with maximum total weight. In this case, the problem is known as the weighted routing and call control problem. Further, each request \( (s_i, t_i) \) may be associated with an integer demand \( d_i \geq 1 \), and the request can be realized only if the demand \( d_i \) is fully satisfied. If the request is required to be on a single path, i.e., unsplittable, then the problem is known as the unsplittable flow problem (UFP). In this thesis we assume that the requests are not splittable. Of course, for graphs like trees, there is no splittability problem since there is a unique path between any two nodes. The unsplittable flow problem is clear NP-hard, since when the graph is a single
edge with all demands going across it, the problem simply becomes the Knapsack problem, which is NP-hard [62]. When every edge has unit capacity, the routing and call control problem is known as the maximum edge-disjoint paths problem, which will be discussed in Section 2.2.5. Any inapproximability result for the MEDP problem holds for the routing and call control problem.

The unsplittable flow problem can be modeled as the following integer linear programming problem when the paths are given:

\[
\begin{align*}
\text{Maximize} & \quad \sum_{i=1}^{k} w_i x_i \\
\text{Subject to} & \quad \sum_{e \in \text{Path}(i)} d_i x_i \leq c(e), \quad \forall e \in E \\
& \quad x_i \in \{0, 1\}, \quad i = 1, \ldots, k.
\end{align*}
\]

When paths are not given, the formulation is similar but more complex. In the formulation, \(\text{Path}(i)\) is the path along which the request \(i\) is routed. The constraint basically says that the total demands of the requests routed on edge \(e\) can be at most \(c(e)\). This formulation is quite generic. For example, setting \(d_i\) to one gives the formulation for the weighted call control problem, and setting both \(d_i\) and \(w_i\) to one gives the (unweighted) call control problem. Since the unsplittable flow problem is in general NP-hard, the above integer linear program cannot be solved in polynomial time. However, the relaxed linear program (by setting \(x_i \in [0, 1]\)) can be solved in polynomial time. The ratio between the fractional solution and the integral solution is usually called the \textit{integrality gap}. In general, the integrality gap could be large for arbitrary graph. For example, it was shown in [63] that there is a grid-like 3-regular graph in which the integrality gap is \(\Omega(\sqrt{n})\) for the MEDP problem.

In the following discussion, we will mainly review the call control problems. We will also review the routing and call control problems in rings. The unsplittable flow problem will be discussed in Section 2.2.4. The discussion of this section is restricted to undirected graphs.

\textbf{Call control in chains.}

The call control problem in undirected chains was studied in [5], and it was shown that the problem can be solved optimally in polynomial time, using a simple greedy algorithm. They also gave a linear time implementation. The weighted call control problem can also be solved in polynomial time, by a similar approach as [38]. If all edge capacities are the same, the problem is equivalent to the Max-PC problem, since any set of paths with load \(L\) can be colored by \(L\) colors in undirected chains (see Section 2.2.1).
CHAPTER 2. PRELIMINARIES AND RELATED WORK

Call control in trees.

The call control problem in undirected trees was studied in [63]. They showed that the call control problem in undirected stars is equivalent to the $b$-matching problem in multigraphs, and can be solved optimally in polynomial time. They also showed that the call control problem in undirected trees is equivalent to a generalization of $b$-matching called cross-free cut $b$-matching. They further showed that the call control problem is NP-hard and MAX SNP-hard even in depth-3 undirected trees with capacities one or two. The reduction is from the maximum three-dimensional matching problem. They formulated the call control problem as an integer linear program and showed that the dual of the relaxed linear program is exactly the minimum multi-cut problem in undirected trees (the minimum multi-cut problem asks for a minimum number of edges whose removal disconnects all the given pairs in a graph). They also gave a primal-dual algorithm that achieves approximation ratio two. The primal-dual method works as follows in approximation algorithms: start with arbitrary solutions to the primal and dual linear programs, and make alternate improvements to each, until good integral solutions to both are found. The improvements are guided by the complementary slackness conditions.

The weighted call control problem in undirected stars is equivalent to the weighted $b$-matching problem which can also be solved optimally in polynomial time. The weighted call control problem in undirected trees was studied in [41]. They also formulated the problem as an integer linear program, and showed that the relaxed linear program has integrality gap at most 4. This gives a 4-approximation algorithm. Their algorithm implies the following result. Let $J$ be any (multi)set of requests, and $k$ be an integer, such that for any edge $e$ in the tree, at most $k \times c(e)$ of the paths contain $e$. Then $J$ can be partitioned into $4k$ routable sets. This latter result also implies that the (non-uniform) path multicoloring problem on undirected trees can be approximated with ratio 4 (see Section 2.2.3 and [54]).

Routing and call control in rings.

The call control problem in undirected rings is solvable in polynomial time [5]. This is achieved by calling the optimal chain algorithm iteratively.

The routing and call control problem in undirected rings was studied in [7], and it was shown that there is an algorithm that admits at least $OPT - 3$ requests, where $OPT$ is the maximum number requests that can be admitted in any optimal solution. The algorithm is
based on linear programming. They first formulate the problem as an integer linear program, and then solve the relaxed linear program. They obtain a solution by some sophisticated rounding techniques. The complexity of the problem is still unknown.

The weighted call control and weighted routing and call control problems in undirected rings can both be approximated with ratio two, using a simple cut-one-link approach [6]. However, the complexity status of both problems is still unknown. In particular, it was shown in [6] that the exact matching problem in bipartite graphs can be reduced to the weighted call control problem in undirected rings, thus the weighted call control problem in undirected rings is at least as hard as the exact matching problem. The exact matching problem is known solvable in random polynomial time (RP) [93], but no (deterministic) polynomial time algorithm is known. Finding the complexity of these problems is interesting but probably challenging as well.

Routing and call control in trees of rings.

We are not aware of any algorithmic results on the routing and call control problem in trees of rings (whether paths are given or not), when the edge capacities can be arbitrary. The unit edge capacity case will be discussed in Section 2.2.5. The call control problem in trees of rings with unit edge capacities has also been studied from a different prospective, with the goal of rejecting as few requests as possible [8].

The unsplittable flow problem.

Recall that in the unsplittable flow problem, each request \((s_i, t_i)\) has an integer demand \(d_i \geq 1\). Let \(d_{\text{max}} = \max_{i=1}^{k} d_i\) be the maximum demand, and \(c_{\text{min}} = \min_{e \in E} c(e)\) be minimum edge capacity. The result for the unsplittable flow problem generally requires \(d_{\text{max}} \leq c_{\text{min}}\), which has been known as the no-bottleneck assumption. Without this assumption, the unsplittable flow problem is hard to approximate within a factor of \(\Omega(m^{1-\epsilon})\) unless \(P = NP\) [17]. The unsplittable flow problem in undirected chains has been studied extensively. In the uniform capacity case (every edge has the same capacity, not necessarily one), the first constant factor approximation algorithm was given in [102], who obtained a 6-approximation algorithm. The ratio was then improved to 3 [19], and then to \(2 + \epsilon\) [32]. For the unsplittable flow problem in undirected chains with arbitrary capacity, the first constant factor (close to 80) approximation algorithm was given in [39]. This ratio was then improved to \(2 + \epsilon\).
In [18], it was shown that there is a quasi-polynomial time (i.e., in time $2^{\text{polylog}(n)}$) approximation scheme for the unsplittable flow problem on undirected chains and rings, provided that all capacities and demands are integers bounded by $2^{\text{polylog}(n)}$. Previously, it was not known whether the unsplittable flow problem in chains is MAX SNP-hard or not. This result rules out a MAX SNP-hard result unless $NP \subseteq DTIME(2^{\text{polylog}(n)})$.

The unsplittable flow problem in undirected stars is also known as the demand matching problem, which has also been proved to be MAX SNP-hard [115]. The demand matching problem can be defined as follows: Given a graph $G = (V, E)$ with each node $v$ assigned an integer capacity $b_v$, each edge $e$ having an integral demand $d_e$ and a weight $w_e$, find a subset $M \subseteq E$ with maximum total weight such that the sum of the demands of the edges incident to $v$ is at most $b_v$. Obviously, when $d_e = 1$ for every edge $e \in E$, the demand matching is actually a $b$-matching. For the demand matching problem in bipartite graphs, the integrality gap is between $2^{1/2}$ and $2^{1/5}$. For general graphs, the integrality gap is between 3 and $3^{3/10}$.

The unsplittable flow problem in undirected trees was studied in [41]. The problem was formulated as an integer linear program. It was shown that the integrality gap for the demand version is at most 11.542 times that for the unit demand case. The latter has an integrality gap of 4, thus the integrality gap for the demand version is at most 48.

### 2.2.5 The Maximum Edge-disjoint Paths Problem

The maximum edge-disjoint paths (MEDP) problem can be defined as follows: Given a set of $k$ source-destination pairs in a graph $G$, connect as many of these pairs as possible using edge disjoint paths. It is not hard to see that the maximum edge-disjoint paths problem is a special case of the routing and call control problem in which every edge has unit capacity. The maximum edge-disjoint paths problem is regarded as one of the classic NP-hard problems. In fact, its decision version is one of Karp’s original NP-complete problems [78]. It has received much attention during the past several decades. In the maximum edge-disjoint paths with pre-specified paths (MEDPwPP) problem, a set of paths (instead of source-destination pairs) is given. Of course, for graphs like trees, there is no routing problem. In the weighted case, each routing request (or path) may be given a positive weight, and the goal is to maximize the total weight of accepted requests (or paths).

In undirected general graphs, the MEDP problem is solvable for any fixed $k$ [111], but is NP-hard for general value $k$. The MEDP problem is hard to approximate within ratio
log$^{\frac{1}{3} - \epsilon} n$ for any fixed $\epsilon > 0$ unless $NP \subset ZPTIME(n^{\text{poly}(\log n)})$ [9]. However, there is no corresponding poly-logarithmic upper bound on the approximation ratio for all undirected graphs yet, and whether such upper bound exists or not is an outstanding open problem. Until now, poly-logarithmic approximation algorithms are known only for the following undirected graphs: trees with parallel edges, grids and grid-like graphs, expanders, even-degree planar graphs, and planar graphs in which certain congestion on the edges is allowed (see [79] and the references therein). The general approach for approximating MEDP in these graphs is to decompose the graph into node disjoint induced subgraphs, and then find a large routable set within each subgraph. The union of the solutions for the subgraphs is taken as the solution for the problem on the original graph.

The maximum edge-disjoint paths problem has been known to be NP-hard in directed graphs with only two terminal pairs [58]. It can also be approximated with ratio $O(\sqrt{m})$ on directed graphs $D = (V, A)$ using a simple greedy algorithm [81]. The ratio was refined to $O(\min\{n^{2/3}, m^{1/2}\})$ in [40]. On the negative side, the MEDP problem is NP-hard to approximate in directed graphs within a factor better than $\Omega(m^{\frac{1}{2} - \epsilon})$ unless $P = NP$ [67].

The weighted MEDP problem in undirected stars is equivalent to the maximum weight matching problem in general graphs, and can be solved optimally [63]. The weighted MEDP problem in undirected trees can also be solved optimally, using the maximum weight matching algorithm as a subroutine [63]. The MEDP problem is MAX SNP-hard in directed trees, and the reduction is from the maximum three-dimensional matching problem. The problem can be solved in polynomial time for directed stars and spiders. For bounded degree directed trees, the problem can also be solved in polynomial time, using a dynamic programming approach [46]. For directed trees with arbitrary degree, the problem can be approximated with ratio $\frac{5}{3} + \epsilon$, for every fixed $\epsilon > 0$ [49]. The algorithm consists of one bottom-up pass followed by a top-down pass. The reason for using two passes is that some paths have to be in an intermediate state during the first pass. The weighted MEDP problem in directed trees cannot be solved this way, but an algorithm with approximation ratio $\frac{5}{3} + \epsilon$ is given in [51], using a completely different method.

The MEDP problem in undirected rings can be solved in polynomial time as follows [120]. Let $e$ be an edge in the ring, and $R$ be the set of routing requests. In any set of edge disjoint paths, either no path is on $e$ or exactly one path uses $e$. For the latter case, we can try every request $(s, t) \in R$. We then delete all edges used by $(s, t)$ from the ring, and solve the MEDP problem in the obtained undirected chain in polynomial time. Similarly, the
MEDP in directed rings, and the MEDP problem with pre-specified paths in undirected and directed rings can also be solved in polynomial time. The weighted MEDP problem (and the fixed paths version) can be solved in polynomial time in undirected and directed rings using a similar approach.

The MEDP problem, whether paths are fixed or not, is MAX SNP-hard in directed and undirected trees of rings [47]. When paths are not fixed, there is a 3-approximation algorithm for undirected trees of rings, and a $5+\epsilon$-approximation algorithm for directed trees of rings. When paths are fixed, there are 4-approximation and 8-approximation algorithms for undirected and directed trees of rings, respectively. The weighted MEDP and MEDPwPP problems in trees of rings can be approximated within the same ratio as in the unweighted cases.
Chapter 3

Path Coloring on Trees of Rings

This chapter studies the minimum path coloring (Min-PC) problem on trees of rings. In Section 3.1, we first give a polynomial time algorithm which uses at most $3L$ colors for the problem on trees of rings with arbitrary degrees (recall that $L$ is the maximum number of paths on any link in the network). This improves the previous $4L$ and 4-approximation result of [47]. The $3L$ upper bound is tight since there are instances of the Min-PC problem that require $3L$ colors even on a tree of rings with degree four. Our algorithm is based on a processing order proposed in [47] and novel applications of edge-coloring of multigraphs. In Section 3.2, we give two efficient path coloring schemes for trees of rings and some useful facts on edge-coloring of multigraphs. Based on these techniques, in Section 3.3, we show that the $3L$ algorithm achieves an approximation ratio of 2.75 asymptotically for the Min-PC problem on trees of rings with arbitrary degrees. In Section 3.4, we further give a $3L$ and 2-approximation (resp. 2.5-approximation) algorithm for the Min-PC problem on trees of rings with degree at most six (resp. eight and ten). The algorithms on trees of rings with bounded degrees are of independent interest and improve the previous 2-approximation algorithm for trees of rings with degree four [25, 44], and 2.5-approximation algorithm for trees of rings with degree six [25]. Our $3L$ result also implies a 3-approximation algorithm for the Min-RPC problem on trees of rings. The algorithm does not use the cut-one-link approach, and gives an alternative approach for solving the Min-RPC problem. This approach might provide a better fault-tolerance than the cut-one-link approach. Our $3L$ algorithm implies a $6L$ algorithm for the Min-PC problem on directed trees of rings.
3.1 The 3L Upper Bound

In this section, we give an algorithm which solves the Min-PC problem in trees of rings with arbitrary degrees using at most 3L colors. We start with some more definitions. A tree of rings network is denoted by a graph $TR$ with node set $V(TR)$ and link set $E(TR)$. For $TR$, we have the following property.

Proposition 3.1.1 For any node $u \in V(TR)$, a path on $u$ can be on at most two rings which contain $u$.

For a node $u$ in a ring of $TR$, we denote $u^-$ as the neighbor of $u$ in the counter-clockwise direction and $u^+$ as the neighbor of $u$ in the clockwise direction in the ring (see (a) of Figure 3.1).

We say a set of elements is assigned distinct colors if for any two different elements in the set, the elements are assigned different colors. Throughout this chapter, we will denote $W_P$ as the set of colors assigned to a set $P$ of paths, and denote $W_{uv}$ as the set of colors assigned to the paths on a link $(u,v)$ of $TR$.

The 3L algorithm.

We give a simple algorithm, called ALG3.1, which uses at most 3L colors for the Min-PC problem on $TR$ with an arbitrary degree and show that the 3L upper bound is tight. We
CHAPTER 3. PATH COLORING ON TREES OF RINGS

Procedure Framework\((TR,P)\)

**Input:** A set \(P\) of paths in \(TR\).

**Output:** A valid coloring from \(W = \{\lambda_1, \lambda_2, \ldots\}\) to \(P\).

begin
1. Fix a DFS (depth-first search) order, starting from a node (say \(u_0\)) of degree two, on the nodes of \(TR\).
2. Process the starting node \(u_0\).
3. Process the other nodes \(u\) in the DFS order.
   
   Let \(r_0\) be the ring which contains \(u\) and the parent of \(u\).
   
   3.1 Color the set \(P_0\) of uncolored paths on \(u\) and \(r_0\).
   
   3.2 Color the set \(P_1\) of other uncolored paths on \(u\).

end.

Figure 3.2: A framework of algorithms for the Min-PC problem on trees of rings.

first give a framework in Figure 3.2 for all algorithms in this chapter. Paths are colored in some order defined later. At any stage of the coloring procedure, a path is called *colored* if it has been assigned a color, otherwise *uncolored*. In the algorithm, processing a node \(u\) means coloring the uncolored paths on \(u\). We call a node \(u\) *processed* if the coloring process for \(u\) has been completed, otherwise *unprocessed*. Notice that before the coloring process for node \(u\), some paths on \(u\) may have been colored due to the processing of other nodes. Node \(u\) is still called unprocessed if the coloring process for \(u\) has not been performed even all paths on \(u\) have been colored due to the processing of other nodes. Our algorithm processes the nodes of \(TR\) in the DFS (depth-first search) order proposed in [47]. For a node \(u\), its *parent* is the node from which \(u\) is reached in the DFS order (see (b) of Figure 3.1). A link is called *special* if it connects a processed node and an unprocessed node (see (b) of Figure 3.1). There are either 0 or 2 special links in a ring in \(TR\). A path on a special link is colored and only such a path has a possibility to intersect with an uncolored path. We assume that in Step 1, the nodes in the same ring are searched in the clockwise direction in the DFS order. Notice that a node of degree two always exists in a finite \(TR\).

The steps of Algorithm ALG3.1 are given in the framework in Figure 3.2. In Step 2, we first assign colors of \(W\) to the paths on link \((u_0, u_0^-)\) by the first-fit coloring. Next we assign the uncolored paths on link \((u_0, u_0^+)\) the colors of \(W \setminus W_{u_0u_0^-}\) by the first-fit coloring.
In Step 3, the parent of node $u$ in the DFS order is node $u^-$ in some ring which is called $r_0$. If $u$ appears in $k + 1$ rings, the other $k$ rings are denoted by $r_i$, $1 \leq i \leq k$ (see (b) of Figure 3.1). Let $Q_0$ be the set of paths on special links $(u, u^-)$ or $(w, w^-)$. In Step 3.1, we color $P_0$ using the colors of $W \setminus W_{Q_0}$ by the first-fit coloring.

In Step 3.2, we convert the path coloring problem to the edge-coloring problem of a multigraph $G_u$ with rings $r_i$ ($0 \leq i \leq k$) as vertices and all paths on $u$ as edges. By Proposition 3.1.1, a path on $u$ is on either one ring or two rings. A path on $u$ is called a long path if it is on two rings, otherwise a short path (see (b) of Figure 3.1). To eliminate self-loops, we introduce a vertex $s_i$ for every $r_i$ in $G_u$. More specifically, $G_u$ is defined as:

$V(G_u) = \{r_i, s_i \mid 0 \leq i \leq k\}$,

$E(G_u) = \{(r_i, r_j, p) \mid p$ is a long path on $r_i$ and $r_j, 0 \leq i < j \leq k\}$

$\cup \{(r_i, s_i, p) \mid p$ is a short path on $u$ and $r_i, 0 \leq i \leq k\}$,

where $(x, y, p)$ denotes an edge between vertices $x$ and $y$ with label $p$. From Proposition 3.1.1, there is a one-to-one correspondence between the paths on $u$ and the edges in $G_u$. Assume that a valid edge-coloring for $G_u$ has been found and let $C_{G_u} = \{\mu_1, \mu_2, \ldots\}$ be the set of virtual colors used for the edge-coloring. We use the mapping $f_1 : C_{G_u} \rightarrow W$ defined below to color the corresponding paths on $u$. Let $Q_1$ be the set of colored paths on $u$ before Step 3.2 and $C_{Q_1}$ be the set of virtual colors assigned to the edges $(x, y, p)$ of $G_u$ with $p \in Q_1$. The mapping $f_1$ is defined as follows:

1. For each $\mu_i \in C_{Q_1}$ assigned to edge $(x, y, p)$ with $p \in Q_1$, $f_1(\mu_i) = \lambda_j$, where $\lambda_j \in W_{Q_1}$ is the color assigned to path $p$ before Step 3.2.

2. For each $\mu_i \in C_{G_u} \setminus C_{Q_1}$, $f_1$ maps $\mu_i$ to a $\lambda_j \in W \setminus W_{Q_1}$ with the smallest available index $j$ such that $C_{G_u} \setminus C_{Q_1}$ is assigned distinct colors.

Since all paths of $Q_1$ are on ring $r_0$, edges $(x, y, p)$ with $p \in Q_1$ are given distinct virtual colors in the edge-coloring of $G_u$. From this and the above definition, $f_1$ is a function from $C_{G_u}$ to $W$ which implies that for any two edges (in $G_u$), the corresponding paths are colored by the same real color if and only if these two edges are colored by the same virtual color. Also, $f_1$ does not change the colors of the paths which were colored before Step 3.2.

To apply the edge-coloring of $G_u$ in Step 3.2 as shown above, it is required that $Q_1$ has been assigned distinct colors. In other words, no color has been given to more than one
path in $Q_1$. A set of colored paths is called a \textit{0-set} if the paths are assigned distinct colors. We say the 0-set condition is true on a ring if the set of paths on special links of the ring is a 0-set. As shown later, the 0-set condition is kept true for each ring of $TR$ in Algorithm ALG3.1. This is critical in applying the edge-coloring of $G_u$ in Step 3.2.

Algorithm ALG3.1 colors the paths step by step. In each step, there are a set of colored paths and a set of paths to be colored. The following lemma is useful to get the total number of colors from that used for coloring in each step.

\textbf{Lemma 3.1.2} Given a set $Q$ of colored paths and a set $R$ of uncolored paths, assume that at most $w$ colors have been used for $Q$, and that a subset $Q'$ of $Q$ contains every colored path intersecting with a path of $R$. If an algorithm colors $R$ such that the coloring for $R$ and the previous coloring for $Q'$ give a valid coloring for $Q' \cup R$ using at most $w$ colors, then $Q \cup R$ can be colored with at most $w$ colors.

\textbf{Proof:} From the condition of the lemma, $\left| W_R \setminus W_Q' \right| = \left| W_R \cup W_Q' \right| - \left| W_Q' \right| \leq w - \left| W_Q' \right|$. Since each path of $Q \setminus Q'$ is edge-disjoint with any path of $R$, all colors of $W_Q \setminus W_Q'$ can be used as the colors for $R$. Therefore, $Q \cup R$ can be colored with at most $w$ colors. \hfill \Box

By the above lemma, if an algorithm colors $R$ such that at most $w$ colors are used for $Q' \cup R$ in every step, it solves the Min-PC problem with at most $w$ colors. In what follows, we only analyze the number of colors used in each step for $Q' \cup R$. Let $Q_u$ be the set of colored paths and $P_u$ be the set of paths to be colored when node $u$ is being processed in Algorithm ALG3.1.

\textbf{Theorem 3.1.3} Algorithm ALG3.1 solves the Min-PC problem on $TR$ with $n$ nodes and degree $2(k + 1)$ using at most $3L$ colors in $O(nkL(k + L))$ time.

\textbf{Proof:} In Step 2, since there are at most $L$ paths on any link of $TR$, there are at most $2L$ paths on $u_0$. Therefore, the paths on $u_0$ can be colored with at most $2L$ colors. Since each path is given a distinct color, the 0-set condition is true for every ring of $TR$ after this step. We now show that Algorithm ALG3.1 colors $Q_u \cup P_u$ using at most $3L$ colors for every node $u$ in $TR$.

In Step 3, assume that $Q_u$ has been colored with at most $3L$ colors and the 0-set condition is true for every ring of $TR$. In Step 3.1, $|P_0| \leq L$ because each path in $P_0$ is on link $(u, u^+)$. Therefore, $P_0$ can be colored with at most $L$ colors. Since $Q_0$ defined for
CHAPTER 3. PATH COLORING ON TREES OF RINGS

Step 3.1 of Algorithm ALG3.1 contains every colored path intersecting with a path of $P_0$ and $|Q_0| \leq 2L$, $Q_0 \cup P_0$ (thus $Q_u \cup P_0$ by Lemma 3.1.2) can be colored with most $3L$ colors. Since $Q_0 \cup P_0$ is assigned distinct colors, after Step 3.1 the 0-set condition is true for $r_0$ and $Q_1$ defined for Step 3.2 of Algorithm ALG3.1 is a 0-set. The latter is critical in Step 3.2.

In Step 3.2, $Q_1$ contains every colored path intersecting with a path of $P_1$. By the edge-coloring of $G_u$, the definition of mapping $f_1$, and the 0-set condition of $Q_1$, the set of paths on ring $r_i$ and node $u$ is assigned distinct colors. Also $f_1$ does not change the color of any path in $Q_1$. Therefore, $f_1$ colors $P_1$ such that the colorings for $P_1$ and $Q_1$ give a valid coloring for $Q_1 \cup P_1$. The number of colors used for $Q_1 \cup P_1$ is $|C_{G_u}|$. Since there are at most $L$ paths on any link of $TR$, there are at most $2L$ paths on node $u$ and any ring $r_i$. Therefore, $\Delta(G_u) \leq 2L$. By Proposition 2.1.1, a valid edge-coloring of $G_u$ can be found using $|C_{G_u}| \leq 3L$ colors. Thus, at most $3L$ colors are used for $Q_1 \cup P_1$, implying at most $3L$ colors for $Q_u \cup P_u$. Since the set of paths on $u$ and any ring $r_i$ is assigned distinct colors, the 0-set condition holds for every ring.

Summarizing the above, the algorithm solves the Min-PC problem on $TR$ using at most $3L$ colors. The edge-coloring of multigraph $G_u$ is the dominant part in Algorithm ALG3.1 for the time complexity. Since $\Delta(G_u) \leq 2L$, $|V(G_u)| \leq 2(k + 1)$, and $|E(G_u)| = O(kL)$, by Proposition 2.1.1, the edge-coloring of $G_u$ takes $O(kL(k + L))$ time. Since Algorithm ALG3.1 executes Steps 3.1 and 3.2 $O(n)$ times, the time complexity of the algorithm is $O(nkL(k + L))$.

It is known that there are instances which require $3L$ colors for the Min-PC problem on trees of rings [25]. This lower bound implies that in the worst case one cannot do better than $3L$ even for trees of rings with node degree four. Algorithm ALG3.1 achieves the $3L$ tight upper bound for trees of rings with arbitrary degrees. Since $L$ is a lower bound on the number of colors for any optimal solution, Algorithm ALG3.1 achieves an approximation ratio of 3 for the Min-PC problem on $TR$ with an arbitrary degree. The algorithm can be used to obtain a 3-approximation algorithm for the Min-RPC problem on trees of rings as following. First, for a given set of connection requests, a path for each request can be found efficiently such that $L$ is minimized [47]. Then, the set of found paths is colored by Algorithm ALG3.1 using at most $3L$ colors. Since the load $L$ is optimal, it is also a lower bound for the original Min-RPC problem. In this way, the approximation ratio of 3 is achieved without using the cut-one-link approach. Our $3L$ algorithm also implies a
6L algorithm for the Min-PC problem on directed trees of rings with two directed links, one in each direction, between a pair of adjacent nodes. It is interesting to improve the approximation ratio for the Min-PC problem on directed trees of rings.

3.2 Preparation for Improvement

In Algorithm ALG3.1, the 0-set condition is kept for every ring for edge-coloring \( G_u \) in Step 3.2 in a straightforward way. One observation is that the 0-set condition may be too strict for solving the Min-PC problem on \( TR \) since two paths on special links of a ring can have the same color if they are edge-disjoint. Better approximation ratios may be achieved if the 0-set condition is relaxed. Another observation is that we may use less colors for the edge-coloring of multigraph \( G_u \) if a more advanced algorithm like that in [95] is used. In this section, we first give two schemes for coloring paths on trees of rings with the 0-set condition relaxed. The path coloring schemes make more efficient use of colors. Then we show some properties of multigraph \( G_u \) related to its edge-coloring. The path coloring schemes and properties of \( G_u \) will be used in the following sections to get algorithms with better approximation ratios.

3.2.1 Efficient Path Coloring Schemes

We first introduce the notion of \( \beta \)-set which is an extension of 0-set. A color for a set of colored paths is called a multi-color if the color has been assigned to two paths in the set. For an integer \( \beta \geq 0 \), a set of colored paths is called a \( \beta \)-set if each color is assigned to at most two paths, the paths with the same color are edge-disjoint, and the number of multi-colors for the path set is at most \( \beta \). We say the \( \beta \)-set condition is true on a ring if the set of paths on special links of the ring is a \( \beta \)-set. For any given integer \( \beta \) with \( 0 \leq \beta \leq L \), the schemes given below use as few colors as possible to keep the \( \beta \)-set condition for every ring.

Recall that \( P_0 \) and \( P_1 \) are the sets of paths to be colored in Step 3.1 and Step 3.2 of the framework in Figure 3.2, respectively. We give a scheme for coloring \( P_0 \) and a scheme for coloring a subset of \( P_1 \). The scheme for \( P_0 \), called S31, works as follows. Assume that the \( \beta \)-set condition is true for every ring before \( P_0 \) is colored. Recall that \( Q_0 \) is the set of paths on special links \((u, u^-)\) or \((w, w^-)\). Let \( W_{Q_0}^m \subseteq W_{Q_0} \) be the set of multi-colors for \( Q_0 \). Then from the \( \beta \)-set condition, \( |W_{Q_0}^m| \leq \beta \). Define \( A_0 \) (resp. \( B_0 \)) to be the set of paths on link
Figure 3.3: The sets of paths related to Schemes S31 and S32.

(a) $(u, u^-)$ (resp. on $(w, w^-)$), each of which has a color in $W_{Q_0} \setminus W_{Q_0}$ (see (a) of Figure 3.3). Then $|A_0| + |W_{Q_0}^m| \leq L$, $|B_0| + |W_{Q_0}^m| \leq L$, and $A_0 \cup B_0$ is assigned distinct colors. We construct a graph $G_0$ with

$$V(G_0) = P_0 \cup A_0 \text{ and } E(G_0) = \{(p, q) \mid p \text{ and } q \text{ are edge-disjoint}\}.$$ 

We find a maximum matching $M_0$ of $G_0$. Notice that $G_0$ is bipartite and for each pair $(p, q) \in M_0$, $p \in P_0$ and $q \in A_0$. We select $\min\{|M_0|, \beta - |W_{Q_0}^m|\}$ pairs from $M_0$. For each selected pair $(p, q)$, assign the color of $q \in A_0$ to $p$. We assign the remaining paths of $P_0$ the colors of $W \setminus W_{Q_0}$ by the first-fit coloring. As shown later, the $\beta$-set condition is true for every ring after $P_0$ is colored.

The second scheme, called S32, is used to color a subset of $P_1$. More specifically, S32 is used to color the long paths on rings $r_i$ and $r_j$ ($i, j \neq 0, i \neq j$) subject to the condition that every colored path on $r_i$ or $r_j$ is also on $r_0$ when S32 is called. Without loss of generality, we assume that $r_i = r_1$ and $r_j = r_2$ for simplicity. Let $P_{12} \subseteq P_1$ be the set of long paths on rings $r_1$ and $r_2$ (see (b) of Figure 3.3). Recall that $Q_1$ is the set of colored paths on $u$ before Step 3.2. Then every path of $Q_1$ is on $r_0$. Let $Q_1' \subseteq Q_1$ be the set of colored long paths on links $(u, u^-)$ or $(u, u^+)$ and on rings $r_1$ or $r_2$. We define $W_{Q_1'}^m \subseteq W_{Q_1'}^m$ to be the set of multi-colors for the set $Q_1'$. From the $\beta$-set condition, $|W_{Q_1'}^m| \leq \beta$. Define $A_1$ (resp. $B_1$) to be the set of long paths on link $(u, u^-)$ (resp. on $(u, u^+)$) and on rings $r_1$ or $r_2,$
each of which has a color in $W^{m}_{Q'} \setminus W^{m}_{Q}$ (see (b) of Figure 3.3). Then $|A_1| + |W^{m}_{Q'}| \leq L$, $|B_1| + |W^{m}_{Q'}| \leq L$, and $A_1 \cup B_1$ is assigned distinct colors. We construct a graph $G_1$ with

$$V(G_1) = P_{12} \cup A_1$$

and

$$E(G_1) = \{(p, q) \mid p \text{ and } q \text{ are edge-disjoint}\}.$$  

We find a maximum matching $M_1$ of $G_1$. For each pair $(p, q) \in M_1$, either $p \in P_{12}$ and $q \in A_1$ or $p, q \in P_{12}$. We select $\min\{|M_1|, \beta - |W^{m}_{Q'}|\}$ pairs from $M_1$. For each selected pair $(p, q)$ with $q \in A_1$, assign the color of $q$ to $p$. For each selected pair $(p, q)$ with $p, q \in P_{12}$, assign the pair a distinct color from $W \setminus W^{m}_{Q'}$ by the first-fit coloring ($p$ and $q$ share the same color, but different pairs are given different colors). Let $W'$ be the set of colors assigned to the selected pairs $(p, q)$ with $p, q \in P_{12}$. We assign each of the remaining paths of $P_{12}$ a color from $W \setminus (W^{m}_{Q'} \cup W')$ by the first-fit coloring such that the set of the remaining paths of $P_{12}$ is assigned distinct colors.

Let $OPT_0$ (resp. $OPT_1$) be the number of colors required to color $P_0 \cup A_0$ (resp. $P_{12} \cup A_1$) in an optimal solution.

**Lemma 3.2.1** $OPT_0 = |P_0| + |A_0| - |M_0| \text{ and } OPT_1 = |P_{12}| + |A_1| - |M_1|.$

**Proof:** It is easy to see that $|P_0| + |A_0| - |M_0|$ colors are sufficient for $P_0 \cup A_0$. We prove that $|P_0| + |A_0| - |M_0|$ colors are also necessary. Notice that each color is used by at most two paths in $P_0 \cup A_0$. Assume to the contrary that there is a valid coloring which uses $OPT' = k_1 + k_2 < |P_0| + |A_1| - |M_0|$ colors, where each of the $k_1$ colors is used by one path and each of the $k_2$ colors is used by two paths. Since $M_0$ is a maximum matching, $k_2 \leq |M_0|$. The total number of paths colored by $OPT'$ is

$$|P_0| + |A_0| = k_1 + 2k_2 \leq k_1 + k_2 + |M_0| < |P_0| + |A_0| - |M_0| + |M_0| = |P_0| + |A_0|,$$

a contradiction.

The proof for $OPT_1 = |P_{12}| + |A_1| - |M_1|$ is similar, noting that a color is used for at most two paths in $P_{12} \cup A_1$. \hfill \(\square\)

**Lemma 3.2.2** Scheme S31 colors $P_0$ such that the colorings for $Q_0$ and $P_0$ give a valid coloring for $Q_0 \cup P_0$ using at least $|Q_0 \cup P_0| - \beta$ and at most $\min\{|Q_0 \cup P_0|, \max\{|Q_0 \cup P_0| - \beta, OPT_0 + L\}\}$ colors. Furthermore, $Q_0 \cup P_0$ is a $\beta$-set.

**Proof:** Clearly, at most $|Q_0 \cup P_0|$ colors are used, paths with the same color are edge-disjoint, and a color is used for at most two paths in $Q_0 \cup P_0$. If $|M_0| > \beta - |W^{m}_{Q'}|$ then there
are exactly $\beta$ multi-colors for $Q_0 \cup P_0$ and $|Q_0 \cup P_0| - \beta$ colors are used. Otherwise, there are at most $\beta$ multi-colors for $Q_0 \cup P_0$ and at least $|Q_0 \cup P_0| - \beta$ colors are used. In the latter case, the number of colors used is $|A_0| + |B_0| + |W_{Q_0}^m| + |P_0| - |M_0|$. From Lemma 3.2.1 and $|B_0| + |W_{Q_0}^m| \leq L$, at most $OPT_0 + L$ colors are used.

**Lemma 3.2.3** Scheme $S3^2$ colors $P_{12}$ such that the colorings for $Q'$ and $P_{12}$ give a valid coloring for $Q' \cup P_{12}$ using at least $|Q' \cup P_{12}| - \beta$ and at most $\min\{|Q' \cup P_{12}|, \max\{|Q' \cup P_{12}| - \beta, OPT_1 + L\}\}$ colors. Furthermore, $Q' \cup P_{12}$ is a $\beta$-set.

**Proof:** Clearly, at most $|Q' \cup P_{12}|$ colors are used, paths with the same color are edge-disjoint, and each color is used for at most two paths in $Q' \cup P_{12}$. If $|M_1| > \beta - |W_{Q_1}^m|$ then there are exactly $\beta$ multi-colors for $Q' \cup P_{12}$ and $|Q' \cup P_{12}| - \beta$ colors are used. Otherwise, there are at most $\beta$ multi-colors for $Q' \cup P_{12}$ and at least $|Q' \cup P_{12}| - \beta$ colors are used. In the latter case, the number of colors used is $|A_1| + |B_1| + |W_{Q_1}^m| + |P_{12}| - |M_1|$. By Lemma 3.2.1 and $|B_1| + |W_{Q_1}^m| \leq L$, at most $OPT_1 + L$ colors are used. \qed

### 3.2.2 Edge-coloring of Multigraphs

For a (multi)graph $G$, $l(G)$ defined in Proposition 2.1.2 is a lower bound on the number of colors for the edge-coloring of $G$. The multigraph $G_u$ constructed in Step 3.2 of Algorithm ALG3.1 has maximum degree $2L$, and $l(G_u)$ can be as large as $3L$. Thus, a direct application of more advanced edge-coloring algorithms (such as that of \cite{95}) in Step 3.2 of Algorithm ALG3.1 cannot improve the approximation ratio. In this subsection, we show some properties of $G_u$ when $l(G_u)$ is large. These properties, Schemes $S31$ and $S32$, and the application of a more advanced edge-coloring algorithm in Step 3.2 will be used to improve the approximation ratio of Algorithm ALG3.1.

**Lemma 3.2.4** For any subgraph $H$ of $G_u$, if $L(H) > [2.5L]$ then $|V(H)| = 3$.

**Proof:** If $L(H) > [2.5L]$, then clearly $|V(H)| \geq 3$. Therefore, it suffices to show that $L(H) \leq [2.5L]$ if $|V(H)| > 3$. Consider two cases. If $|V(H)| = 2j$ ($j \geq 2$),

$$L(H) = \left[ \left\lfloor \frac{|E(H)|}{|V(H)|/2} \right\rfloor \right] = \left[ \frac{|E(H)|}{j} \right] \leq \left[ \sum_{u \in V(H)} d(u) \right] \frac{|V(H)|/2}{2j} \leq \left[ \frac{|V(H)| \times 2L}{2j} \right] = 2L.$$
CHAPTER 3. PATH COLORING ON TREES OF RINGS

If $|V(H)| = 2j + 1$ ($j \geq 2$),

$$L(H) = \left\lfloor \frac{|E(H)|}{|V(H)|/2} \right\rfloor = \left\lfloor \frac{|E(H)|}{j} \right\rfloor \leq \left\lfloor \frac{\sum_{u \in V(H)} d(u)}{2j} \right\rfloor \leq \left\lfloor \frac{|V(H)| \times 2L}{2j} \right\rfloor = \left\lfloor (1 + \frac{1}{2j}) \times 2L \right\rfloor \leq \lceil 2.5L \rceil.$$  

Lemma 3.2.5 For subgraphs $H_1$ and $H_2$ of $G_u$ with $L(H_1) > \lceil 2.5L \rceil$ and $L(H_2) > \lceil 2.5L \rceil$, $V(H_1) \cap V(H_2) = \emptyset$.

Proof: To prove the lemma by contradiction, assume that $V(H_1) \cap V(H_2) \neq \emptyset$. By Lemma 3.2.4, both $H_1$ and $H_2$ have three vertices. There are two cases to consider: $|V(H_1) \cap V(H_2)| = 1$ and $|V(H_1) \cap V(H_2)| = 2$. For the first case, the total number of edges in $H_1 \cup H_2$ is

$$|E(H_1 \cup H_2)| = |E(H_1)| + |E(H_2)| > \lceil 2.5L \rceil + \lceil 2.5L \rceil \geq 5L.$$  

However,

$$|E(H_1 \cup H_2)| \leq \frac{\sum_{u \in (V(H_1) \cup V(H_2))} d(u)}{2} \leq \frac{5 \times 2L}{2} = 5L,$$

a contradiction.

For the second case, assume that $V(H_1) \cap V(H_2) = \{r_a, r_b\}$. Let $m(r_a, r_b)$ be the number of multi-edges between $r_a$ and $r_b$. The total number of edges in $H_1 \cup H_2$ is

$$|E(H_1 \cup H_2)| = |E(H_1)| + |E(H_2)| - m(r_a, r_b) > 5L - m(r_a, r_b).$$  

However,

$$|E(H_1 \cup H_2)| \leq d(r_a) + d(r_b) - m(r_a, r_b) \leq 4L - m(r_a, r_b),$$

a contradiction.

For multigraph $G_u$, let $F_u$ be the graph obtained by contracting each subgraph $H$ of $G_u$ with $L(H) > \lceil 2.5L \rceil$ into a single vertex $v_H$. More precisely, let $V' = \cup_{H: L(H) > \lceil 2.5L \rceil} V(H)$ and $E' = \cup_{H: L(H) > \lceil 2.5L \rceil} E(H)$. Graph $F_u$ is defined by

$$V(F_u) = \{v_H | L(H) > \lceil 2.5L \rceil\} \cup (V(G_u) \setminus V')$$  

and

$$E(F_u) = E(G_u) \setminus E'.$$
CHAPTER 3. PATH COLORING ON TREES OF RINGS

Figure 3.4: A multigraph $G_u$ and its contracted graph $F_u$.

where for each edge in $E(G_u) \setminus E'$, if an end vertex of the edge is in $V(H)$, the end vertex is replaced by $v_H$ in $E(F_u)$. From Lemma 3.2.5, $v_{H_1} \neq v_{H_2}$ for $H_1 \neq H_2$. We call $F_u$ the contracted graph of $G_u$. Figure 3.4 gives an example of $G_u$ and $F_u$.

**Lemma 3.2.6** \( l(F_u) \leq [2.5L] \) and \( d(v_H) < L \).

**Proof:** The degree of $v_H$ in graph $F_u$ is

\[
\sum_{u \in V(H)} d(u) - 2 \times |E(H)| < |V(H)| \times 2L - 2 \times [2.5L] \leq 3 \times 2L - 5L = L.
\]

After every subgraph $H$ with $L(H) > [2.5L]$ is contracted to a vertex $v_H$ in $F_u$, from $d(v_H) < L$, any subgraph of $F_u$ with three vertices including $v_H$ has at most \( \left\lfloor \frac{d(v_H) + 2L + 2L}{2} \right\rfloor < [2.5L] \) edges. Therefore, by Lemma 3.2.4, \( l(F_u) \leq [2.5L] \).

### 3.3 A 2.75-approximation Algorithm

Applying the results of the previous section, we show a better approximation algorithm for the Min-PC problem on TR with an arbitrary degree. By Proposition 2.1.2, the edge-coloring of $G_u$ can be done with at most $[2.5L]$ colors if $l(G_u) \leq [2.5L]$. On the other hand, if $l(G_u) > [2.5L]$, we can contract $G_u$ into $F_u$ with $l(F_u) \leq [2.5L]$ (Lemma 3.2.6) and then apply the edge-coloring algorithm of [95] to $F_u$. Each contracted subgraph $H$ has three vertices, corresponding to three rings containing node $u$, and the paths corresponding
to the edges in $H$ can be colored by Schemes S31 and S32, with a properly chosen integer $\beta$. For simplicity, in what follows, we sometimes refer to the edges in the multigraph $G_u$ and the corresponding paths on node $u$ of $TR$ without distinguishing them, if there is no confusion.

Our algorithm, called ALG3.2, follows the framework in Figure 3.2. Step 2 of ALG3.2 is the same as that in Algorithm ALG3.1. Step 3.1 uses Scheme S31. By Lemma 3.2.2, $|Q_0 \cup P_0| \leq 3L$, and $OPT_0 \leq OPT$, at most $\min\{3L, \max\{3L - \beta, OPT + L\}\}$ colors are used for $Q_0 \cup P_0$. In Step 3.2, we color $P_1$. Similar to Algorithm ALG3.1, we convert the path coloring problem to the edge-coloring problem of multigraph $G_u$, but we use the algorithm of [95] to solve the edge-coloring problem. There are two cases.

**Case 1:** $l(G_u) \leq \lceil 2.5L \rceil$.

We apply the algorithm of [95] to $G_u$. Since Scheme S31 is used for Step 3.1, $Q_1$ is a $\beta$-set. If Scheme S31 is used with a $\beta > 0$, two paths of $Q_1$ may have been colored by the same multi-color from $W_{Q_1}$. To get a valid coloring from $W$ to the paths of $G_u$, for each pair of paths $p, q \in Q_1$ with the same multi-color from $W_{Q_1}$, we re-assign a new virtual color $\mu_{pq} \notin C_{G_u}$ to $p$ and $q$. Let $C'_{G_u}$ (resp. $C'_{Q_1}$) be the set of virtual colors assigned to the paths of $G_u$ (resp. $Q_1$) after the re-assignment. We map $C'_{G_u}$ to $W$ by mapping $f_i$ defined in Section 3.1 to get a valid coloring from $W$ to the paths of $G_u$. More specifically, we perform the following:

1. For each $\mu_i \in C'_{Q_1}$ assigned to edge $(x, y, p)$ with $p \in Q_1$, $f_1(\mu_i) = \lambda_j$, where $\lambda_j \in W_{Q_1}$ is the color assigned to path $p$ before Step 3.2.

2. For each $\mu_i \in C'_{G_u} \setminus C'_{Q_1}$, $f_1$ maps $\mu_i$ to a $\lambda_j \in W \setminus W_{Q_1}$ with the smallest available index $j$ such that $C'_{G_u} \setminus C'_{Q_1}$ is assigned distinct colors.

Since $Q_1$ is a $\beta$-set, $|C'_{G_u}| \leq |C_{G_u}| + \beta$. Also notice that $\Delta(G_u) \leq 2L$, and $[(11\Delta(G_u) + 8)/10] \leq [2.2L + 0.8] \leq [2.5L]$ for any positive integer $L$. From Proposition 2.1.2 and $l(G_u) \leq \lceil 2.5L \rceil$, the valid coloring uses at most $\lceil 2.5L \rceil + \beta$ colors. This suggests a small $\beta$. However, the upper bound $\min\{3L, \max\{3L - \beta, OPT + L\}\}$ in Step 3.1 suggests a large $\beta$. To minimize $\max\{\lceil 2.5L \rceil + \beta, 3L - \beta\}$, we choose $\beta = \lceil 0.25L \rceil$ for Scheme S31 in Step 3.1. Notice that $\lceil 2.5L \rceil + \lceil 0.25L \rceil \leq 3L - \lceil 0.25L \rceil = \lceil 2.75L \rceil$.

**Case 2:** $l(G_u) > \lceil 2.5L \rceil$.

From Lemma 3.2.4, there is at least one subgraph $H$ of $G_u$ with $L(H) > \lceil 2.5L \rceil$ and $|V(H)| = 3$. There are two subcases depending on whether ring $r_0$ is a vertex of some $H$ or
CHAPTER 3. PATH COLORING ON TREES OF RINGS

Case 2.1: Ring $r_0$ is not a vertex of any subgraph $H$ of $G_u$ with $L(H) > [2.5L]$.

We contract $G_u$ to $F_u$. From Lemma 3.2.6, $l(F_u) \leq [2.5L]$. We then apply the algorithm of [95] to $F_u$ and get a valid coloring for the paths corresponding to the edges of $F_u$ by the mapping $f_1$ as we did in Case 1. After this we color the paths corresponding to the edges in each contracted subgraph $H$ by virtual colors of $C_H = \{\nu_1, \nu_2, \ldots\}$, using Scheme S32 as a subroutine (the details will be given shortly). Notice that some paths between ring $r_0$ and a ring of $H$ may have been colored by multi-colors. Because those multi-colors are also multi-colors for the paths on a ring of $H$, we need to subtract the number of those multi-colors from $[0.25L]$ to get $\beta$ for Scheme S32 to keep the $[0.25L]$-set condition for each ring. We need some more definitions to formally define the $\beta$ for Scheme S32.

Assume $V(H) = \{r_a, r_b, r_c\}$. We use $P_{ij}$ ($i, j = a, b, c; i \neq j$) for the set of long paths in $H$ on $r_i$ and $r_j$, and use $R_i$ ($i = a, b, c$) for the set of paths not in $H$ but on $r_i$ (see (a) of Figure 3.5). Notice that $R_a \cup R_b \cup R_c$ has been colored and contains every colored path intersecting with a path of $P_{ab} \cup P_{ac} \cup P_{bc}$. Let $Q' = R_a \cup R_b \cup R_c$ and $W_{Q'}^m$ be the set of multi-colors for $Q'$. Since paths with a color from $W_{Q'}^m$ are on $r_0$, from the $[0.25L]$-set condition on $r_0$, $|W_{Q'}^m| \leq [0.25L]$.

For any two paths $p$ and $q$ with a multi-color $\lambda_m \in W_{Q'}^m$, there are two cases. Case (i), $p$ and $q$ are on $r_0$ and a single ring of $H$ (say $r_a$, the dashed edges in (a) of Figure 3.5). Case (ii), $p$ and $q$ are on $r_0$ and two rings of $H$ (say $r_a$ and $r_c$, the dotted edges in (a) of Figure 3.5). In Case (i), $\lambda_m$ is a multi-color for the ring of $H$ (say $r_a$). In Case (ii), $\lambda_m$ is not a multi-color for any ring of $H$. Let $W^m = \{\lambda_m | \lambda_m \in W_{Q'}^m\}$ is used in Case (ii)). Then at most $|W_{Q'}^m| - |W^m|$ colors of $W_{Q'}^m$ are multi-colors for each ring of $H$. From this, we take $\beta = [0.25L] - |W_{Q'}^m| + |W^m|$ for applying Scheme S32 as a subroutine. To color $P_{ab} \cup P_{ac} \cup P_{bc}$ by virtual colors from $C_H$, we first assign $P_{ab} \cup P_{ac}$ distinct virtual colors from $C_H$. After this, a path colored with a virtual color from $C_H$ on $r_b$ or $r_c$ must be in $P_{ac} \cup P_{ab}$ and thus is on $r_a$. Subject to this condition, we color the paths of $P_{bc}$ with virtual colors from $C_H$ using Scheme S32 with $\beta = [0.25L] - |W_{Q'}^m| + |W^m|$, $r_a, r_b, r_c$ corresponding to $r_0, r_1, r_2$ in the description of S32 in Section 3.2.1, respectively, $P_{ab} \cup P_{ac}$ corresponding to $Q'_1$, and $P_{bc}$ corresponding to $P_{12}$.

After $P_{ab} \cup P_{ac} \cup P_{bc}$ is colored by virtual colors of $C_H$, we map the virtual colors to the colors of $W$. In the mapping, we try to use the colors of $W_{Ra}$ to paths in $P_{bc}$. Similarly, we try to use the colors of $W_{Rb}$ (resp. $W_{Rc}$) to paths in $P_{ac}$ (resp. $P_{ab}$). Notice that the colors
of $W^m$ are not used in the mapping to keep the $[0.25L]$-set condition on each ring. Let $C_{P_{ij}}$ ($i, j = a, b, c; i \neq j$) be the set of virtual colors for $P_{ij}$. We define mapping $f_2 : C_H \to W$ to color the paths in $H$ as follows.

1. Select $|W_{R_0} \setminus W^m|$ virtual colors from $C_{P_{bc}} \setminus (C_{P_{ab}} \cup C_{P_{ac}})$ arbitrarily, and $f_2$ maps each selected color $\nu_i$ to a $\lambda_j \in W_{R_0} \setminus W^m$ such that the selected virtual colors are assigned distinct real colors.

Similarly, select $|W_{R_0} \setminus W^m|$ (resp. $|W_{R_c} \setminus W^m|$) virtual colors from $C_{P_{ac}} \setminus (C_{P_{ab}} \cup C_{P_{bc}})$ (resp. $C_{P_{ab}} \setminus (C_{P_{ac}} \cup C_{P_{bc}})$), and $f_2$ maps each selected virtual color $\nu_i$ to a $\lambda_j \in W_{R_b} \setminus W^m$ (resp. $\lambda_j \in W_{R_c} \setminus W^m$) such that the selected virtual colors are assigned distinct real colors.

Let $C_s$ be the set of selected virtual colors.

2. For each $\nu_i \in C_H \setminus C_s$, $f_2$ maps $\nu_i$ to a $\lambda_j \in W \setminus (W_{R_0} \cup W_{R_b} \cup W_{R_c})$ with the smallest available index $j$ such that $C_H \setminus C_s$ is assigned distinct colors.

The intuition of Step (1) of $f_2$ is to use as many colors of $W_{R_0} \cup W_{R_b} \cup W_{R_c}$ for $E(H)$ as possible. It is shown later that $|C_{P_{bc}} \setminus (C_{P_{ab}} \cup C_{P_{ac}})| \geq |W_{R_0} \setminus W^m|$, $|C_{P_{ac}} \setminus (C_{P_{ab}} \cup C_{P_{bc}})| \geq |W_{R_b} \setminus W^m|$, and $|C_{P_{ab}} \setminus (C_{P_{bc}} \cup C_{P_{ac}})| \geq |W_{R_c} \setminus W^m|$. This implies that Step (1) of $f_2$ can be done.

**Case 2.2:** Ring $r_0$ is a vertex of some $H$ with $L(H) > [2.5L]$. 

Figure 3.5: Paths in and incident to subgraphs $H$ with $L(H) > [2.5L]$. 

(a) $r_0$ is not in $H$

(b) $r_a = r_0$
CHAPTER 3. PATH COLORING ON TREES OF RINGS

Assume \( V(H) = \{ r_a = r_0, r_b, r_c \} \) (see (b) of Figure 3.5). Notice that \( P_{ab} \cup P_{ac} \cup R_a \) has been colored in Step 3.1 and every colored path on \( r_b \) or \( r_c \) is also on \( r_a \). We color \( P_{bc} \) by Scheme S32 with \( \beta = [0.25L], r_a, r_b, r_c \) corresponding to \( r_0, r_1, r_2 \) in the description of S32 in Section 3.2.1, respectively, \( P_{ab} \cup P_{ac} \) corresponding to \( Q_1' \), and \( P_{bc} \) corresponding to \( P_{12} \). Next we color \( R_b \) (resp. \( R_c \)), trying to use the colors of \( P_{ac} \) (resp. \( P_{ab} \)). Note that a color used by \( P_{ac} \) may have already been assigned to a path in \( P_{ab} \) or \( P_{bc} \), and thus cannot be assigned to \( R_b \) without violating the \( [0.25L] \)-set condition for ring \( r_b \) (similar situation exists for the set \( R_c \) and the ring \( r_c \)). To keep the \( [0.25L] \)-set condition for rings \( r_b \) and \( r_c \) and for the set \( R_a \cup R_b \cup R_c \), we do not use multi-colors for \( R_b \) and \( R_c \). More specifically, let \( W^m_{Q'} \) be the set of multi-colors for \( Q' = P_{ab} \cup P_{ac} \cup R_a \) and \( W^m_{Q''} \) be the set of multi-colors for \( Q'' = P_{ab} \cup P_{ac} \cup P_{bc} \). Let \( W_{P_{ab}} \) (resp. \( W_{P_{ac}} \)) be the set of colors for the paths of \( P_{ab} \) (resp. \( P_{ac} \)). For each path \( p \in R_b \) (resp. \( p \in R_c \)), assign \( p \) a color from \( W_{P_{ac}} \setminus (W^m_{Q'} \cup W^m_{Q''}) \) (resp. \( W_{P_{ab}} \setminus (W^m_{Q'} \cup W^m_{Q''}) \)) such that \( R_b \) (resp. \( R_c \)) is assigned distinct colors. Finally, we contract the \( H \) to one vertex, called \( r_0' \), in multigraph \( G_u \) to get another multigraph \( G'_u \). In \( G'_u \), \( r_0' \notin V(H) \) for any subgraph \( H \) of \( G'_u \) with \( l(H) > [2.5L] \) (by Lemma 3.2.6) and the set of paths incident to \( r_0' \) is colored and satisfies the \( [0.25L] \)-set condition (this will be shown in the proof). We solve the edge-coloring of \( G'_u \) as in previous cases to get a valid path coloring.

Theorem 3.3.1 Algorithm ALG3.2 solves the Min-PC problem on \( TR \) with \( n \) nodes and degree \( 2(k+1) \) using at most \( \min\{3L, \max\{[2.75L], OPT + [1.25L]\}\} \) colors in \( O(nkL(k+L^{1.5})) \) time.

Proof: We show that for every node \( u \), Algorithm ALG3.2 colors \( Q_u \cup P_u \) with at most \( \min\{3L, \max\{[2.75L], OPT + [1.25L]\}\} \) colors. In Step 2, we get a valid coloring for \( Q_u \cup P_u \) and the \( [0.25L] \)-set condition for every ring of \( TR \) with \( 2L \) colors. In Step 3.1, by Lemma 3.2.2, we get a valid coloring for \( Q_0 \cup P_0 \) and the \( [0.25L] \)-set condition with at most \( \min\{3L, \max\{[2.75L], OPT + L\}\} \) colors. In Step 3.2, for Case 1 of \( l(G_u) \leq [2.5L] \), by the \( [0.25L] \)-set condition, the paths corresponding to edges in \( G_u \) can be colored with at most \( [2.75L] \) colors. Thus, we can get a valid coloring for \( Q_u \cup P_u \) with at most \( \min\{3L, \max\{[2.75L], OPT + [1.25L]\}\} \) colors, and the \( [0.25L] \)-set condition holds after the coloring.

For Case 2 of \( l(G_u) > [2.5L] \), in Case 2.1, we contract \( G_u \) into \( F_u \) and solve the edge-coloring of \( F_u \). The paths corresponding to the edges of \( F_u \) can be colored with at most
colors, according to Lemma 3.2.6 and the \([0.25L]\)-set condition. For each subgraph
\(H\) of \(G_u\) with \(L(H) > [2.5L]\), the paths in \(H\) are colored by virtual colors of \(C_H\) and \(f_2\)
maps the virtual colors to real colors of \(W\). To see Step (1) of \(f_2\) can be done, we show
that \(|C_{P_{bc}} \setminus (C_{P_{ab}} \cup C_{P_{ac}})| \geq |W_{R_a} \setminus W^m|\), i.e., the number of virtual colors used only by
\(P_{bc}\) is greater than or equal to the number of real colors used only by \(R_a\). By Lemma 3.2.3,
\(|C_H| \geq |E(H)| - \beta\), where \(\beta = [0.25L] - |W_{Q'}^m| + |W^m|\) (recall that \(Q' = R_a \cup R_b \cup R_c\), \(W_{Q'}^m\)
is the set of multi-colors for \(Q'\), and \(W^m\) is the subset of \(W_{Q'}^m\) such that the two paths with
a color from \(W^m\) are incident to different vertices of \(H\). Since \(|E(H)| > [2.5L]\),
\(|C_H| \geq |E(H)| - \beta > [2.25L] + |W_{Q'}^m| - |W^m| \geq [2.25L].

Therefore,
\[
|C_{P_{bc}} \setminus (C_{P_{ab}} \cup C_{P_{ac}})| \geq |C_H| - (|C_{P_{ab}}| + |C_{P_{ac}}|)
\geq |C_H| - (|P_{ab}| + |P_{ac}|)
\geq [2.25L] - (d(r_a) - |R_a|)
\geq [0.25L] + |W_{R_a}|
\geq |W_{R_a} \setminus W^m|.
\]

Similarly, \(|C_{P_{ac}} \setminus (C_{P_{ab}} \cup C_{P_{bc}})| \geq |W_{R_a} \setminus W^m|\) and \(|C_{P_{ab}} \setminus (C_{P_{ac}} \cup C_{P_{bc}})| \geq |W_{R_c} \setminus W^m|\).

Summarizing the above, \(f_2\) gives a valid coloring for \(R_a \cup R_b \cup R_c \cup P_{ab} \cup P_{ac} \cup P_{bc}\). In
addition, all the colors of \(R_a \cup R_b \cup R_c\), except those in \(W^m\), are mapped to the virtual
colors in \(C(H)\). The following calculations will show that the total number of colors used
by \(R_a \cup R_b \cup R_c \cup E(H)\) is at most \(|C(H)| + |W^m|\) after the mapping \(f_2\).

Since \(|P_{ab} \cup P_{ac}| \leq 2L\), \(P_{ab} \cup P_{ac}\) can be colored with at most \(2L\) distinct colors. Notice
that \(|E(H)| = |P_{ab} \cup P_{ac} \cup P_{bc}|\). By Lemma 3.2.3, \(|C_H| \leq \max\{|P_{ab} \cup P_{ac} \cup P_{bc}| - \beta, OPT + L\} =
\max\{|E(H)| - ([0.25L] - |W_{Q'}^m| + |W^m|), OPT + L\}. The number of real colors used for
\(R_a \cup R_b \cup R_c \cup P_{ab} \cup P_{ac} \cup P_{bc}\) is at most, noting that \(|E(H)| \leq \left\lfloor \frac{6L - |Q'|}{2} \right\rfloor \leq 3L - |W_{Q'}^m|\) and
CHAPTER 3. PATH COLORING ON TREES OF RINGS

55

\[ |W^m| \leq |0.25L|, \]

\[ |W_{Ra} \cup W_{Rb} \cup W_{Rc}| + \max\{|E(H)| - (|0.25L| - |W^m_Q| + |W^m|), \]

\[ OPT + L\} - (|W_{Ra} \setminus W^m| + |W_{Rb} \setminus W^m| + |W_{Rc} \setminus W^m|) \]

\[ \leq \max\{|E(H)| - |0.25L| + |W^m_Q| - |W^m|, OPT + L\} + |W^m| \]

\[ \leq \max\{3L - |W^m_Q| - |0.25L| + |W^m_Q| - |W^m|, OPT + L\} + |W^m| \]

\[ = \max\{[2.75L], OPT + L + |W^m|\} \]

\[ \leq \max\{[2.75L], OPT + [1.25L]\}. \]

Now we show the \([0.25L]\)-set condition is true for every ring. Notice that the paths of \(R_a \cup R_b \cup R_c\) are given distinct virtual colors of \(C_{G_u}\) in the edge-coloring of \(F_u\) because all these paths are incident to the same vertex \(v_H\). Therefore, only the paths incident to \(r_0\) may be colored by real multi-colors when a valid coloring from \(W\) to the paths of \(F_u\) is found. This implies that \(W_{R_i} \cap W_{R_j} \subseteq W^m\) \((i, j = a, b, c, i \neq j)\). From this, sets \((W_{R_i} \setminus W^m)\) \((i = a, b, c)\) are pairwise disjoint. Recall that \(C_{P_{ab}}\) and \(C_{P_{ac}}\) are the sets of virtual colors from \(C_H\) assigned to \(P_{ab}\) and \(P_{ac}\), respectively. The mapping \(f_2\) selects a subset of \(C_{P_{ab}}\) (resp. \(C_{P_{ac}}\)), assigns the subset distinct real colors from \(W_{Ra}\setminus W_{Rb}\setminus W_{Rc}\) (resp. \(W_{Ra}\setminus W_{Rb}\setminus W_{Rc}\)), and assigns the remaining colors of \(C_{P_{ab}}\cup C_{P_{ac}}\) distinct real colors from \(W\setminus (W_{Ra} \cup W_{Rb} \cup W_{Rc})\). From \((W_{Ra} \setminus W^m) \cap (W_{Rb} \setminus W^m) = \emptyset\) and the fact that the paths in \(P_{ab} \cup P_{ac}\) are given distinct virtual colors of \(C_H\), the mapping \(f_2\) assigns the paths in \(P_{ab} \cup P_{ac}\) distinct real colors not in \(W_{Ra} \setminus W^m\). Therefore, the \([0.25L]\)-set condition is true for \(r_a\). There are at most \(\beta = |0.25L| - |W^m_Q| + |W^m|\) virtual multi-colors of \(C_H\) for \(P_{bc}\) and each of them is mapped to a distinct real color in \(W_{Ra} \setminus W^m\) or \(W \setminus (W_{Ra} \cup W_{Rb} \cup W_{Rc})\). Therefore, there are at most \(|W^m_Q| - |W^m| + \beta = |0.25L|\) real multi-colors for the paths on each of rings \(r_b\) and \(r_c\). That is, the \([0.25L]\)-set condition is true for every ring.

In Case 2.2, \(Q' = R_a \cup P_{ab} \cup P_{ac}\) has been colored with at most \(2L\) colors and is a \([0.25L]\)-set. In addition, \(P_{ab} \cup P_{ac}\) contains every colored path intersecting with a path of \(P_{bc}\). By Lemma 3.2.3 and \(|E(H)| \leq 3L\), \(Q'' = P_{ab} \cup P_{ac} \cup P_{bc}\) is colored with at most \(\min\{3L, \max\{[2.75L], OPT + L\}\}\) colors. From this and Lemma 3.1.2, \(R_a \cup P_{ab} \cup P_{ac} \cup P_{bc}\) is colored with at most \(\min\{3L, \max\{[2.75L], OPT + [1.25L]\}\}\) colors. On the other hand, by Lemma 3.2.3, \(Q''\) is colored with at least \(|E(H)| - |0.25L|\) colors. Since \(|E(H)| > [2.5L]\), \(|W^m_Q| \leq |0.25L|, |W^m_{Q''}| \leq |0.25L|, W^m_Q \setminus W^m_{Q''} \subseteq W_{P_{bc}}\), and \(|W_{P_{ab}} \cup W_{P_{ac}} \cup W_{P_{bc}}| = \)
CHAPTER 3. PATH COLORING ON TREES OF RINGS

\[ |E(H)| - |W_{Q'}^m|, \] we have
\[ |W_{Pa_b} \setminus (W_{Q'}^m \cup W_{Q''}^m)| \geq |W_{Pa_b} \setminus W_{Pc}| - |W_{Q'}^m| \]
\[ \geq (|E(H)| - |W_{Q'}^m| - |W_{Pa_c}| - |W_{Pc}|) - |W_{Q'}^m| \]
\[ > [2.25L] - (\delta(r_c) - |R_c|) - [0.25L] \]
\[ \geq |R_c|. \]

Similarly, \[ |W_{Pa_c} \setminus (W_{Q'}^m \cup W_{Q''}^m)| \geq |R_b|. \] Therefore, \( R_c \) (resp. \( R_b \)) is assigned distinct colors of \( W_{Pa_b} \setminus (W_{Q'}^m \cup W_{Q''}^m) \) (resp. \( W_{Pa_c} \setminus (W_{Q'}^m \cup W_{Q''}^m) \)), and \( R_a \cup R_b \cup R_c \cup P_{ab} \cup P_{ac} \cup P_{bc} \) is colored with at most \( \min\{3L, \max\{[2.75L], OPT + [1.25L]\}\} \) colors. The \( [0.25L] \)-set condition holds for \( R_a \cup R_b \cup R_c \) and each of rings \( r_a, r_b, \) and \( r_c \), because no multi-color is introduced when coloring \( R_b \) and \( R_c \). By solving the edge-coloring of \( G' \), we can get a valid coloring for \( Q_u \cup P_u \) with at most \( \min\{3L, \max\{[2.75L], OPT + [1.25L]\}\} \) colors.

The edge-coloring of \( G_u \) takes \( O(kL(k + L)) \) time by Proposition 2.1.2. It takes \( O(L^{2.5}) \) time to color a subgraph \( H \) of \( G_u \) with \( L(H) > [2.5L] \) (since \( H \) has degree at most \( 2L \)), the graph constructed in Scheme S32 has \( O(L) \) vertices, and it takes \( O(L^{2.5}) \) time to find a maximum matching in such graph [91]). There can be \( O(k) \) such subgraphs \( H \) in \( G_u \). Therefore, it takes \( O(kL(k + L) + kL^{2.5}) = O(kL(k + L^{1.5})) \) time in Steps 3.1 and 3.2. The algorithm executes these steps \( O(n) \) times. The time complexity of the algorithm is \( O(nkL(k + L^{1.5})) \).

Since \( L \leq OPT \), \( \min\{3L, \max\{[2.75L], OPT + [1.25L]\}\}/OPT \leq [2.75L]/L. \) Thus Algorithm ALG3.2 achieves an approximation ratio of 2.75 asymptotically.

It seems difficult to extend the approach used in Algorithm ALG3.2 to improve the approximation ratio of 2.75 for a tree of rings with arbitrary degrees. One possible direction is to lower the threshold value to some \( T < [2.5L] \) for subgraph \( H \). However, this will introduce the following problems. First, a subgraph \( H \) may have five or more vertices. A new scheme for coloring \( H \) is needed. Second, after the contraction of \( H \) in \( G_u \), the resulting graph \( F_u \) may still have \( l(F_u) > T \). To apply the edge-coloring algorithm, we may need to contract \( F_u \) as well. A new mapping function for converting the virtual colors of edge-coloring to real colors is needed. It is difficult to solve either of the problems. Nevertheless, in the next section, we show that the approach of Algorithm ALG3.2 can be used to derive algorithms with improved approximation ratios for bounded degree trees of rings.
CHAPTER 3. PATH COLORING ON TREES OF RINGS

3.4 Algorithms for Bounded Degrees

The ideas for the 2.75-approximation algorithm can be used to design more efficient algorithms for the Min-PC problem on trees of rings with bounded degrees. Actually, Schemes S31 and S32 shown in Section 3.2.1 imply a $3L$ and 2-approximation algorithm for the Min-PC problem on trees of rings with degree at most six. We first give the algorithm explicitly and then describe algorithms for degrees eight and ten.

3.4.1 Algorithm for Degree Six

The algorithm, called ALG3.3, follows the framework in Figure 3.2. Step 2 of ALG3.3 is the same as that in Algorithm ALG3.1. Step 3.1 uses Scheme S31. In Step 3.2, we first use Scheme S32 to color the long paths in $P_{12}$. Then we color the short paths on $r_1$ and those on $r_2$. Let $Q'$ be the set of all long paths on $u$ and $r_1$. We assign the short paths on $r_1$ the colors of $W \setminus W_{Q'}$ by the first-fit coloring such that the set of short paths is assigned distinct colors. Let $Q''$ be the set of all long paths on $u$ and $r_2$. We assign the short paths on $r_2$ the colors of $W \setminus W_{Q''}$ by the first-fit coloring such that the set of short paths is assigned distinct colors.

Theorem 3.4.1 Algorithm ALG3.3 solves the Min-PC problem on $T_R$ with $n$ nodes and degree at most six using at most $\min\{OPT + L, 3L\}$ colors in $O(nL^{2.5})$ time.

Proof: To prove the theorem, we take $\beta = L$. We show that Algorithm ALG3.3 colors $Q_u \cup P_u$ using at most $\min\{OPT + L, 3L\}$ colors for every node $u$ in $T_R$.

In Step 2, $Q_u = \emptyset$ and $|P_u| \leq 2L$. At most $2L \leq \min\{OPT + L, 3L\}$ colors are used for $Q_u \cup P_u$. Obviously, the $L$-set condition is true for every ring after Step 2. In Step 3.1, by Lemma 3.2.2 and $|Q_0 \cup P_0| \leq 3L$, at most $\min\{3L, \max\{3L - L, OPT_0 + L\}\}$ colors are used for $Q_0 \cup P_0$. Since $OPT_0 \leq OPT$ and $L \leq OPT$, $\max\{3L - L, OPT_0 + L\} \leq OPT + L$ and at most $\min\{OPT + L, 3L\}$ colors are used for $Q_u \cup P_u$. By Lemma 3.2.2, the $L$-set condition is true for every ring after Step 3.1.

In Step 3.2, recall that $Q_1' \subseteq Q_1$ is the set of colored long paths on links $(u, u^-)$ or $(u, u^+)$. Each path of $Q_1'$ (resp. $P_{12}$) is on one (resp. two) of the four links incident to $u$ in $r_1$ and $r_2$, implying $|Q_1'| + 2|P_{12}| \leq 4L$. From this and $|Q_1'| \leq 2L$, we have $|Q_1' \cup P_{12}| \leq 3L$. Notice that $Q_1'$ contains every colored path intersecting with a path of $P_{12}$. By Lemma 3.2.3, $OPT_1 \leq OPT$, $L \leq OPT$, and $|Q_1' \cup P_{12}| \leq 3L$, at most $\min\{OPT + L, 3L\}$ colors are used
for $Q'_i \cup P_{12}$. By Lemma 3.2.3, the $L$-set condition is true for every ring after the coloring of $P_{12}$.

Since there are at most $2L$ paths on the two links of $r_1$ (resp. $r_2$) that are incident to node $u$, all the paths on the two links in $r_1$ (resp. $r_2$), including all the short paths, can be colored with $2L$ colors. Obviously, the $L$-set condition is true for every ring after the short paths are colored. Thus, Algorithm ALG3.3 colors $Q_u \cup P_u$ with at most $\min\{OPT+L,3L\}$ colors and keeps the $L$-set condition for every ring.

A tree of rings $TR$ with $n$ nodes has $O(n)$ links. There are at most $O(nL)$ paths in a tree of rings with load $L$. To reduce the time complexity, we first construct a conflict graph $G_c$ whose vertex set is $P$ and two vertices of $G_c$ are adjacent if the corresponding paths of $P$ intersect with each other in $TR$. The conflict graph can be constructed in $O(nL^2)$ time, assuming that each path of $P$ is given as a linked list of links of $TR$. The algorithm executes Steps 3.1 and 3.2 $O(n)$ times. The first-fit coloring takes $O(L^2)$ time to color $L$ paths. It takes $O(L^2)$ time to construct a graph $G_u$ of $O(L)$ vertices, by checking the conflict graph (there is an edge between two vertices of $G_u$ if there is no edge between the two vertices in the conflict graph). It takes $O(L^{2.5})$ time to find a maximum matching of the graph [91]. Therefore, Steps 3.1 and 3.2 take $O(L^{2.5})$ time. The time complexity of the algorithm is $O(nL^{2.5})$.

3.4.2 Algorithm for Degree Eight

The algorithm for degree eight, called ALG3.4, is similar to Algorithm ALG3.2, but uses a special scheme for the edge-coloring of multigraph $G_u$. Since the tree of rings considered has degree eight, $G_u$ has at most four vertices $r_i$. Since the paths with an end vertex of $s_i$ of $G_u$ are short paths which can be easily colored with $2L$ colors after the long paths are colored, in what follows, we assume that $G_u$ has only vertices $r_i$ and edges corresponding to long paths. We first show an optimal edge-coloring algorithm for a multigraph with four vertices. We follow the notation used for Algorithm ALG3.2. Especially, for a subgraph $H$ of multigraph $G_u$ with $V(H) = \{r_a,r_b,r_c\}$, we use $P_{ij}$ ($i,j = a,b,c; i \neq j$) for the sets of long paths in $H$ on $r_i$ and $r_j$, and use $R_i$ ($i = a,b,c$) for the sets of paths not in $H$ but on $r_i$.

**Lemma 3.4.2** An edge-coloring of multigraph $G_u$ with four vertices can be done using at most $\max\{\Delta(G_u),l(G_u)\}$ colors in $O(|E(G_u)|)$ time.
**Proof:** If $l(G_u) > \Delta(G_u)$, then there exists a subgraph $H$ with three vertices which has $L(H) = l(G_u)$. To see this, assume that for any subgraph $H$ with three vertices, $L(H) < l(G_u)$. The remaining vertex, which is not in $H$, has degree at most $\Delta(G_u)$. Then

$$l(G_u) \leq \left\lfloor \frac{L(H) + \Delta(G_u)}{2} \right\rfloor < l(G_u),$$

a contradiction. For the subgraph $H$ with $L(H) = l(G_u)$, assume that $V(H) = \{r_a, r_b, r_c\}$ (see Figure 3.6). We first color the edges $E(H)$ using $l(G_u)$ distinct colors. From $E(H) = l(G_u) > \Delta(G_u)$, we have

$$|P_{bc}| = |E(H)| - (d(r_a) - |R_a|) \geq |R_a|.$$

Thus, all edges of $R_a$ can be colored by the colors used for $P_{bc}$, since each edge of $R_a$ does not share a common vertex with any edge of $P_{bc}$. Similarly, all edges of $R_b$ (resp. $R_c$) can be colored by the colors used for $P_{ac}$ (resp. $P_{ab}$). Therefore, $G_u$ can be edge-colored with at most $l(G_u)$ colors.

For the case of $l(G_u) \leq \Delta(G_u)$, assume that $d(r_0) = \Delta(G_u)$. We first color the edges incident to $r_0$ by $\Delta(G_u)$ distinct colors. Assume that the remaining vertices of $G_u$ are $r_a$, $r_b$, and $r_c$ (see (a) of Figure 3.6). Let $H$ be the subgraph with $V(H) = \{r_0, r_a, r_b, r_c\}$. Then

$$|E(H)| \leq l(G_u) \leq \Delta(G_u),$$

and we have

$$|P_{bc}| = |E(H)| - (|R_a| + |R_c|) = \Delta(G_u) - (|E(H)| - |P_{bc}|) \geq |P_{bc}|.$$

So, all edges of $P_{bc}$ can be colored by the colors used for $R_a$. Similarly, all edges of $P_{ab}$ (resp. $P_{ac}$) can be colored by the colors used for $R_c$ (resp. $R_b$). Thus, $G_u$ can be edge-colored with at most $\Delta(G_u)$ colors.

The algorithm first needs to find the larger number of $l(G_u)$ and $\Delta(G_u)$. This takes $O(|E(G_u)|)$ time. The coloring takes $O(|E(G_u)|)$ time. Thus, the time complexity of the algorithm is $O(|E(G_u)|)$.

Algorithm ALG3.4 follows the framework of Figure 3.2. Step 2 of ALG3.4 is the same as that in Algorithm ALG3.1. Step 3.1 uses Scheme S31 to color $P_0$ taking $\beta = [0.5L]$. In Step 3.2 of ALG3.4, to color $P_1$, we convert the path coloring problem to the edge-coloring problem of multigraph $G_u$. Similar to Algorithm ALG3.2, there are two cases.

**Case 1:** $l(G_u) \leq 2L$.

In this case we edge-color $G_u$ by the algorithm given in Lemma 3.4.2 using at most $2L$ virtual colors, re-assign virtual colors to the paths which have been assigned multi-colors of
Figure 3.6: Paths in and incident to the subgraph $H$ with four vertices and $L(H) > 2L$.

$W$ before Step 3.2, and apply the mapping $f_1$ to color the paths of $P_1$, as we did in Case 1 of Step 3.2 of Algorithm ALG3.2.

**Case 2:** $l(G_u) > 2L$.

In this case, there is a subgraph $H$ of $G_u$ with $L(H) = l(G_u)$. There are two subcases.

**Case 2.1:** Ring $r_0$ is not a vertex in $H$.

We color the paths in $H$ by virtual colors of $C_H = \{v_1, v_2, \ldots\}$. Assume that $V(H) = \{r_a, r_b, r_c\}$. Notice that $R_a \cup R_b \cup R_c$ has been colored. Let $W^m_{Q'}$ be the set of multi-colors for $Q' = R_a \cup R_b \cup R_c$. From the $|0.5L|$-set condition, $|W^m_{Q'}| \leq |0.5L|$. Let $W^m$ be the subset of $W^m_{Q'}$ such that the two paths with a color from $W^m$ are incident to different vertices of $H$ (see (a) of Figure 3.6). Similar to Case 2.1 of Algorithm ALG3.2, we color $P_{ab} \cup P_{ac}$ by distinct virtual colors and $P_{bc}$ by Scheme S32 with

$$\beta = \min\{|0.5L| - |W^m_{Q'}| + |W^m|, |E(H)| - 2L\},$$

using virtual colors of $C_H$. Then we apply mapping $f_2$ in the same way as that in Algorithm ALG3.2, using as many as possible of the colors of $W_{R_a}, W_{R_b},$ and $W_{R_c}$ to color $P_{bc}, P_{ac},$ and $P_{ab}$, respectively.

**Case 2.2:** Ring $r_0$ is a vertex of $H$.

Assume $V(H) = \{r_a = r_0, r_b, r_c\}$ (see (b) of Figure 3.6). Notice that $R_a \cup P_{ab} \cup P_{ac}$ has been colored with at most $2L$ colors. Let $W^m_{Q'}$ be the set of multi-colors for $Q' = P_{ab} \cup P_{ac}$. 

```plaintext

60

CHAPTER 3. PATH COLORING ON TREES OF RINGS

(a) $r_0$ is not in $H$

(b) $r_a = r_0$

Colored by $W^m$

```
CHAPTER 3. PATH COLORING ON TREES OF RINGS

Similar to Case 2.2 of Algorithm ALG3.2, we color $P_{bc}$ by Scheme S32 with

$$\beta = \max \{|E(H)| - |\mathbb{R}|, |W_m^m|\},$$

using colors of $W_{Ra} \setminus (W_{Pa} \cup W_{Pa}^c)$ first and then colors of $W \setminus (W_{Ra} \cup W_{Pa} \cup W_{Pa}^c)$. After this we assign $R_b$ distinct colors from $W_{Pa} = W_{Pa} \setminus (W_{Ra} \cup W_{Pa} \cup W_{Pa}^c)$ first and then from $W \setminus (W_{Ra} \cup W_{Pa} \cup W_{Pa} \cup W_{Pa}^c)$ by the first-fit coloring such that $R_b$ is assigned distinct colors. Similarly, we assign $R_c$ distinct colors from $W_{Pa} = W_{Pa} \setminus (W_{Ra} \cup W_{Pa} \cup W_{Pa}^c)$ first and then from $W \setminus (W_{Ra} \cup W_{Pa} \cup W_{Pa} \cup W_{Pa}^c)$ by the first-fit coloring.

**Theorem 3.4.3** Algorithm ALG3.4 solves the Min-PC problem on TR with $n$ nodes and degree eight using at most $\min\{3L, OPT + 1.5L\}$ colors in $O(nL^{2.5})$ time.

**Proof:** We prove that Algorithm ALG3.4 colors $Q_u \cup P_u$ with at most $\min\{3L, OPT + 1.5L\}$ colors for every node $u$. Similar to the proof for Algorithm ALG3.2, if $l(G_u) \leq 2L$, we get a valid coloring for $Q_u \cup P_u$ with at most $\min\{3L, OPT + 1.5L\}$ colors and the $|0.5L|$-set condition holds after the coloring.

Assume that $l(G_u) > 2L$ in Step 3.2. In Case 2.1, by Lemma 3.2.3, the paths in $H$ are colored with at least $|E(H)| - \beta \geq 2L$ virtual colors, where $\beta = \min\{|0.5L| - |W_Q^m|, |W_m|, |E(H)| - 2L\}$, $Q' = R_a \cup R_b \cup R_c$, and $W_m$ is the set of multi-colors on $Q'$. Let $C_{P_{ij}}$ $(i, j = a, b, c; i \neq j)$ be the subset of virtual colors of $C_H$ assigned to $P_{ij}$. We have

$$|C_{Pa} \setminus (C_{Pa} \cup C_{Pa}^c)| \geq |C_H| - |C_{Pa}^c| - |C_{Pa}^c| \geq 2L - (d(r_a) - |R_a|) \geq |W_{Ra}|.$$

Similarly, $|C_{Pa} \setminus (C_{Pa} \cup C_{Pa}^c)| \geq |W_{Ra}|$ and $|C_{Pa} \setminus (C_{Pa} \cup C_{Pa}^c)| \geq |W_{R_b}|$. Thus Step (1) of mapping $f_2$ can be done and we get a valid coloring for $R_a \cup R_b \cup R_c \cup E(H)$. The number of colors used is, noting $|E(H)| \leq \left[\frac{nL - |Q'|}{2}\right] \leq 3L - |W_Q^m|$ and $|W_m| \leq \lfloor 0.5L\rfloor$,

$$\max\{|E(H)| - \beta, OPT + L\} + |W_m|$$

$$\leq \max\{|E(H)| - (|0.5L| - |W_Q^m| + |W_m|), |E(H)| - (|E(H)| - 2L),$$

$$OPT + L\} + |W_m|$$

$$\leq \max\{3L - |W_Q^m| - |0.5L| + |W_Q^m| - |W_m|, OPT + L\} + |W_m|$$

$$= \max\{|2.5L|, OPT + L + |W_m|\}$$

$$\leq OPT + 1.5L.$$

$P_{ab} \cup P_{ac}$ are given distinct real colors. There are at most $\beta$ virtual multi-colors of $C_H$ for $P_{bc}$ and each of them is mapped to a distinct real color in $W_{Ra} \setminus W_m$ or $W \setminus (W_{Ra} \cup W_{R_b} \cup$
Notice that \( W_{R_i} \cap W_{R_j} \subseteq W^m \) for \( i, j = a, b, c; i \neq j \). Therefore, there are at most \( |W^m_Q| - |W^m| + \beta = [0.5L] \) real multi-colors for the paths on each of rings \( r_a, r_b, \) and \( r_c \). That is, the \([0.5L]\)-set condition is true for every ring.

In Case 2.2, \( P_{bc} \) is colored by Scheme S32 with \( \beta = \max\{|E(H)| - [2.5L], |W^m_Q|\} \), where \( Q' = P_{ab} \cup P_{ac} \) and \( W^m_Q \) is the set of multi-colors on \( Q' \). Since \( |E(H)| \leq 3L, \beta \leq [0.5L] \). Notice that \( Q' \) contains every colored path intersecting with a path of \( P_{bc} \). By Lemma 3.1.2 and Lemma 3.2.3, \( R_a \cup E(H) \) can be colored with at most \( \min\{3L, \max\{[2.5L], OPT + L\}\} \leq \min\{3L, OPT + [1.5L]\} \) colors. Recall that we choose \( \beta = \max\{|E(H)| - [2.5L], |W^m_Q|\} \) in the algorithm. Consider which of the two values \( \beta \) takes. Assume that \( \beta = |E(H)| - [2.5L] \).

We show that \( R_b \) can be assigned distinct colors of \( W'_{P_{ac}} \). Notice that

\[
|W'_{P_{ac}}| \geq |W_{P_{ac}} \setminus W_{P_{bc}}| - |W_{P_{ac}} \cap (W_{R_a} \cup W_{P_{ab}})|.
\]

By Lemma 3.2.3, at least \([2.5L]\) colors are used for \( H \). Since \( W_{P_{ac}} \cap (W_{R_a} \cup W_{P_{ab}}) \) is a subset of the multi-colors on \( W_{R_a} \cup W_{P_{ab}} \cup W_{P_{ac}} \), from the \([0.5L]\)-set condition on ring \( r_a \), \( |W_{P_{ac}} \cap (W_{R_a} \cup W_{P_{ab}})| \leq [0.5L] \). Therefore,

\[
|W'_{P_{ac}}| \geq (|2.5L| - |W_{P_{ab}}| - |W_{P_{bc}}|) - [0.5L] \\
\geq |2.5L| - (d(r_b) - |R_b|) - [0.5L] \\
\geq |R_b|.
\]

From this, \( R_b \) can be assigned distinct colors of \( W'_{P_{ac}} \). Similarly, \( R_c \) can be assigned distinct colors of \( W'_{P_{ab}} \). Thus \( R_a \cup R_b \cup R_c \cup E(H) \) can be colored with at most \( \min\{3L, OPT + [1.5L]\} \) colors, and from \( \beta = |E(H)| - [2.5L] \leq [0.5L] \) the \([0.5L]\)-set condition holds for \( R_a \cup R_b \cup R_c \) and each of rings \( r_a, r_b, \) and \( r_c \).

Assume that \( \beta = |W^m_Q| \). By the proof above, we can assume that \( |W'_{P_{ac}}| < |R_b| \) or \( |W'_{P_{ab}}| < |R_c| \). We further assume, without loss of generality, that \( |W'_{P_{ac}}| < |R_b| \) and \( |W'_{P_{ab}}| < |R_c| \) (the other two cases can be proved similarly). From \( \beta = |W^m_Q| \) and Scheme S32, \( P_{bc} \) is assigned distinct colors and \( W_{P_{bc}} \cap (W_{P_{ab}} \cup W_{P_{ac}}) = \emptyset \). From \( E(H) > 2L \) and \( d(r_a) \leq 2L, |R_{bc}| > |R_a| \) which implies that all colors of \( W_{R_a} \setminus (W_{P_{ab}} \cup W_{P_{ac}}) \) are used for \( P_{bc} \). Therefore, \( R_a \cup E(H) \) is colored with at most \( E(H) - |W^m_Q| \) colors. To color \( R_b \) and \( R_c \),

\[
(|R_b| - |W'_{P_{ac}}|) + (|R_c| - |W'_{P_{ab}}|)
\]
additional colors are needed. Since $W_{P_{ac}} \cap (W_{P_{ab}} \cup W_{P_{ac}}) = \emptyset$,

\[
(|R_b| - |W'_{P_{ac}}|) + (|R_c| - |W'_{P_{ab}}|)
\]

\[
= (|R_b| + |R_c|) - (|W_{P_{ac}} \setminus (W_{Ra} \cup W_{P_{ab}})| + |W_{P_{ab}} \setminus (W_{Ra} \cup W_{P_{ac}})|)
\]

\[
\leq (|R_b| + |R_c|) - (|W_{P_{ac}}| + |W_{P_{ab}}| - |W_{P_{ac}} \cap W_{Ra}| - |W_{P_{ab}} \cap W_{Ra}|
\]

\[
-2|W_{P_{ab}} \cap W_{P_{ac}}|).
\]

Since $(W_{P_{ac}} \cap W_{Ra}) \cup (W_{P_{ab}} \cap W_{Ra}) \cup (W_{P_{ab}} \cap W_{P_{ac}})$ is a subset of the multi-colors on $R_a \cup P_{ab} \cup P_{ac}$, from the $[0.5L]$-set condition on ring $r_a$, $|W_{P_{ac}} \cap W_{Ra}| + |W_{P_{ac}} \cap W_{P_{ab}}| + |W_{P_{ab}} \cap W_{Ra}| \leq [0.5L]$ (noting that $W_{Ra} \cap W_{P_{ac}} \cap W_{P_{ab}} = \emptyset$). Since $W_{Q'}^m$ is the set of multi-colors on $Q' = P_{ab} \cup P_{ac}$,

\[
|W_{P_{ac}}| + |W_{P_{ab}}| - |W_{P_{ab}} \cap W_{P_{ac}}| = |P_{ac}| + |P_{ab}| - |W_{Q'}^m|.
\]

Also notice that $|P_{ac}| = |E(H)| - d(r_b) + |R_b|$ and $|P_{ab}| = |E(H)| - d(r_c) + |R_c|$. Summarizing the above,

\[
(|R_b| - |W'_{P_{ac}}|) + (|R_c| - |W'_{P_{ab}}|))
\]

\[
\leq |R_b| + |R_c| - (|P_{ac}| + |P_{ab}| - |W_{Q'}^m| - [0.5L])
\]

\[
= d(r_b) + d(r_c) + |W_{Q'}^m| - [0.5L] - 2|E(H)|.
\]

Since at most $|E(H)| - |W_{Q'}^m|$ colors are used for $R_a \cup E(H)$, $d(r_b), d(r_c) \leq 2L$, and $|E(H)| > 2L$, the total number of colors used for $R_a \cup R_b \cup R_c \cup E(H)$ is bounded by

\[
d(r_b) + d(r_c) + |W_{Q'}^m| + [0.5L] - 2|E(H)| + |E(H)| - |W_{Q'}^m| \leq [2.5L].
\]

Obviously, the $[0.5L]$-set condition holds for $R_a \cup R_b \cup R_c$ and each of rings $r_a, r_b$, and $r_c$.

The edge-coloring of $G_u$ takes $O(L)$ time by Lemma 3.4.2. It takes $O(L^{2.5})$ time to color the subgraph $H$ of $G_u$ with $L(H) > 2L$. The first-fit coloring takes $O(L^2)$ time to color $L$ paths. Therefore, it takes $O(L^{2.5})$ time in Steps 3.1 and 3.2. The algorithm executes these steps $O(n)$ times. The time complexity of the algorithm is $O(nL^{2.5})$. \[\square\]

It seems difficult to generalize the approach of Algorithm ALG3.4 for the Min-PC problem on trees of rings with larger constant degrees, although a similar but more complicated analysis shows that the 2.5 approximation ratio is achievable for degree at most ten.
3.4.3 Algorithm for Degree Ten

The algorithm for degree ten, called ALG3.5, is similar to Algorithm ALG3.4, but uses a special scheme for the edge-coloring of multigraph $G_u$. Since the tree of rings considered has degree ten, $G_u$ has at most five vertices. For a multigraph with five vertices, Lemma 3.4.2 can be extended as follows.

**Lemma 3.4.4** An edge-coloring of multigraph $G_u$ with five vertices can be done using at most $\max\{\Delta(G_u) + 1, l(G_u)\}$ colors.

Lemma 3.4.4 follows from the 1.1 edge-coloring algorithm [95]. In the 1.1 edge-coloring algorithm, if a critical path does not contain two vertices with the same missing color, then it has 3, 5 or 7 vertices. $1.1\Delta + 0.8$ colors are needed to ensure a valid edge-coloring. This is not needed in a multigraph with at most 5 vertices. The lemma is true for any multigraph with at most eight vertices.

Algorithm ALG3.5 follows the framework of Figure 3.2. We follow the notation used for Algorithm ALG3.4. Especially, for a subgraph $H$ of multigraph $G_u$ with $V(H) = \{r_a, r_b, r_c\}$, we use $P_{ij}$ ($i,j = a, b, c; i \neq j$) for the sets of long paths in $H$ on $r_i$ and $r_j$, and use $R_i$ ($i = a, b, c$) for the sets of paths not in $H$ but on $r_i$. We use $R_d$ to denote the set of paths not on any ring of $H$ (see Figure 3.7(b)).

Step 2 of ALG3.5 is the same as that in Algorithm ALG3.1. Step 3.1 uses Scheme S31 to color $P_0$ taking $\beta = \lfloor 0.5L \rfloor$. In Step 3.2 of ALG3.5, to color $P_1$, we convert the path coloring problem to the edge-coloring problem of multigraph $G_u$. Similar to Algorithm ALG3.4, there are two cases.

**Case 1:** $l(G_u) \leq 2L$.

In this case we edge-color $G_u$ by the algorithm given in Lemma 3.4.4 using at most $2L + 1$ virtual colors, re-assign virtual colors to the paths which have been assigned multi-colors of $W$ before Step 3.2, and apply the mapping $f_1$ to color the paths of $P_1$, as we did in Case 1 of Step 3.2 of Algorithm ALG3.4. It is easy to see that $f_1$ colors the paths in $G_u$ by at most $2L + 1 + \lfloor 0.5L \rfloor = \lfloor 2.5L \rfloor + 1$ colors if the $\lfloor 0.5L \rfloor$-set condition is true before the step.

**Case 2:** $l(G_u) > 2L$.

We give some more definitions. Edges incident to $r_0$ in $G_u$ are already colored. Let $Q_1$ be the set of paths on $r_0$ (corresponding to edges incident to $r_0$ in $G_u$), $W_{Q_1}^m$, be the set of multi-colors in $W_{Q_1}$, and $Q_1^m \subseteq Q_1$ be the set of edges each of which is colored by
a multi-color ($|Q^n_t| = 2|W^m_{Q_1}|$). For a set of edges $E$ in $G$, we use $G - E$ to denote the subgraph of $G$ obtained by removing the edges in $E$.

Notice that by definition, $L(G_u) \leq l(G_u)$. The value $l(G_u)$ could be achieved on a subgraph with three vertices ($L(H) = l(G_u)$ for some $H \subseteq G_u$ with $|V(H)| = 3$), or on graph $G_u$ ($L(G_u) = l(G_u)$). There are two cases.

Case 2A: If there is a subgraph $H \subseteq G_u$ with $|V(H)| = 3$, and $L(H) = l(G_u)$, then we consider the following two cases.

- Case (i): Ring $r_0$ is not a vertex of the subgraph $H$.

We first assign each edge in $G_u - (E(H) \cup Q_1)$ a distinct color not in $W_{Q_1}$ by the first-fit coloring. These edges are incident to $r_1$ in Figure 3.7(a). Then we contract $r_0$ and $r_1$ to a single node $r_0'$ (throw away any loop edges), and obtain a new graph $G'_u$ which has four vertices. In $G'_u$, only edges incident to $r_0'$ are colored, and they form a $\lfloor 0.5L \rfloor$-set. We color the uncolored edges in $G'_u$ as in Case 2.1 of Algorithm ALG3.4.

- Case (ii): Ring $r_0$ is a vertex of the subgraph $H$.

We contract $r_1$ and $r_2$ to a single node $r_1'$ (throw away any loop edges), and obtain a new graph $G''_u$ which has four vertices (see Figure 3.7(b)). We color the uncolored
edges in $G_u'$ as in Case 2.2 of Algorithm ALG3.4. We then color the edges in $R_a$ with distinct colors not in $W_{R_a} \cup W_{R_b} \cup W_{R_c}$.

**Case 2B:** If $L(G_u) = l(G_u)$, and for all subgraph $H \subseteq G_u$ with $|V(H)| = 3$, $L(H) < l(G_u)$, then $l(G_u)$ is achieved on graph $G_u$ itself (obviously $L(G_u) \leq \lceil 2.5L \rceil$, see Lemma 3.2.4). Let $G'_u = G_u - Q^m_1$. If $l(G'_u) \leq 2L$, then we edge-color $G'_u$ with at most $2L+1$ colors (from Lemma 3.4.4), and then apply mapping $f_1$. Otherwise, $l(G'_u) > 2L$. If $L(G'_u) = l(G'_u)$, then we edge-color $G'_u$ using $l(G'_u)$ colors and apply mapping $f_1$. Otherwise ($L(G'_u) < l(G'_u)$), there is a subgraph $H' \subseteq G'_u$ such that $L(H') = l(G'_u)$. Let $H$ be the subgraph in $G_u$ induced by the three vertices of $H'$. We then consider the two cases in the same way as Case 2A.

**Theorem 3.4.5** Algorithm ALG3.5 solves the Min-PC problem on TR with $n$ nodes and degree ten using at most $\min\{3L, OPT + \lceil 1.5L \rceil + 1\}$ colors in $O(nL^{2.5})$ time.

**Proof** We only consider Case 2 of $l(G_u) > 2L$ in Step 3.2 (other cases are similar to Algorithm ALG3.4). The coloring of the edges in $H$ and edges in $R_a \cup R_b \cup R_c$ (the edges not in $H$ but incident to the vertices of $H$) is essentially the same as in the degree eight case. There are two important facts: (1) in Case (i) of Case 2A, edges incident to $r_1$ are given distinct colors not in $W_{Q_1}$, and (2) in Case (ii) of Case 2A, $R_b$ and $R_c$ are given distinct colors (this is true following from the coloring process in Case 2.2 of Algorithm ALG3.4). In both Case (i) and Case (ii), the contracted node has degree at most $2L$, if the subgraph $H$ has at least $2L$ edges (this condition is always true when we do the contraction). In the following proof, our main effort is to show that $G_u - E(H)$ can be edge-colored by at most $\lceil 2.5L \rceil$ colors.

In Case 2A, there is a subgraph $H \subseteq G_u$ with $|V(H)| = 3$, and $L(H) = l(G_u)$. We show that $G_u - E(H)$ has at most $\lceil 2.5L \rceil$ edges. This is true if $L(H) = l(G_u) > \lceil 2.5L \rceil$, since $G_u - E(H)$ has at most $\frac{2Lx}{2} - L(H) < 5L - \lceil 2.5L \rceil \leq \lceil 2.5L \rceil$ edges. If $2L < L(H) = l(G_u) \leq \lceil 2.5L \rceil$, then the number of edges in $G_u - E(H)$ is at most $2 \times L(G_u) - L(H) \leq 2l(G_u) - L(H) = l(G_u) \leq \lceil 2.5L \rceil$. Thus $G_u - E(H)$ can be edge-colored by at most $\lceil 2.5L \rceil$ colors. The rest of the proof is the same as the degree eight case.

In Case 2B, $l(G_u)$ is achieved on graph $G_u$ itself (obviously $L(G_u) \leq \lceil \frac{2Lx}{4} \rceil = \lceil 2.5L \rceil$). Let $G'_u = G_u - Q^m_1$. If $l(G'_u) \leq 2L$, then $G'_u$ can be edge-colored by at most $2L+1$ colors, and the total number of colors used for $G_u$ is $2L + 1 + |W_{Q_1}^m| \leq \lceil 2.5L \rceil + 1.)$
CHAPTER 3. PATH COLORING ON TREES OF RINGS

Assume \( l(G'_u) > 2L \). If \( l(G'_u) = l(G_u) \), \( G'_u \) can be edge-colored by \( l(G'_u) \) colors. The total number of colors used for \( G_u \) is

\[
l(G'_u) + |W^m_{Q_1}| = L(G'_u) + |W^m_{Q_1}| = \left\lfloor \frac{2L(G_u) - 2|W^m_{Q_1}|}{2} \right\rfloor + |W^m_{Q_1}| = L(G_u) \leq \lceil 2.5L \rceil.
\]

On the other hand, if \( l(G'_u) < l(G_u) \), there is a subgraph \( H' \subseteq G'_u \) such that \( L(G'_u) < L(H') = l(G'_u) \). Consider the following two cases.

In Case (i) (ring \( r_0 \) is not a vertex in the subgraph \( H \)), \( L(G'_u) = \frac{2L(G_u) - 2|W^m_{Q_1}|}{2} = L(G_u) - |W^m_{Q_1}| \), and \( L(H') = L(H) \). Thus \( L(G_u) - |W^m_{Q_1}| < L(H) \). The edges in \( G_u - E(H) \) are colored by at most

\[
(2L(G_u) - L(H) - 2|W^m_{Q_1}|) + |W^m_{Q_1}| < L(G_u) \leq \lceil 2.5L \rceil
\]
colors.

In Case (ii) (ring \( r_0 \) is a vertex in the subgraph \( H \)), define \( W^m_{Q_1} = W_{R_a} \cap (W_{P_{ab}} \cup W_{P_{ac}}) \) (the set of multi-colors each of which is used to color one edge in \( H \) and one edge not in \( H \)), \( W^m_{Q_1} = W_{R_a} \) (the set of multi-colors in \( W_{R_a} \)), and \( W^m_{Q_1} \) be the set of multi-colors each of which is used to color two edges in \( P_{ab} \cup P_{ac} \). Then \( |W^m_{Q_1}| = |W^m_{Q_1}| + |W^m_{Q_1}| + |W^m_{Q_1}| \). We have \( L(G'_u) = \left\lfloor \frac{2L(G_u) - 2|W^m_{Q_1}|}{2} \right\rfloor \), \( L(G_u) - |W^m_{Q_1}| \), \( L(H) - |W^m_{Q_1}| - 2 \times |W^m_{Q_1}| \). Thus,

\[
L(G_u) - |W^m_{Q_1}| < L(H) - |W^m_{Q_1}| \leq \lceil 2.5L \rceil
\]
which implies

\[
L(G_u) - L(H) - |W^m_{Q_1}| < -|W^m_{Q_1}|.
\]
The edges in \( G_u - E(H) \) are colored by at most

\[
(2L(G_u) - L(H) - 2 \times |W^m_{Q_1}|) + |W^m_{Q_1}| = 2L(G_u) - L(H) - |W^m_{Q_1}|
\]
\[
< L(G_u) - |W^m_{Q_1}| \leq L(G_u) \leq \lceil 2.5L \rceil
\]
colors.

The time complexity of Algorithm ALG3.5 is the same as Algorithm ALG3.4, which runs in \( O(nL^{2.5}) \) time.

It is not clear whether this approach can be used for trees of rings with degree more than ten. For a multigraph with more than five vertices, the five vertices subgraph \( H \) may have \( L(H) > 2L \). However, we do not have an algorithm that uses at most \( OPT + L \) colors for degree more than six.
3.5 Summary

We gave a $3L$ and (asymptotic) 2.75-approximation algorithm for the Min-PC problem on trees of rings with arbitrary degrees. The $3L$ upper bound is tight. We also presented a $3L$ and 2-approximation (resp. 2.5-approximation) algorithm for the Min-PC problem on trees of rings with degree at most six (resp. eight and ten). An interesting problem is to improve the 2.75-approximation ratio. A possible approach is to color the edges of multigraph $G_u$, allowing two edges with a common vertex in a given subset of edges sharing the same color. Another direction for the future work is to find better algorithms for the Min-PC problem on trees of rings with constant degrees. Our results imply a 3-approximation algorithm for the Min-RPC problem on a tree of rings. To our best knowledge, this is the first 3-approximation algorithm for this problem without using the cut-one-link strategy. We are not aware of any algorithm with performance ratio better than 3 for the Min-RPC problem on trees of rings, even when the tree of rings has bounded degree. It would be challenging to break this barrier. Our $3L$ algorithm also implies a $6L$ algorithm for the Min-PC problem on directed trees of rings with two directed links, one in each direction, between a pair of adjacent nodes. It is interesting to improve the approximation ratio for the Min-PC problem on directed trees of rings.
Chapter 4

Call Control and Maximum Path Coloring

The goal of the call control problem is to accommodate a maximum number of call requests subject to the bandwidth constraint of the links in the networks. The maximum path coloring problem, on the other hand, is to accept a maximum subset of paths that can be colored by a given number of colors. The call control problem with unit link capacity coincides with the maximum path coloring problem with one available color. Multifiber optical networks have multiple parallel fibers per link. For the path coloring problem in multifiber optical networks, each set of paths colored by a same color has maximum load bounded by the number of fibers on each link and is a feasible solution for the call control problem in which the capacity of an edge is equal to the number of fibers on that edge. The call control problem and the path multicoloring problem are closely related. In this chapter, we study these problems in tree and ring networks. We first study the call control problem in trees in Section 4.1. Then we study the path multicoloring problem in trees in Section 4.2. We study the maximum routing and path coloring problem in rings in Section 4.3.

4.1 Call Control in Bounded Depth Trees

In this section, we study the call control problem on bounded depth trees which are well used topologies in communication networks. In Section 4.1.1, we show that the call control problem is NP-hard and MAX SNP-hard even in depth-2 trees with capacities 1 or 2. Our
proof is a straightforward revision on the reduction for proving the hardness results of depth-3 trees in [63]. In Section 4.1.2, we give a polynomial time algorithm for the call control problem in a special class of depth-2 trees called double-stars. These results suggest that depth-2 trees are a boundary topology for which the call control problem is in P or NP-hard, depending on the node degrees of the trees. We also give 2- and 3-approximation algorithms in Section 4.1.3 for the weighted call control problem on depth-2 and depth-3 trees, respectively. This improves the previous 4-approximation algorithm for the problem on those trees. We show that the call control problem in spiders can be solved optimally in Section 4.1.4. All of our algorithms depend on a subroutine which solves the following restricted weighted call control problem on arbitrary trees in polynomial time: Given a set P of paths in a tree with all paths contain a same node of the tree, find an admissible subset P' ⊆ P such that w(P') = \sum_{p_i \in P'} w_i is maximized. This subroutine is of independent interest and is given in Section 4.1.5. A key technique used in our algorithms is to convert the call control problem in trees to the problem of finding a maximum degree constrained subgraph in auxiliary graphs. In Section 4.1.6, we show that the weighted call control problem in any graphs can be solved optimally if all the paths have length at most 2.

We begin with some definitions. A rooted tree is a tree in which a node r is selected as the root. All trees in this section are rooted trees unless otherwise stated, and will be denoted by T. The level of a node v in a tree is the length of the path from v to the root r which has level 0. The depth of a tree is the maximum level among all nodes of the tree. A depth-i tree is a tree with the maximum level i. An edge with a level-i end-node and a level-(i + 1) end-node in a tree is called a level-(i + 1) edge. A depth-1 tree is also called a star and the root r of the depth-1 tree is called the center node of the star. A double-star is a depth-2 tree in which two nodes have degree greater than one and all other nodes have degree one (see Figure 4.1 (a) for an example).

4.1.1 Hardness of Call Control in Depth-2 Trees

Given three pairwise disjoint sets X, Y, Z, |X| = |Y| = |Z|, and a set S = \{(x_i, y_j, z_k) | x_i \in X, y_j \in Y, z_k \in Z\} of triples, the three-dimensional matching problem is to find the maximum number of disjoint triples (two triples are disjoint if they do not have a common element in any dimension). The three-dimensional matching problem is NP-hard and MAX SNP-hard, even if the number of occurrences of any element in X, Y or Z is bounded by a constant [76]. Garg et al. prove that the call control problem is NP-hard and MAX
SNP-hard for trees by reducing the three-dimensional matching problem to the call control problem on depth-3 trees with edge capacities 1 and 2 [63]. This work actually proves a stronger result: the call control problem is NP-hard and MAX SNP-hard for depth-3 trees with edge capacities 1 and 2.

We observe that the three-dimensional matching problem can be reduced to the call control problem on depth-2 trees with edge capacities 1 and 2.

Theorem 4.1.1 The call control problem is NP-hard and MAX SNP-hard for depth-2 trees with edge capacities 1 and 2.

Proof Given an instance $X, Y, Z, S$ of the three-dimensional matching problem, we first construct a depth-2 tree $T$ with root $r$. For every $x_i \in X$, $y_i \in Y$, and $z_i \in Z$, there are level-1 nodes $x_i$, $y_i$, and $z_i$ in $T$, respectively. Assume that each $x_i$ appears in $S p_i$ times. Then in $T$ each $x_i$ has $2p_i$ children $x_{i,a}$ and $x_{i,b}$ ($1 \leq l \leq p_i$). There are $3|X|$ level-1 nodes and $2|S|$ level-2 nodes in $T$. We assign edges $\{r, x_{i} \}$ ($1 \leq i \leq |X|$) capacity 2 and all other edges capacity 1. Figure 4.2 shows the construction of $T$. We number the occurrences of $x_i$ in $S$ arbitrarily from 1 to $p_i$, and the $l^{th}$ occurrence corresponds to the two level-2 nodes.

Figure 4.1: (a) A double-star, and (b) the double-star after pre-processing.

Figure 4.2: The depth-2 tree for the NP-hardness proof of the call control problem.
CHAPTER 4. CALL CONTROL AND MAXIMUM PATH COLORING

Next we construct a call control instance $P$ on $T$. If $(x_i, y_j, z_k) \in S$ is the $l^{th}$ occurrence of $x_i$, we include three call requests, $\{x_{i,l,a}, x_{i,l,b}\}$, $\{x_{i,l,a}, y_j\}$, and $\{x_{i,l,b}, z_k\}$ into $P$. Notice that $P$ has $3|S|$ call requests.

For any disjoint subset $S' \subseteq S$, we construct an admissible subset $P' \subseteq P$ as follows. Let $(x_i, y_j, z_k)$ be the $l^{th}$ occurrence of $x_i$ in $S$. If $(x_i, y_j, z_k) \in S'$, then $P'$ contains $\{x_{i,l,a}, y_j\}$ and $\{x_{i,l,b}, z_k\}$, otherwise (i.e., $(x_i, y_j, z_k) \notin S'$), $P'$ contains $\{x_{i,l,a}, x_{i,l,b}\}$. Then $P'$ is an admissible set and $|P'| = |S'| + |S|$. Next we show that given an admissible subset $P' \subseteq P$ with $|P'| = t + |S|$, a disjoint subset $S' \subseteq S$ with $|S'| = t$ can be constructed. Let $P_1 \subseteq P$ be a maximum subset of paths with an end-node in the subtree rooted at $x_i$ that can be admitted. Then $P_i$ has $p_i + 1$ paths and contains the two paths for $\{x_{i,l,a}, y_j\}$ and $\{x_{i,l,b}, z_k\}$ for some $l$, and $p_i - 1$ paths for $\{x_{i,m,a}, x_{i,m,b}\}$ for $m \neq l, 1 \leq m \leq p_i$. Since $|P'| = t + |S|$, $P'$ has $t$ such subsets $P_i$. For every $P_i$, let $(x_i, y_j, z_k)$ be the corresponding triple. Let $S'$ be the set of triples $(x_i, y_j, z_k)$ corresponding to those $P_i$'s. Then each $x_i \in X$ appears in at most one triple of $S'$. From the capacities of edges $\{r, y_j\}$ and $\{r, z_k\}$, each $y_j \in Y$ or $z_k \in Z$ appears in at most one triple of $S'$. Thus, a maximum disjoint subset of $S$ can be computed if and only if a maximum admissible subset of $P$ can be computed.

Notice that our proof of Theorem 4.1.1 follows a similar argument of [63] where a depth-3 tree is used.

4.1.2 Call Control in Double-stars

From the previous section, we know that the call control problem is NP-hard in depth-2 trees. It is also known that the call control problem can be solved in polynomial time for depth-1 trees. Now we explore the subset of depth-2 trees for which the call control problem is in $P$. More specifically, we give polynomial time algorithms for the call control problem in double-stars.

Let $T$ be a double-star with two centers $v_0$ and $v_1$ (see Figure 4.1 (a)). If edge $e_0 = \{v_0, v_1\}$ has a constant capacity then the call control problem on $T$ can be solved by a rather straightforward enumeration approach: Given a set $P$ of paths on $T$, the number of paths on $e_0$ in any admissible subset $P' \subseteq P$ is at most $c(e_0)$. Since $c(e_0)$ is a constant, we can enumerate in polynomial time all possible subsets $Q$ such that $Q$ has at most $c(e_0)$ paths on $e_0$. For each enumerated subset $Q$, we find a maximum admissible subset $Q_i$ of $\{p | p \in P \setminus Q, p$ is not on $e_0\}$ in the star with center $v_i$ ($i = 0, 1$). Then the maximum set
$Q_0 \cup Q_1 \cup Q$ over all $Q$'s is a maximum admissible subset $P'$ of $P$ in $T$. The overall running time is polynomial in the input parameters. Similarly, if $c(e_0)$ is arbitrarily large but there are at most $O(\log n)$ paths of length three in $P$ then we can solve the call control problem optimally by first enumerating all possible subsets of length-3 paths on $e_0$ in any optimal solution, in $2^{O(\log n)} = O(n^{c})$ time for some constant $c > 0$, and then solving two call control problems in the two stars.

The enumeration approach does not give a polynomial time algorithm for the double-stars if $c(e_0)$ is arbitrarily large and $P$ has more than $O(\log n)$ length-3 paths. The difficulty lies in how to choose the length-3 paths in an optimal solution. We now give Algorithm ALG4.1 which solves the call control problem in double-stars in polynomial time. For simplicity, we do the following pre-processing. If there is a set $Q$ of paths with $v_0$ as an end-node, then we create a new leaf node $v'_0$ and a new edge $\{v_0, v'_0\}$ with capacity $|Q|$, and extend the paths in $Q$ to $v'_0$ (see Figure 4.1 (b)). We do a similar pre-processing for node $v_1$. It is easy to see that after the pre-processing, all end-nodes of every path are leaf nodes in $T$, there are only length-2 and length-3 paths, and no length-2 path contains both $v_0$ and $v_1$. From now on, we use $P$ to denote the set of pre-processed paths. We also do the following pre-processing on the capacities: $c(e) := \min\{c(e), L(e)\}$. This does not affect the solution, but helps to reduce the time complexity.

Given a double-star $T$ with centers $v_0$ and $v_1$, let $T_i$ ($i = 0, 1$) be the star with center $v_i$ obtained by removing edge $e_0 = \{v_0, v_1\}$. Edges of $T_i$ have the same capacities as the corresponding edges in $T$. We define $E_0 = E(T_0)$ and $E_1 = E(T_1)$. Let $P_i = \{p | p \in P, p$ is on edges of $T_i$ only$\}$ and $OPT_i$ be a maximum admissible subset of $P_i$ in $T_i$ ($i = 0, 1$). $OPT_i$ can be computed using the algorithm of [63]. Notice that there are only length-2 paths in $OPT_i$. Let $OPT$ be a maximum admissible subset of $P$ in $T$. Then

$$|OPT_0'| + |OPT_1'| \leq |OPT| \leq \min \left\{ |OPT_0'| + |OPT_1'| + c(e_0), \frac{1}{2} \sum_{e \in E_0 \cup E_1} c(e) \right\}. \quad (4.1)$$

We define $OPT_i \subseteq OPT$ to be the subset of paths in $OPT$ using only edges in $T_i$ ($i = 0, 1$). Notice that $OPT_i$ is not necessarily a subset of $OPT_i$.

In Algorithm ALG4.1, we first perform the pre-processing described above, then transform the call control problem in $T$ to a maximum weight DCS problem in an auxiliary graph $H$, and finally show that an optimal solution of the DCS problem can be converted to an optimal solution for the call control problem.
CHAPTER 4. CALL CONTROL AND MAXIMUM PATH COLORING

The auxiliary graph $H$ is constructed as follows. For each edge $e$ in $E_0 \cup E_1$, we create a node $u_e$ in $H$ with $b_1(u_e) = b_2(u_e) = c(e)$. For each length-2 path on $e_i, e_j \in E_0$ or $e_i, e_j \in E_1$, we create an edge $\{u_{e_i}, u_{e_j}\}$ in $H$ ($H$ is a multigraph in general, see Figure 4.3). These edges are shown as solid edges in Figure 4.3. For each length-3 path on a leaf edge $e_i \in E_0$ and a leaf edge $e_j \in E_1$, we create an edge $\{u_{e_i}, u_{e_j}\}$ in $H$. These edges are shown as dashed edges in Figure 4.3. We create one additional node $u$ in $H$, and set $b_1(u) = b_2(u) = \sum_{e \in E_0} c(e) + \sum_{e \in E_1} c(e) - 2 \cdot g$, where $g$ is an integer between $|OPT_0| + |OPT_1|$ and $\min\{|OPT_0| + |OPT_1| + c(e_0), \frac{1}{2} \sum_{e \in E_0 \cup E_1} c(e)\}$. We create $c(e)$ parallel edges $\{u, u_e\}$, for each $e_i \in E_0 \cup E_1$. These edges are shown as the dash-dotted edges in Figure 4.3. We give each edge $\{u_{e_i}, u_{e_j}\}$ with $e_i \in E_0$ and $e_j \in E_1$ a weight of $1 - \epsilon$, for some small positive $\epsilon$, say $\epsilon = 1/|P|$, and give all other edges a weight of 1. Notice that $H$ is in general a multigraph: if $c(e) > 1$ or there are two paths with the same end-nodes in $T$, then edges may represent multiple parallel edges in Figure 4.3(b).

Algorithm ALG4.1 finds a maximum weight DCS $M$ in $H$ (if there exists one), using the algorithm of [59], for every possible values of $g$ between $|OPT_0| + |OPT_1|$ and $\min\{|OPT_0| + |OPT_1| + c(e_0), \frac{1}{2} \sum_{e \in E_0 \cup E_1} c(e)\}$. For each found $M$, Algorithm ALG4.1 checks if the set of paths corresponding to the edges of $M$ is admissible. As shown later, a maximum admissible set $P'$ can be obtained from a maximum weight DCS $M$ for $g = |OPT|$.

**Theorem 4.1.2** Algorithm ALG4.1 solves the call control problem in double-stars in polynomial time.

**Proof** The sum of the capacities of the nodes in $H$, excluding $u$, is $\sum_{e \in E_0} c(e) + \sum_{e \in E_1} c(e)$. Since each edge in $M$ is incident to two nodes of $H$ and $b_2(u)$ edges of $M$ are incident to node $u$, the number of edges $\{u_{e_i}, u_{e_j}\}$ with $e_i, e_j \in E_0 \cup E_1$ that can be included in any DCS $M$ is exactly

$$\sum_{e \in E_0} c(e) + \sum_{e \in E_1} c(e) - b_2(u) = \frac{2g}{2} = g.$$ 

Assume that there is an $OPT$ which has $k \leq c(e_0)$ length-3 paths. Then for $g = |OPT| = |OPT_0| + |OPT_1| + k$, there is a DCS $M$ in $H$ such that $M$ has exactly $|OPT_0| + |OPT_1| + k$ edges $\{u_{e_i}, u_{e_j}\}$ with $e_i, e_j \in E_0$ or $e_i, e_j \in E_1$, $k$ edges $\{u_{e_i}, u_{e_j}\}$ with $e_i \in E_0$ and $e_j \in E_1$, and $b_2(u)$ edges $\{u, u_e\}$ with $e_i \in E_0 \cup E_1$. The weight of this $M$ is $w(M) = \sum_{e \in E_0 \cup E_1} c(e) - |OPT| - ke$. For any DCS $M'$ of $H$ with $l > k$ edges $\{u_{e_i}, u_{e_j}\}$ with $e_i \in E_0$ and $e_j \in E_1$, the weight of $M'$ is $w(M') = \sum_{e \in E_0 \cup E_1} c(e) - |OPT| - le$ which is smaller than $w(M)$. 
Figure 4.3: (a) An instance of the call control problem in a double-star $T$ with $c(e) = 1$ for every $e \in E(T)$, and (b) an instance of the DCS problem in an auxiliary graph $H$.

Therefore, any maximum weight DCS must contain at most $k$ edges $\{u_{e_i}, u_{e_j}\}$ with $e_i \in E_0$ and $e_j \in E_1$. Let $P'$ be the set of paths in $P$ corresponding to the edges in a maximum weight DCS in $H$. Then $P'$ contains at most $k \leq c(e_0)$ length-3 paths and does not violate the capacity constraint of any leaf edge of $T$ either. So $P'$ is an admissible subset of $P$ in $T$. Since $P'$ contains exactly $|OPT_0| + |OPT_1| + k = |OPT|$ paths, $P'$ is a maximum admissible subset.

From Inequality (4.1) and the fact that $ALG4.1$ computes a maximum weight DCS of $H$ for every possible value of $g$ between $|OPT_0'| + |OPT_1'|$ and $\min\{|OPT_0'| + |OPT_1'| + c(e_0), \frac{1}{2} \sum_{e \in E_0 \cup E_1} c(e)\}$, and checks the corresponding subset of $P$, $ALG4.1$ finds a maximum admissible subset $P' \subseteq P$ in $T$.

The auxiliary graph $H$ has $|E(T)|$ nodes, and less than $(|P| + |E(T)|)$ edges. The maximum edge multiplicity of $H$ is bounded by the load $L$ of $P$ in $T$. Thus, a maximum weight DCS in $H$ can be found in $O((|P| + |E(T)|)^2(\log |E(T)|)(\log L))$ time. The total running time of $ALG4.1$ is $O((|P| + |E(T)|)^2(\log |E(T)|)(L \log L))$, since the maximum weight DCS algorithm is called $c(e_0) \leq L$ times.

The complexity of the weighted call control problem in double-stars is not known. It seems that our method above cannot be used for the weighted case, since we already assign weights to edges in the graph $H$ constructed in the (unweighted) call control problem and it is not clear how to assign weights to edges in the weighted call control problem. Nevertheless, we show in Section 4.1.3 that the weighted call control problem in depth-2 trees (thus in double-stars) can be approximated with ratio two.

\[\square\]
4.1.3 Weighted Call Control in Depth-2 and Depth-3 Trees

The weighted call control problem can be solved optimally in depth-1 trees (stars) [63], and it can be approximated with a ratio of four in arbitrary trees [41]. The algorithm of [41] does not seem to have a performance ratio better than four when applied to bounded depth trees. In this section, we show that the weighted call control problem can be approximated with ratios two and three in depth-2 and depth-3 trees, respectively. Recall that the (unweighted) call control problem is already NP-hard in depth-2 trees, and can be approximated with a ratio of two. Our algorithms use the optimal weighted call control algorithm in stars [63], and an algorithm developed in Section 4.1.5 that solves in polynomial time the weighted call control problem for the instances in which all paths contain the root in arbitrary trees. This algorithm is of independent interest since it optimally solves the weighted call control problem in arbitrary trees for a restricted class of instances.

Given a set $P$ of paths in a depth-2 or depth-3 tree $T$, we pre-process the paths as follows. For every internal node $u$ of $T$, if there is a set $Q$ of paths with $u$ of $T$ as an end-node, then we create a new leaf node $u'$ and a new edge $\{u, u'\}$ with capacity $|Q|$, and extend the paths in $Q$ to $u$. The pre-processing does not change the depth of $T$ or the value of the optimal solution. Now we assume that the end-nodes of every path of $P$ are leaf nodes of $T$. Let $r$ be the root of $T$.

We first give Algorithm $ALG4.2$ for the weighted call control problem in a depth-2 tree $T$. The algorithm works as follows.

1. Let $P_1 \subseteq P$ be the subset of paths such that each path of $P_1$ contains root $r$. Find a maximum admissible subset of $P_1$ in $T$ using the algorithm $ALG4.5$ described later, and denote the solution by $SOL_1$.
2. Find a maximum admissible subset of $P_2 = P \setminus P_1$ of remaining paths (that does not contain $r$) in $T$, and denote the solution by $SOL_2$.
3. Output the set of $SOL_1$ or $SOL_2$ which has the maximum weight as the final solution $SOL$.

For the weighted call control problem in a depth-3 tree $T$, we give a 3-approximation algorithm $ALG4.3$ as follows.

1. Let $P_1 \subseteq P$ be the subset of paths such that each path of $P_1$ contains root $r$. Find a
maximum admissible subset of $P_1$ in $T$ using the algorithm ALG4.5 described later, and denote the solution by $SOL_1$.

2. Find a maximum admissible subset of $P_2 = P \setminus P_1$ of remaining paths (that does not contain $r$) in $T$ using Algorithm ALG4.2, and denote the solution by $SOL_2$.

3. Output the set of $SOL_1$ or $SOL_2$ which has the maximum weight as the final solution $SOL$.

**Theorem 4.1.3** ALG4.2 and ALG4.3 are 2-approximation and 3-approximation algorithms for depth-2 and depth-3 trees, respectively.

**Proof** Let $P_1$ be the subset of $P$ such that each path of $P_1$ contains root $r$ of $T$. We first assume that a maximum admissible subset of $P_1$ in $T$ can be computed in polynomial time. By this assumption, we get $SOL_1$ in Step 1 of ALG4.2. In Step 2, the set $P_2$ of paths can be partitioned into several subsets of paths, with each subset of paths on a star obtained by removing $r$ and the edges incident to $r$ in the depth-2 tree. Thus, the maximum admissible subset of $P_2$ can be found in polynomial time by the algorithm in [63]. Let $OPT$ be a maximum admissible subset of $P$ in $T$, $OPT_1 = OPT \cap P_1$ and $OPT_2 = OPT \cap P_2$. Then $OPT_1 \cap OPT_2 = \emptyset$, $OPT_1 \cup OPT_2 = OPT$, $w(OPT_1) \leq w(SOL_1)$ and $w(OPT_2) \leq w(SOL_2)$. Thus,

$$w(OPT) = w(OPT_1) + w(OPT_2) \leq w(SOL_1) + w(SOL_2) \leq 2 \max\{w(SOL_1), w(SOL_2)\} \leq 2w(SOL),$$

and Algorithm ALG4.2 has an approximation ratio of 2.

Similarly, let $OPT$ be a maximum admissible subset of $P$ in the depth-3 tree $T$. Then $w(OPT_1) \leq w(SOL_1)$. The set $P_2$ of paths can be partitioned into several subsets of paths, with each subset of paths on a depth-2 tree obtained by removing $r$ and the edges incident to $r$ in the depth-3 tree. Thus, $w(OPT_2) \leq 2w(SOL_2)$, since ALG4.2 is used to find $SOL_2$ and it is a 2-approximation algorithm. Therefore,

$$w(OPT) = w(OPT_1) + w(OPT_2) \leq w(SOL_1) + 2w(SOL_2) \leq 3 \max\{w(SOL_1), w(SOL_2)\} \leq 3w(SOL),$$

and ALG4.3 has an approximation ratio of 3.

The assumption that a maximum admissible subset of $P_1$ in $T$ can be computed in polynomial time is shown by Theorem 4.1.8 in Section 4.1.5. This completes the proof. \(\square\)
4.1.4 Call Control in Spiders

In this section, we study the call control problem in spiders. The result will be used in Section 4.2.2 to develop algorithms for the Max-PMC problems in spiders. We first give some more definitions. Let \( P \) be a set of paths in a spider \( G \), and \( c : E(G) \to \mathbb{N} \) be a capacity function. We say a subset \( Q \subseteq P \) of paths is feasible if \( L_Q(e) \leq c(e) \) for all \( e \in E(G) \). Notice that any path in \( G \) can have edges in at most two legs of \( G \). A path is called a short (resp. long) path if it is on edges of only one leg (resp. two legs). Let \( P_S \) (resp. \( P_L \)) be the set of short (resp. long) paths of \( P \). For the convenience of description, we draw \( G \) as shown in Figure 4.4. The legs of \( G \) are numbered from 1 to \( \Delta \). For each leg \( l \) with \( n_l + 1 \) nodes, the node set and edge set are defined as \( V^l = \{v_0, v_i \mid 1 \leq i \leq n_l\} \) and \( E^l = \{e_i^l \mid e_i^l = (v_0, v_i^l), e_i^l = (v_i^l, v_{i+1}^l), 2 \leq i \leq n_l\} \). For simplicity, we occasionally use \( v_0^l \) for \( v_0 \). Node \( v_i^l \) is called the right-node of \( e_i^l \) and node \( v_{i-1}^l \) is called the left-node of \( e_i^l \). Similarly, we can define the left-node and right-node of short paths. For a long path on legs \( l \) and \( m \), we will also use “left-nodes” to denote its endpoints on legs \( l \) and \( m \). For \( P \) in \( G \) and \( 1 \leq l \leq \Delta \), let \( P_l \subseteq P \) be the set of paths, each of which is on an edge in leg \( l \). Let \( P_S^l = P_l \cap P_S \).

The following algorithm ALG4.4 is used to solve the call control problem in spiders.

1. For each leg \( l \) of \( G \), process the nodes from \( v_{n_l-1}^l \) to \( v_0 \). When we process node \( v_i^l \), we check the paths of \( P_S^l \) with right-node \( v_i^l \) in an arbitrary order, and accept as many of these paths as possible subject to capacities \( c(e) \). The processing for leg \( l \) is finished when node \( v_0 \) is processed. Let \( SOL_S \) be the set of paths accepted in processing all legs of \( G \).
2. For each edge $e$ of $G$, let $c'(e) = c(e) - L_{SOL_S}(e)$ be the new capacity function. Find a maximum cardinality subset $SOL_L$ of $P_L$ subject to new capacities $c'(e)$, using Algorithm ALG4.5 described later.

3. Output $SOL_S \cup SOL_L$ as the final solution.

In order to prove that Algorithm ALG4.4 gives an optimal solution for the call control problem, we need to prove the following two claims: (1) The two steps can both be done optimally in polynomial time, and (2) the two-step approach gives an optimal solution for the original problem.

By the following proposition, the first step can be done optimally in polynomial time.

Proposition 4.1.4 [5] The call control problem can be solved optimally in polynomial time for chains.

This result is a generalization of the algorithm for $k$-coloring of interval graphs [38]. The greedy algorithm for chains processes the nodes of a chain one by one from left to right. When processing node $v$, the paths with right-node $v$ can be included into $SOL_S$ in an arbitrary order (subject to the capacity constraint), and the solution is still optimal. This property is critical to the correctness of Algorithm ALG4.4.

The second step can be done optimally in polynomial time, i.e., the call control problem can be solved optimally for spiders if all the paths are long paths. This will be shown in Section 4.1.5.

The two-step algorithm

We now show that the two-step algorithm ALG4.4 gives an optimal solution for the call control problem in spiders.

Theorem 4.1.5 The solution $SOL_S \cup SOL_L$ of Algorithm ALG4.4 is optimal for the call control problem in spiders.

Proof For a given set $P = P_S \cup P_L$ of paths in a spider $G$, where $P_S$ is the set of short paths and $P_L$ is the set of long paths, let $OPT$ be an optimal solution, $OPT_S$ be the set of short paths in $OPT$, and $OPT_L$ be the set of long paths in $OPT$. Then $|OPT| = |OPT_S| + |OPT_L|$, and $|OPT_S| \leq |SOL_S|$. Let $Q = SOL_S \cup OPT_L$. Recall that $c(e)$ is the capacity of edge $e$ for the call control problem and $c'(e) = c(e) - L_{SOL_S}(e)$ is the new capacity of $e$ in Algorithm
ALG4.4. If $L_Q(e) \leq c(e)$ for every $e \in E(G)$ then $|OPT_L| \leq |SOL_L|$, since $OPT_L$ is a subset of $P_L$ with $L_{OPT_L}(e) \leq c'(e)$ and $SOL_L$ is the maximum cardinality subset of $P_L$ with $L_{SOL_L}(e) \leq c'(e)$ for $e \in E(G)$. In this case,

$$|OPT| = |OPT_S| + |OPT_L| \leq |SOL_S| + |SOL_L| = |SOL|,$$

implying $SOL$ is optimal and we are done.

Assume that $L_Q(e_t) > c(e_t)$ for some edge $e_t$ in leg $l$. We show that a new $OPT'$ with $|OPT'| \geq |OPT|$ can be obtained from $OPT$ by replacing $OPT_S$ with $SOL_S$ and replacing $OPT_L$ with a subset of $OPT_L$, for every such a leg $l$, such that $SOL_S \cup OPT_L$ is a feasible solution. From the argument above, this implies $|OPT'| \leq |SOL|$. For a long path $p \in P_L$ on an edge of leg $l$, let $p(l)$ be the segment of $p$ from $u_0$ to the endpoint of $p$ in leg $l$. Define $Q_l = \{p(l) \mid p \in OPT_L\}$. Let $Q'_l$ be the maximum cardinality subset of paths in $Q_l$ which can be accepted on leg $l$ subject to the new capacity function $c'$. Then $|Q'_l| < |Q_l|$. Let $OPT'_S$ (resp. $P'_S$) be the subset of paths in $OPT_S$ (resp. $P_S$) which is on edges of leg $l$. Then $SOL'_S \cup Q'_l$ is the maximum cardinality subset of paths in $P'_S \cup Q_l$ that can be accepted subject to the capacity function $c$ (see the remarks following Proposition 4.1.4). $OPT'_S \cup Q_l$ is a feasible subset of paths in $P'_S \cup Q_l$ subject to the capacity function $c$. Thus, $|OPT'_S \cup Q_l| \leq |SOL'_S \cup Q'_l|$. In $OPT$, we replace $OPT'_S$ by $SOL'_S$, replace $OPT'_L$ by $\{p \mid p \in OPT'_L, p(l) \in Q'_l\}$, and use $OPT'$ to denote the set of paths obtained. Then $|OPT| \leq |OPT'|$ and $L_{OPT'}(e_j^m) \leq c(e_j^m)$ ($1 \leq m \leq \Delta$, $1 \leq j \leq n_m$). However, $OPT'$ has the additional property that for $Q' = SOL_S \cup OPT'_L$, $L_Q(e_j^l) \leq c(e_j^l)$ ($1 \leq j \leq n_l$).

Rename $OPT'$ as the new $OPT$, and perform the above procedure as long as $SOL_S \cup OPT_L$ is not feasible. This process terminates after at most $\Delta(G)$ rounds (since $G$ has $\Delta(G)$ legs). At this point, $SOL_S \cup OPT_L$ is feasible, and we have proved that the solution produced by algorithm ALG4.4 is optimal.

4.1.5 Call Control in Trees with Central Paths

We give Algorithm ALG4.5 which solves the weighted call control problem optimally in an arbitrary tree $T$, if all the paths contain a same node of $T$. Suppose the tree $T$ is rooted at node $r$, and all the paths in $P$ contain $r$ (we call such paths central paths). After proper pre-processing as in previous sections, we assume that no path in $P$ contains $r$ as an end-node. We use $T_v$ to denote the subtree rooted at a node $v \in V(T)$. For a node $v \in V(T) \setminus \{r\}$, the unique neighbor of $v$ whose level is one smaller than that of $v$ is called the parent $p(v)$.
of \( v \), and we use \( e^v_0 \) to denote the edge \( \{v, p(v)\} \). For a non-leaf node \( v \), all neighbors of \( v \) whose level are one larger than that of \( v \) are called the children of \( v \), and we use \( e^v_1, \ldots, e^v_d \) to denote the edges between \( v \) and its children (assuming \( v \) has \( d = \delta(v) - 1 \) children).

An important observation is that for a non-leaf node \( v \in V(T) \setminus \{r\} \), a path on any edge \( e^v_i \) \((1 \leq i \leq d)\) must be on edge \( e^v_0 \) as well. Let \( Q \subseteq P \) be a set of central paths. Then \( \sum_{i=1}^{d} L_Q(e^v_i) \leq L_Q(e^v_0) \). Similarly, we have \( \sum_{i=1}^{d} L_{P \setminus Q}(e^v_i) \leq L_{P \setminus Q}(e^v_0) \). If \( \sum_{i=1}^{d} L_{P \setminus Q}(e^v_i) \geq L(e^v_0) - c(e^v_0) \), then \( L_Q(e^v_0) \leq c(e^v_0) \) for every \( e^v_0 \) because

\[
L_Q(e^v_0) = L(e^v_0) - L_{P \setminus Q}(e^v_0) \leq L(e^v_0) - \sum_{i=1}^{d} L_{P \setminus Q}(e^v_i) 
\leq L(e^v_0) - (L(e^v_0) - c(e^v_0)) = c(e^v_0).
\]

On the other hand, if \( \sum_{i=1}^{d} L_{P \setminus Q}(e^v_i) < L(e^v_0) - c(e^v_0) \), then at least \( L(e^v_0) - c(e^v_0) - \sum_{i=1}^{d} L_{P \setminus Q}(e^v_i) \) paths on \( e^v_0 \) (but not on \( e^v_i \), \( 1 \leq i \leq d \)) cannot be included in \( Q \) if \( L_Q(e^v_0) \leq c(e^v_0) \). These observations are essential to our algorithm.

We give Algorithm ALG4.5 which solves optimally the call control problem in trees with only central paths. We reduce the call control problem for the set \( P \) of paths containing \( r \) in \( T \) to the DCS (degree constrained subgraph) problem in an auxiliary graph \( H \) constructed below. Let \( v \) be the \( l \)th child of \( r \). We use \( E_l \) to denote \( \{e^r_l\} \cup E(T_v) \) \((1 \leq l \leq \delta(r))\). Clearly any path of \( P \) is on exactly two children of \( r \). For each path \( p \in P \) on the \( l \)th and \( m \)th children of \( r \), we create two nodes \( y^l_p \) and \( y^m_p \), and an edge \( e_p = \{y^l_p, y^m_p\} \) in \( H \). These nodes and edges are called path-nodes and path-edges, respectively. For each path-node \( y^l_p \), we set \( b_1(y^l_p) = 0 \) and \( b_2(y^l_p) = 1 \). For every edge \( e \in E_l \) \((1 \leq l \leq \delta(r))\), we create in \( H \) a node \( u(e) \). We create an edge \( \{u(e), y^l_p\} \) if path \( p \) is on edge \( e \in E_l \). Nodes \( u(e) \) and edges \( \{u(e), y^l_p\} \) are called aux-nodes and aux-edges, respectively. We set the capacities of the aux-nodes in the following order. The capacity of an aux-node corresponding to a non-leaf edge \( e^v_0 \in E(T) \) is set only if the capacities of all aux-nodes \( \{u(e)|e \in E(T_v)\} \) have been set. For a leaf node \( v \) in \( T \), the aux-node corresponding to the leaf edge \( e^v_0 \) is \( u(e^v_0) \), and

\[
b_1(u(e^v_0)) = b_2(u(e^v_0)) = \max\{L(e^v_0) - c(e^v_0), 0\},
\]

and for any non-leaf node \( v \in V(T) \setminus \{r\} \),

\[
b_1(u(e^v_0)) = b_2(u(e^v_0)) = \max\{L(e^v_0) - c(e^v_0) - \sum_{e \in E(T_v)} b_2(u(e)), 0\}.
\]

We find a maximum cardinality DCS in the constructed graph \( H \). We prove that the set of the paths corresponding to the path-edges in the maximum cardinality DCS is an optimal
solution of the call control problem for $P$ in $T$. Let $OPT$ be an optimal solution for the call control problem in $T$.

**Lemma 4.1.6** There exists a DCS of cardinality $|OPT| + \sum_{e \in E(T)} b_2(u(e))$ in $H$.

**Proof** Given an optimal solution $OPT$ in $T$, a DCS $M$ in $H$ can be constructed as follows. We first include into $M$ all the path-edges corresponding to the paths in $OPT$. Since each path-node is incident to only one path-edge, $b_1(v) \leq \delta_M(v) \leq b_2(v)$ for all nodes of $M$. Next, we process every aux-node $u(e)$ ($e \in E(T)$) to include into $M$ $b_2(u(e))$ aux-edges incident to aux-node $u(e)$. The processing order of $u(e)$ ($e \in E(T)$) is based on the order of the nodes in $V(T)$. Again, an aux-node corresponding to a non-leaf edge $e_o \in E(T)$ is processed only if all aux-nodes $\{u(e) | e \in E(T_v)\}$ have been processed. For a leaf node $v \in V(T)$, since $L_{OPT}(e_o^v) \leq c(e_o^v)$, there are at least $\max\{L(e_o^v) - c(e_o^v), 0\}$ path-nodes adjacent to $u(e_o^v)$ that are not matched by the edges of $M$ prior to the processing of $u(e_o^v)$. So $b_2(u(e_o^v))$ aux-edges incident to $u(e_o^v)$ can be included into $M$ such that $b_1(v) \leq \delta_M(v) \leq b_2(v)$ for all nodes of $M$ after processing $u(e_o^v)$. Similarly, for any non-leaf node $v \in V(T) \setminus \{r\}$, at least

$$b_2(u(e_o^v)) = \max\{L(e_o^v) - c(e_o^v) - \sum_{e \in E(T_v)} b_2(u(e)), 0\}$$

path-nodes adjacent to $u(e_o^v)$ are not matched by the edges of $M$ prior to the processing of $u(e_o^v)$. Therefore, $b_2(u(e_o^v))$ edges incident to $u(e_o^v)$ can be included into $M$ such that $b_1(v) \leq \delta_M(v) \leq b_2(v)$ holds for all nodes of $M$ after processing $u(e_o^v)$. When all aux-nodes are processed, the constructed DCS $M$ satisfies $b_1(v) \leq \delta_M(v) \leq b_2(v)$ for every $v \in V(H)$ and has $|OPT|$ path-edges. The cardinality of $M$ is

$$|OPT| + \sum_{e \in E(T)} b_2(u(e)).$$

**Lemma 4.1.7** Let $SOL$ be the set of paths corresponding to path-edges in any DCS of $H$. Then $SOL$ is a feasible solution for the call control problem in $T$.

**Proof** We show that $L_{SOL}(e) \leq c(e)$ holds for every $e \in E(T)$. Let $M$ be the given DCS in $H$. Then $\delta_M(u(e)) = b_1(u(e)) = b_2(u(e))$ for $e \in E(T)$. For any leaf node $v$ of $T$, $b_1(u(e_o^v)) = b_2(u(e_o^v)) = \max\{L(e_o^v) - c(e_o^v), 0\}$ and at least $\max\{L(e_o^v) - c(e_o^v), 0\}$ paths on
edge $e_0^v$ are not included in $SOL$. From this, $L_{SOL}(e_0^v) \leq c(e_0^v)$ holds. For any non-leaf node $v \in V(T) \setminus \{r\}$, if $\sum_{e \in E(T_v)} b_2(u(e)) \geq L(e_0^v) - c(e_0^v)$ ($b_2(u(e_0^v)) = 0$), then at least $L(e_0^v) - c(e_0^v)$ paths on edges in $E(T_v)$ are not included in $SOL$, and $L_{SOL}(e_0^v) \leq c(e_0^v)$ holds (see Inequality (4.2)). Otherwise, $b_2(u(e_0^v)) = L(e_0^v) - c(e_0^v) - \sum_{e \in E(T_v)} b_2(u(e))$ and there are $b_2(u(e_0^v)) + \sum_{e \in E(T_v)} b_2(u(e)) = L(e_0^v) - c(e_0^v)$ path-nodes of $\{y_p^e|p \in e_0^v, p \in P\}$ which are matched by aux-edges incident to aux-nodes $\{u(e)|e \in E(T_v) \cup \{e_0^v\}\}$. From this, there can be at most $c(e_0^v)$ path-edges in $M$ whose corresponding paths are on edge $e_0^v$. This implies $L_{SOL}(e_0^v) \leq c(e_0^v)$. Thus $SOL$ is feasible.

**Theorem 4.1.8** There is an optimal polynomial time algorithm for the call control problem in an arbitrary tree if the set $P$ of paths contains a same node of the tree.

**Proof** Any DCS contains at most $\sum_{e \in E(T)} b_2(u(e))$ aux-edges and at most $|OPT|$ path-edges (by Lemma 4.1.7). Thus, a maximum cardinality DCS contains exactly $\sum_{e \in E(T)} b_2(u(e))$ aux-edges and $|OPT|$ path-edges (Lemma 4.1.6). Let $SOL$ be the set of paths corresponding to the path-edges in the maximum cardinality DCS. Then $SOL$ contains $|OPT|$ paths. According to Lemma 4.1.7, $SOL$ is feasible. Thus, $SOL$ is optimal. Since a maximum cardinality DCS can be found in polynomial time, the theorem holds.

Algorithm ALG4.5 can be extended to work for the weighted call control problem in trees with only central paths. We make the following modifications to the above construction. Each path-edge $e_p$ in $H$ is assigned a weight equal to the weight of the corresponding path $p$ in $T$. Each aux-edge is assigned a very small positive weight $\epsilon$, where $\epsilon < \min_{p \in P} w(p)/(|P|+\sum_{e \in E(T)} b_2(u(e)))$. We can then show that the path-edges in a maximum weight DCS in $H$ correspond to the paths in an optimal solution for the weighted call control problem in $T$.

### 4.1.6 Call Control with Length-2 Paths

From the proof of the NP-hardness and MAX SNP-hardness for the call control problem in depth-2 trees, we can see that the call control problem is NP-hard and MAX SNP-hard even if the path length is restricted to at most 3. On the other hand, we show that the weighted call control problem can be solved optimally in any graphs, if the path length is restricted to at most 2, even if the edge capacities are arbitrary. Let $G = (V, E)$ be the input graph for the call control problem and $c(e)$ be the capacity of an edge $e \in E(G)$. Without loss of generality, we may assume that all paths have length exactly 2, after proper pre-processing.
We give an algorithm which optimally solves the weighted call control problem when all the
paths have length 2.

1. Construct a multigraph $H$ as follows. For each edge $e \in E(G)$, construct a node $u_e$
in $H$, and let $b_1(u_e) = 0$ and $b_2(u_e) = c(e)$. For any path $p$ on edges $e_1$ and $e_2$ in $G$,
construct an edge $e_p = \{u_{e_1}, u_{e_2}\}$ in $H$, and give $e_p$ the same weight as $p$.

2. Find a maximum weight DCS $M$ in $H$. The paths in $G$ corresponding to the edges in
$M$ are taken as the solution for the weighted call control problem in $G$.

To see the above algorithm gives an optimal solution for the weighted call control problem
with paths length 2, we notice that for any optimal solution for the weighted call control
problem, there is a DCS with the same weight in $H$. On the other hand, for any DCS $M$ in $H$,
the corresponding paths in $G$ do not violate any edge capacity constraint, since the node
capacity constraint in $H$ translates to edge capacity constraint in $G$. Thus, a maximum
weight DCS in $H$ corresponds to an optimal solution for the weighted call control problem
in $G$.

4.1.7 Remarks

We have shown that the call control problem is NP-hard and MAX SNP-hard even in
depth-2 trees. We give polynomial time optimal algorithms for the call control problem in
double-stars which are special depth-2 trees, and for the call control problem in spiders. The
double-star has depth two but has only two nodes with degree greater than one. The spider
may have unbounded depth but only has one node with degree greater than two. These
results suggest that whether the call control problem is in P or NP-hard depends largely
on the node degree and depth of the trees. We have shown that the weighted call control
problem is optimally solvable in arbitrary trees if all the paths contain a same node of the
tree, while the weighted call control problem in any graphs can be solved optimally if all
the paths have length at most 2. The pattern of the paths may also play an important role
in the solvability of the call control problem.
4.2 Path Multicoloring in Multifiber Star and Spider Networks

In this section, we study the path multicoloring problem in WDM optical trees with multiple parallel fibers. We focus on the hardness of the PMC problems in stars and spiders in Section 4.2.1. Recall that the Min-PMC and Max-PMC problems in 1-fiber stars are NP-hard [48, 105]. For every even \( k > 1 \), the problems are known polynomial time solvable in \( k \)-fiber stars [87, 88]. A natural question here is whether there are efficient algorithms for the problems in \( k \)-fiber stars for every \( k > 1 \). We give a negative answer to this question by showing that for every odd integer \( k \geq 3 \), the Min-PMC and Max-PMC problems in \( k \)-fiber stars (and thus spiders) are NP-hard. These results are contrasted to the even \( k \) case. We give efficient algorithms for the Min-PMC problem in non-uniform stars with even number of fibers in every link and \( k \)-fiber (\( k \) even) spiders. The results above suggest that the evenness of the number of fibers plays an important role in the polynomial solvability of the problems. By the result for spiders of even fibers and the delete-one-fiber approach, we have a \((1 + \frac{1}{k-1})\)-approximation algorithm for the Min-PMC problem in \( k \)-fiber spiders for every odd \( k \geq 3 \).

We study the Max-PMC problems in Section 4.2.2. By using the algorithm for the Min-PMC problem in stars of even fibers as a subroutine, we get an efficient algorithm for the Max-PMC problem in non-uniform stars with even fibers in every link. We also give an efficient algorithm and a 1.58-approximation algorithm for the Max-PMC problem in \( k \)-fiber (\( k \) even) spiders and non-uniform spiders, respectively. The algorithms for spiders rely on an optimal algorithm for the call control problem which has been developed in Section 4.1.4.

4.2.1 Hardness of the PMC Problem in Stars

In this section, we prove that the path multicoloring problem in \( k \)-fiber stars (and thus spiders) is NP-hard for every odd \( k \geq 3 \). We also give efficient algorithms for the Min-PMC problem in the non-uniform multifiber stars with even number of fibers in every edge and the \( k \)-fiber (\( k \) even) spiders.
NP-hardness result

Efficient algorithms for the Min-PMC problem in \( k \)-fiber \((k \text{ even})\) stars have been known [87, 88]. When each edge of the star has one fiber, the Min-PMC problem becomes conventional path coloring problem and is NP-hard [48, 105]. Note that an efficient algorithm for the path coloring problem in stars with bounded maximum degree is known [48]. We show that the Min-PMC problem in \( k \)-fiber stars with unbounded maximum degree is NP-hard for every odd \( k \geq 3 \). We first give the proof for \( k = 3 \) and then generalize the proof to arbitrary odd \( k \). We reduce the decision version of the path coloring problem in single fiber stars to the decision version of the path multicoloring problem in 3-fiber stars. The NP-completeness of the decision problem implies that both the Min-PMC and the Max-PMC problems are NP-hard.

The decision version of the path coloring problem in single fiber stars can be stated as follows: Given a set \( P \) of paths in a single fiber star and an integer \( w > 0 \), is \( P \) \( w \)-colorable? The decision version of the path multicoloring problem in \( k \)-fiber stars can be defined similarly: Given a set \( P \) of paths in a \( k \)-fiber star and an integer \( w > 0 \), is \( P \) \( w \)-colorable?

**Theorem 4.2.1** The Min-PMC problem in 3-fiber stars is NP-hard.

**Proof** Let \( G_1 \) be the 3-fiber star with \( V(G_1) = \{v_i|0 \leq i \leq 3\} \) and \( E(G_1) = \{(v_0,v_i)|1 \leq i \leq 3\} \) (see Figure 4.5(a)). Let \( Q_1 \) be the set of \( 3w - 1 \) paths between \( v_2 \) and \( v_3 \), \( Q'_1 = \{q_2, q_3\} \), where \( q_2 \) is the path between \( v_1 \) and \( v_2 \) and \( q_3 \) is the path between \( v_1 \) and \( v_3 \), and \( P_1 = Q_1 \cup Q'_1 \). Obviously there is a valid \( w \)-coloring for \( P_1 \). On the other hand, in any valid \( w \)-coloring for \( P_1 \), each color must be used by exactly three paths on \((v_0,v_2)\) and three paths on \((v_0,v_3)\) since the load on each of the two edges is \( 3w \). By the definition of \( Q_1 \), there are \( w - 1 \) colors each of which is used by three of the \( 3w - 1 \) paths, and exactly one color (say \( \lambda \)) which is used by two of the \( 3w - 1 \) paths of \( Q_1 \). The paths \( q_2 \) and \( q_3 \) of \( Q'_1 \) can only be colored by \( \lambda \). Thus, in any valid \( w \)-coloring of \( P_1 \), paths \( q_2 \) and \( q_3 \) must be assigned the same color.

Let \( G_w \) be the 3-fiber star with \( V(G_w) = \{v_i|0 \leq i \leq 2w + 1\} \) and \( E(G_w) = \{(v_0,v_i)|1 \leq i \leq 2w + 1\} \) (see Figure 4.5(b)). \( G_w \) can be considered as the star obtained from \( w \) copies of \( G_1 \) by merging the \( w \) edges \((v_0,v_1)\) in the copies into one edge. Let \( Q_j \) be the set of \( 3w - 1 \) paths between \( v_{2j} \) and \( v_{2j+1} \) and \( Q'_j = \{q_{2j}, q_{2j+1}\} \) for \( 1 \leq j \leq w \), where \( q_{2j} \) is the path
between \( v_1 \) and \( v_{2j} \) and \( q_{2j+1} \) is the path between \( v_1 \) and \( v_{2j+1} \). Let \( P_w = \bigcup_{j=1}^{w}(Q_j \cup Q_j') \) in \( G_w \). Then it is easy to find a valid \( w \)-coloring for \( P_w \). By the analysis on \( G_1 \) above, in any valid \( w \)-coloring of \( P_w \), the paths in each \( Q_j' \) must be given the same color. For any pairs \( Q_j' \) and \( Q_j'' (1 \leq j_1 \neq j_2 \leq w) \), there are four paths in \( Q_j' \) and \( Q_j'' \), the four paths are on edge \( (v_0, v_1) \), and the edge has three fibers. Therefore, in any valid \( w \)-coloring of \( P_w \), the two paths in any \( Q_j' \) are assigned the same color and any two pairs \( Q_j' \) and \( Q_j'' (j_1 \neq j_2) \) are assigned different colors. This implies that each of the \( w \) colors is used by exactly one set of \( Q_j' \).

We are now ready to give the reduction. Given a single fiber star \( G \) with \( V(G) = \{u_l|0 \leq l \leq \Delta\} \) and \( E(G) = \{(v_0, u_l)|1 \leq l \leq \Delta\} \), and a set \( P \) of paths in \( G \), we create a 3-fiber star \( G' \) with \( V(G') = \{v_0\} \cup \{v_i|1 \leq i \leq \Delta\} \) and \( E(G') = \{(v_0, v_i)|1 \leq i \leq \Delta, 1 \leq i \leq 2w+1\} \). \( G' \) can be considered as the star obtained from \( \Delta \) copies \( G_w^l \) (1 \( \leq l \leq \Delta \)) of \( G \) by merging the \( \Delta \) centers of the copies into one center. For \( 1 \leq l \leq \Delta \), let \( P^l \) be the set of paths in \( G_w^l \) as \( P_w \) defined for \( G_w \) above. Let \( \hat{P} = \{\hat{p} | p \in P\} \), where \( \hat{p} \) is a path between \( v_i^l \) and \( v_i^m \) if \( p \) is a path between \( u_l \) and \( u_m \) \((l, m \neq 0)\) and \( \hat{p} \) is a path between \( v_0 \) and \( v_i^l \) if \( p \) is a path between \( u_0 \) and \( u_l \) \((l \neq 0)\). Let \( P' = \hat{P} \cup \bigcup_{l=1}^{\Delta} P^l \).

We show that there is a valid \( w \)-coloring for \( P' \) in \( G' \) if and only if \( P \) is \( w \)-colorable in \( G \). Assume \( P \) is \( w \)-colorable in \( G \). In \( G' \), we color each path \( \hat{p} \in \hat{P} \) using the color of the corresponding path \( p \in P \) in \( G \). By the definition of \( \bigcup_{l=1}^{\Delta} P^l \), it is easy to find a valid \( w \)-coloring for \( P' \setminus \hat{P} \). Combining the colorings for \( \hat{P} \) and \( P' \setminus \hat{P} \) gives a valid \( w \)-coloring of \( P' \) in \( G' \), since on each edge \( (v_0, v_i^l) \) in \( G' \), each color is used by at most one path in \( \hat{P} \) and by exactly two paths in \( P' \setminus \hat{P} \). On the other hand, if there is a valid \( w \)-coloring for \( P' \), then for each edge \( (v_0, v_i^l) \) of \( G' \), each of the \( w \) colors is used by exactly two paths in \( P' \setminus \hat{P} \). Thus, on each edge \( (v_0, v_i^l) \) of \( G' \), each color is used by at most one path in \( \hat{P} \). We can obtain a valid coloring for the set \( P \) of paths in \( G \) by coloring each path \( p \in P \) using the color for path \( \hat{p} \) in \( G' \). The above reduction clearly runs in polynomial time.

By a generalization of the reduction in the proof for Theorem 4.2.1, we have the following result.

**Theorem 4.2.2** The Min-PMC problem in \( k \)-fiber stars is NP-hard for every odd \( k \geq 3 \).

**Proof** Suppose \( k = 2k_0 + 1 \) \((k_0 \geq 1)\). Let \( G_1 \) be a \( k \)-fiber star with \( V(G_1) = \{u_i|0 \leq i \leq 3\} \) and \( E(G_1) = \{(v_0, u_i)|1 \leq i \leq 3\} \). Let \( Q_1 \) be the set of \( kw - 1 \) paths between \( v_2 \) and \( v_3 \),
CHAPTER 4. CALL CONTROL AND MAXIMUM PATH COLORING

Figure 4.5: Stars $G_1$ and $G_w$.

$Q'_1 = \{q_2, q_3\}$, where $q_2$ is the path between $v_1$ and $v_2$ and $q_3$ is the path between $v_1$ and $v_3$, and $P_1 = Q_1 \cup Q'_1$. It is easy to prove that $P_1$ is $w$-colorable and in any valid $w$-coloring of $P_1$, paths $q_2$ and $q_3$ must be assigned the same color. Let $G_{kw}$ be a $k$-fiber star with $V(G_{kw}) = \{v_i|0 \leq i \leq 2k_0w + 1\}$ and $E(G_{kw}) = \{(v_0, v_i)|1 \leq i \leq 2k_0w + 1\}$. $G_{kw}$ can be considered as the star obtained from $k_0w$ copies of $G_1$ by merging the $k_0w$ edges $(v_0, v_1)$ in the copies into one edge. Let $Q_j$ be the set of $kw - 1$ paths between $v_{2j}$ and $v_{2j+1}$ and $Q'_j = \{q_{2j}, q_{2j+1}\}$ for $1 \leq j \leq k_0w$, where $q_{2j}$ is the path between $v_1$ and $v_{2j}$ and $q_{2j+1}$ is the path between $v_1$ and $v_{2j+1}$. Let $P_{kw} = \bigcup_{j=1}^{k_0w}(Q_j \cup Q'_j)$ in $G_{kw}$. Again $P_{kw}$ is $w$-colorable and in any valid $w$-coloring of $P_{kw}$, the two paths in any $Q'_j$ are assigned the same color and each of the $w$ colors is used by exactly $k_0$ pairs from $\{Q'_j|j = 1, ..., k_0w\}$. To see this last point is true, suppose some color $\lambda$ is used by $k'_0 < k_0$ pairs. Each of the remaining $w - 1$ color can be used by at most $k_0$ pairs, since there are $2k_0 + 1$ fibers per edge. Thus, the total number of pairs colored by the $w$ colors is at most $k_0(w - 1) + k'_0 < k_0w$, a contradiction. Thus, the claim is true.

We are now ready to give the reduction. Given a single fiber star $G$ with $V(G) = \{u_l|0 \leq l \leq \Delta\}$ and $E(G) = \{(u_0, u_l)|1 \leq l \leq \Delta\}$, and a set $P$ of paths in $G$, we create a $k$-fiber star $G'$ with $V(G') = \{v_0\} \cup \{v_l|1 \leq l \leq \Delta, 1 \leq i \leq 2k_0w + 1\}$ and $E(G') = \{(v_0, v_i^l)|1 \leq l \leq \Delta, 1 \leq i \leq 2k_0w + 1\}$. $G'$ can be considered as the star obtained from $k_0w$ copies $G_{kw}$ by merging the $\Delta$ centers of the copies into one center. For $1 \leq l \leq \Delta$, let $P_l$ be the set of paths in $G_{kw}$ as $P_{kw}$ defined for $G_{kw}$ above. Let $\hat{P} = \{\hat{p}|p \in P\}$, where $\hat{p}$ is a path between $v_l^i$ and $v_l^m$ if $p$ is a path between $u_l$ and $u_m$ ($l, m \neq 0$) and $\hat{p}$ is a path between $v_0$ and $v_l^i$ if $p$ is a path between $u_0$ and $u_l$ ($l \neq 0$). Let $P' = \hat{P} \cup (\bigcup_{l=1}^{\Delta} P_l)$. It
is easy to prove that $P'$ is $w$-colorable in $G'$ if and only if $P$ is $w$-colorable in $G$. \qed

**Efficient algorithm for Min-PMC problem in stars**

Efficient algorithms for the Min-PMC problem in $k$-fiber ($k$ even) stars have been known [87, 88]. We have just shown that the Min-PMC problem is NP-hard for $k$-fiber ($k$ odd) stars. A natural question is, does the evenness of the number of fibers play an important role in the polynomial solvability of the Min-PMC problem in stars? In this subsection, we give an efficient algorithm for the Min-PMC problem in the non-uniform multifiber star $G$ with an even number $\mu(e)$ of fibers in every edge $e$. This suggests that the evenness is a key in the polynomial solvability for the problem. A path in a star is called a *short path* (resp. *long path*) if the path is on one edge (resp. two edges). Let $P$ be the given set of paths in $G$ and $w_{lb} = \max_{e \in E(G)} \lceil \frac{L(e)}{\mu(e)} \rceil$. Then $w_{lb}$ is an obvious lower bound on the number of colors required for coloring $P$. Algorithm ALG4.6 shown below uses exactly $w_{lb}$ colors (thus is optimal), if $\mu(e)$ is even for every $e \in E(G)$.

1. For any edge $e$ with odd $L(e)$, add one short dummy path on $e$. Let $P'$ be the union of $P$ and the set of dummy paths, and $P_L \subseteq P$ be the set of long paths in $P$. The number of paths in $P' \setminus P_L$ (short paths) is even, since $\sum_{e \in E(G)} L_{P'}(e)$ is even, each path of $P_L$ is on two edges and $\sum_{e \in E(G)} L_{P_L}(e)$ is even.

   Create a multigraph $G'$ with $V(G') = V(G)$ and there is an edge between $v_i$ and $v_j$ in $G'$ if there is a path between $v_i$ and $v_j$ in star $G$. Notice that every node in multigraph $G'$ has even degree and there is an Euler circuit in every connected component of $G'$.

2. Find an Euler circuit for every connected component of multigraph $G'$ and orient each Euler circuit. Then each path in star $G$ is assigned a direction by the oriented Euler circuits. Discard the dummy paths. For any edge $e$ in star $G$, the load of the paths with the same direction on $e$ is at most $\lfloor \frac{L(e)}{2} \rfloor$ for each direction.

3. For each leaf node in star $G$, partition the outgoing paths into sets, each of which has $w_{lb}$ paths (except the last set, which may have less than $w_{lb}$ paths). Similarly, partition the incoming paths of each leaf node into sets, each of which has $w_{lb}$ paths. For the center $v_0$, partition the incoming (resp. outgoing) short paths into incoming (resp. outgoing) sets each of which has $w_{lb}$ paths. Let $V_{in}$ (resp. $V_{out}$) be the collection
of sets of incoming (resp. outgoing) paths. Then each path $p \in P$ is in exactly one set of $V_{in}$ and exactly one set of $V_{out}$.

4. Create a bipartite (multi-)graph with bipartitions $V_{in}$ and $V_{out}$. For each path $p \in P$ in a set of $V_{in}$ and a set of $V_{out}$, create an edge $e_p$ between the corresponding nodes in $V_{in}$ and $V_{out}$.

5. The bipartite graph has node degree $w_{lb}$, and its edges can be colored optimally by $w_{lb}$ colors [43]. Color each path $p$ using the color of its corresponding edge $e_p$.

**Theorem 4.2.3** Algorithm ALG4.6 solves the Min-PMC problem in non-uniform stars with even number $\mu(e)$ of fibers in every edge $e$ using $w_{lb} = \max_{e \in E(G)} \lceil \frac{L(e)}{\mu(e)} \rceil$ colors in polynomial time.

**Proof** For any node $u$ of the bipartite graph, the edges incident to $u$ are assigned different colors. Thus, paths in any set of $V_{in}$ (or $V_{out}$) are assigned different colors. The color repetition at any edge $e$ incident to a leaf node $v$ of the star is just the total number of incoming and outgoing sets of $v$. When $L(e)$ is even, the color repetition at edge $e$ is at most

$$2 \left\lceil \frac{L(e)/2}{w_{lb}} \right\rceil \leq 2 \left\lceil \frac{L(e)/2}{\mu(e)} \right\rceil \leq 2 \left\lceil \frac{L(e)/2}{L(e)/\mu(e)} \right\rceil = 2 \left\lceil \frac{\mu(e)}{2} \right\rceil = \mu(e).$$

The last equality holds since $\mu(e)$ is even. Similarly, when $L(e)$ is odd,

$$\left\lfloor \frac{L(e)}{\mu(e)} \right\rfloor = \left\lfloor \frac{(L(e) + 1) / \mu(e)}{2} \right\rfloor$$

for even $\mu(e)$. The color repetition at edge $e$ is at most

$$2 \left\lceil \frac{\lceil L(e)/2 \rceil}{w_{lb}} \right\rceil \leq 2 \left\lceil \frac{\lceil L(e)/2 \rceil}{L(e)/\mu(e)} \right\rceil = \mu(e).$$

Therefore, the color repetition is always bounded from above by $\mu(e)$, the number of available fibers on edge $e$. Thus, the $w_{lb}$-coloring is valid. Algorithm ALG4.6 runs in polynomial time, since the construction of the multigraph and the bipartite graph can be done in polynomial time, and the Euler circuit and the bipartite graph edge-coloring can be found in polynomial time.

\[\square\]
The \( f \)-coloring of multigraphs is an extension of the edge-coloring, and can be defined as follows. Given a multigraph \( G \), a node capacity function \( f : V(G) \rightarrow \mathbb{N} \), color the edges in \( E(G) \) such that each color is used by at most \( f(v) \) edges incident to node \( v \). The goal is to minimize the number of colors used (this number is usually called the \( f \)-chromatic index of \( G \)). The edge-coloring problem is a special case of the \( f \)-coloring problem in which \( f(v) = 1 \) for all \( v \in V(G) \). The \( f \)-coloring problem is NP-hard and an asymptotic \((9/8)\)-approximation algorithm for the problem is known [94J. It is easy to show that the \( f \)-coloring problem in multigraphs is equivalent to the Min-PMC problem in stars (following the reduction in [48]). Thus, we can have an efficient algorithm for the \( f \)-coloring problem if \( f(v) \) is even for every node \( v \).

**Efficient algorithm for Min-PMC problem in \( k \)-fiber spiders**

We give an efficient algorithm ALG4.7 for the Min-PMC problem in \( k \)-fiber (\( k \) even) spiders. Let \( G \) be a \( k \)-fiber spider with the center node \( v_0 \). Given a set \( P \) of paths in \( G \), \( \lceil \frac{L}{k} \rceil \) is a lower bound on the number of colors for coloring \( P \). Algorithm ALG4.7 works as follows.

1. For every edge \( e \) of \( G \), if \( L(e) < \lfloor \frac{L}{k} \rfloor k \) then add unit-length dummy paths on \( e \) until \( L(e) = \lfloor \frac{L}{k} \rfloor k \). Let \( Q \) be the set of dummy paths on all edges of \( G \).

2. Let \( P_0 \subseteq (P \cup Q) \) be the set of paths on the center node \( v_0 \) of \( G \). Color \( P_0 \) using Algorithm ALG4.6 for the Min-PMC problem in stars.

3. For every leg in \( G \), color the paths of \( P \setminus P_0 \) in the leg by the algorithm for a chain regarding the segments of paths of \( P_0 \) in the leg as pre-colored paths.

**Theorem 4.2.4** Algorithm ALG4.7 solves the Min-PMC problem in \( k \)-fiber (\( k \) even) spiders using \( \lfloor \frac{L}{k} \rfloor \) colors in polynomial time.

**Proof** By Theorem 4.2.3 the number of colors used in Step 2 is \( \lfloor \frac{L}{k} \rfloor \), since the load of \( P_0 \) is \( \lfloor \frac{L}{k} \rfloor k \) (\( k \) even) and the number of colors needed is simply the load divided by \( k \). Furthermore, on each edge incident to the central node \( v_0 \), each color is used by exactly \( k \) paths in \( P_0 \). In Step 3, the paths on each leg can be colored by \( \lfloor \frac{L}{k} \rfloor \) colors. This is true since the chain algorithm of [96] works for the uniform case in which a subset of the paths is already colored and all the pre-colored paths are on the same endpoint of the chain. \( \square \)
Algorithm ALG4.7 can be used to derive an approximation algorithm for the Min-PMC problem in $k$-fiber ($k$ odd) spiders by the delete-one-fiber approach used in [97] for stars. For a $k$-fiber spider with odd $k \geq 3$, we consider the network as a $(k-1)$-fiber network and apply Algorithm ALG4.7 to solve the Min-PMC problem by $\left\lceil \frac{k}{k-1} \right\rceil$ colors. Since $\left\lceil \frac{k}{k-1} \right\rceil$ is a lower bound on the number of colors required for any optimal solution, this approach gives a $(1 + \frac{1}{k-1})$-approximation algorithm.

### 4.2.2 Max-PMC Problems in Stars and Spiders

In this section, we study the Max-PMC problem in multifiber stars and spiders. For a multifiber network with $\mu(e)$ fibers on edge $e$ and $w$ colors, any optimal solution for the Max-PMC problem has load at most $\mu(e) \times w$ on edge $e$. Consider the call control problem in the same network with edge capacity $c(e) = \mu(e) \times w$ on edge $e$. The optimal solution for the call control problem is an obvious upper bound on the optimal solution for the Max-PMC problem. The results in this section will use this observation. Our optimal algorithms for the Max-PMC problem achieve this upper bound.

#### Max-PMC problem in stars

The proofs of Theorems 4.2.1 and 4.2.2 imply that the Max-PMC problem is also NP-hard in $k$-fiber ($k$ odd) stars. We show that the Max-PMC problem can be solved optimally in stars with even number of fibers. Let $G$ be a star with an even number $\mu(e)$ of fibers in every edge $e$ and $P$ be a set of paths in $G$. Recall that in any optimal solution for the Max-PMC problem, the load of an edge $e$ is at most $\mu(e) \times w$. Thus, we can reduce the Max-PMC problem to the call control problem as follows: Assign an edge $e$ a capacity of $c(e) = \mu(e) \times w$ and solve the call control problem with $P$ as the input set of paths and $c$ as the capacity function. This call control problem can be solved optimally since it can be formulated as a $b$-matching problem [63]. Let $P' \subseteq P$ be the optimal solution for this call control problem. Then $|P'|$ is an upper bound on the cardinality of the optimal solution for the Max-PMC problem in $G$. The chosen set $P'$ of paths can be colored by $\max_{e \in E(G)} \left\lceil \frac{\mu(e) \times w}{\mu(e)} \right\rceil = w$ colors using Algorithm ALG4.6 in Section 4.2.1, since $\mu(e)$ is even for every $e \in E(G)$. Thus, $P'$ is an optimal solution for the original Max-PMC problem.

The same argument holds for the weighted Max-PMC problem in stars with non-uniform even fibers: one simply selects a maximum weight subset of paths with load at most $\mu(e) \times w$
on edge $e$ (this can also be done in polynomial time by reducing to the weighted $b$-matching problem), and then color the chosen set of paths using $w$ colors.

**Max-PMC in $k$-fiber ($k$ even) spiders**

We show that the Max-PMC problem can be solved optimally in $k$-fiber ($k$ even) spiders. Let $G$ be a $k$-fiber ($k$ even) spider and $P$ be a given set of paths in $G$ for the Max-PMC problem. We first select a maximum cardinality subset $P' \subseteq P$ of paths with load at most $kw$. This can be done in polynomial time since it is a special case of the call control problem in spiders (in which $c(e) = kw$ for every edge $e$), and can be efficiently solved using Algorithm ALG4.4 in Section 4.1.4. The selected set $P'$ of paths has load at most $kw$, and can be colored by $\frac{kw}{k} = w$ colors, using Algorithm ALG4.7 for $k$-fiber ($k$ even) spiders (see Section 4.2.1). $|P'|$ is an upper bound on the cardinality of the optimal solution for the Max-PMC problem. Thus, $P'$ is an optimal solution for the Max-PMC problem.

**Max-PMC problem in non-uniform spiders**

The Max-PMC problem is NP-hard in spiders with non-uniform fibers ($\mu(e)$ can be different for different edges and can be even or odd), since the Max-PMC problem is NP-hard in $k$-fiber ($k$ odd) stars (which are non-uniform spiders by definition). We solve the problem using the standard approach of calling the call control algorithm as a subroutine. Note that the call control problem is equivalent to the Max-PMC problem with $w = 1$. Suppose we have an approximation algorithm for the call control problem. Let $G$ be a spider with $\mu(e)$ fibers in edge $e \in E(G)$ and $P$ be a set of paths in $G$. Consider the call control problem with capacity function $c(e) = \mu(e)$ for all $e \in E(G)$. Then we call the algorithm for this call control problem, select a maximum cardinality subset of paths $P' \subseteq P$, and remove the paths $P'$ from $P$ (i.e., $P \leftarrow P \setminus P'$). The procedure is repeated $w$ times. Each selected subset is colored by a distinct color. The union of the $w$ chosen sets is taken as the solution for the Max-PMC problem. It is known that if the call control algorithm has approximation ratio $\rho$, this iterative greedy algorithm for the Max-PMC problem has approximation ratio $\frac{1}{1 - e^{-1/\rho}}$ [54]. The call control problem in spiders is polynomial time solvable (as shown in Section 4.1.4). Thus, $\rho = 1$, and the iterative greedy algorithm for the Max-PMC problem in spiders has an approximation ratio of $\frac{1}{1 - e^{-1}}$ (about 1.58). Summarizing the above, we have the following theorem.
Theorem 4.2.5 There is a 1.58-approximation algorithm for the Max-PMC problem in non-uniform spiders.

4.3 The Weighted Max-RPC Problem on Rings

In this section, we study the weighted maximum routing and path coloring problem in rings. Recall that for the weighted Max-RPC problem, we are given a set $S$ of routing requests, a weight function $w : S \rightarrow \mathbb{Z}^+$, and a set of $k$ colors. Each request can be routed either clockwise or counterclockwise. A feasible solution is a subset $S' \subseteq S$ such that each request $(s, t) \in S'$ is routed (through one of the two possible routes) and assigned one of the $k$ colors, with no two requests using the same color if they are routed through the same edge. The goal is to find a feasible solution which has the maximum total weight. For a set of routing requests $S$, we define $w(S) = \sum_{s \in S} w(s)$ as the total weight of all the requests in $S$. We define $w(p) = w(s)$ if $p$ is the routing path for $s$. For a set $P$ of paths, we define $w(P) = \sum_{p \in P} w(p)$ as the total weight of all the paths in $P$. The optimal solution is denoted by $OPT$ (which is used for both the set of requests and the total weight of the requests).

The following proposition will be used several times.

Proposition 4.3.1 [38] The weighted Max-RPC problem can be solved optimally in polynomial time for chain networks.

Our algorithm first discards one edge $e$ from the given ring, and solves the weighted Max-RPC problem on the obtained chain, using the optimal algorithm of [38] for chains. The result is refined by considering every edge $e$ on the ring as a candidate for deletion. Note that this second step is not necessary in the unweighted case [98], but is critical in the weighted case. Our algorithm uses a second approach, namely the maximum weight matching method, to see if a better result is possible. A (weighted) compatible graph $G_c$ is constructed as follows. Each routing request on the ring corresponds to a node in $G_c$ with the same weight. Two nodes are adjacent in $G_c$ if and only if the corresponding requests are parallel. (Two requests are parallel if and only if their end-points do not interleave on the ring, otherwise are crossing, see Figure 4.6. A pair of parallel requests may be routed without overlapping, while a pair of crossing requests cannot.) In addition, for each node $v$ just created, we add a duplicated node $v^d$ with weight 0 and connect $v$ and $v^d$ by a new
edge. (The introducing of the dummy node $v^d$ is necessary, since in an optimal solution, some request may be assigned a color that is not used by any other request.) To use the weighted matching algorithm, we define the weight of an edge in $G_c$ to be the sum of the weights of its two end-points. A maximum weighted matching of cardinality at most $k$ in $G_c$ is found and the set of connection requests is routed and colored according to this matching, where $k$ is the given number of colors. Our algorithm outputs the maximum of the two approaches. The detailed algorithm is shown in Figure 4.7.

Let OPT be an optimal solution (the selected subset of routing requests and the resulting paths). Given an edge $e$ on the ring, the paths in OPT can be divided in two subsets (with respect to $e$): $OPT^2_e$ which contains all the paths on edge $e$ (solid lines in Figure 4.8), and $OPT^1_e$ which contains all the paths not on edge $e$ (dashed lines in Figure 4.8). Then $OPT = OPT^1_e + OPT^2_e$, and $OPT^1_e \leq SOL_1$ (since $SOL_1$ is the maximum of the optimal solutions for the chain obtained by avoiding an edge $e$, among all edges $e$ on the ring). In the following discussion, we assume there exists an optimal solution $OPT$ (with routes chosen and paths colored), and we will compare the weight of our solution to the weight of the optimal. Note that the assumption of having an optimal solution $OPT$ is for the convenience of the proof. In the algorithm, we do not need the explicit knowledge of $OPT$.

For any two distinct edges $e_1$ and $e_2$ on the ring, let $P_{e_1} \subseteq OPT$ be the set of paths on $e_1$ (but not on $e_2$), $P_{e_2} \subseteq OPT$ be the set of paths on $e_2$ (but not on $e_1$), and $P_{e_1e_2} \subseteq OPT$.
CHAPTER 4. CALL CONTROL AND MAXIMUM PATH COLORING

ALG 4.8 Output $\max\{SOL_1, SOL_2\}$

SUB$_e$ Let $e$ be an edge of the ring. Route requests so that the resulting set $P$ of paths does not use $e$. Color a maximum weight subset of paths in $P$ using an optimal algorithm for chains (Proposition 4.3.1). Denote the solution by SOL$_e$.

ALG$_c$ Call SUB$_e$ for every edge $e$ of the ring. Output the maximum weight SOL$_1$.

(ALG$_c$ = max$_{e \in E}$ SOL$_e$).

ALG$_m$ Construct a compatible graph $G_e$. Find a maximum weight matching of cardinality at most $k$ in $G_e$. Route each pair of matched requests in a compatible manner and assign a distinct color to the pair. The solution is denoted by SOL$_2$.

Figure 4.7: The algorithm for the weighted Max-RPC

Figure 4.8: Different sets of paths in OPT.
be the set of paths on both $e_1$ and on $e_2$ (see Figure 4.9). $P_{e_1}$, $P_{e_2}$ and $P_{e_1,e_2}$ are pairwise disjoint sets. We have the following lemma:

**Lemma 4.3.2** $SOL_2 \geq w(P_{e_1}) + w(P_{e_2}) + w(P_{e_1,e_2})$.

**Proof** Consider the paths in $P_0 = P_{e_1} \cup P_{e_2} \cup P_{e_1,e_2}$. At most two paths in $P_0$ are assigned the same color, since paths in $P_0$ pass through at least one of the two edges $e_1$ and $e_2$, $P_0$ is a subset of $OPT$, and paths in $OPT$ have valid colorings. Paths in $P_{e_1,e_2}$ are assigned distinct colors, and do not share colors with paths in either $P_{e_1}$ or $P_{e_2}$. Paths in $P_{e_1}$ may share colors with paths in $P_{e_2}$. Let $P'_{e_1} \subseteq P_{e_1}$ (resp. $P'_{e_2} \subseteq P_{e_2}$) be the subset of paths each of which shares a color with some path in $P_{e_2}$ (resp. $P_{e_1}$). Then $|P'_{e_1}| = |P'_{e_2}|$, and $|P_{e_1}| - |P'_{e_1}| + |P_{e_1,e_2}| + |P_{e_2}| \leq k$. We can construct a matching $M_P$ in $G_c$ as follows. Let $v_p$ be the node in $G_c$ corresponding to the request assigned path $p$. For each path $p \in P'_{e_1}$, let $p' \in P'_{e_2}$ be the path which has the same color as $p$, and we include into $M_P$ the edge between $v_p$ and $v_{p'}$ in $G_c$. For each of the remaining paths $p \in P_{e_1,e_2} \cup (P_{e_1} \setminus P'_{e_1}) \cup (P_{e_2} \setminus P'_{e_2})$, we include into $M_P$ the edge between $v_p$ and its duplicated node $v^d_p$ in $G_c$. It is not hard to see that we have selected a total of $|P_{e_1}| - |P'_{e_1}| + |P_{e_1,e_2}| + |P_{e_2}| \leq k$ edges from $G_c$, each selected pair can be colored using one color, and requests corresponding to paths in $P_0$ are all selected. Thus in a matching of cardinality at most $k$, we can have at least all the requests corresponding to paths in $P_0$, and the selected requests can be colored by $k$ colors. The matching $M_P$ has weight $w(P_{e_1}) + w(P_{e_2}) + w(P_{e_1,e_2})$. The lemma is true since $SOL_2$ is
a maximum weight cardinality-\(k\) matching, and thus has weight at least equal to the weight of the matching \(M_P\) just constructed. \(\square\)

From Lemma 4.3.2, we have the following observations. If \(w(P_{e_1e_2}) = 0\) (i.e., no path is on both \(e_1\) and \(e_2\)), then \(OPT_{e_1}^2 = w(P_{e_1})\), \(OPT_{e_2}^2 = w(P_{e_2})\), and \(\min\{OPT_{e_1}^2, OPT_{e_2}^2\} \leq \frac{SOL}{2}\). Assume \(OPT_{e_1}^2 \leq OPT_{e_2}^2\) (the other case is symmetric),

\[
OPT = OPT_{e_1}^1 + OPT_{e_1}^2 \\
\leq SOL + 0.5SOL_2 \\
\leq 1.5SOL.
\]

In other words, \(SOL \geq OPT/1.5\). This gives a good bound on \(SOL\). For an optimal solution \(OPT\), there may not exist two edges \(e_1\) and \(e_2\) such that \(w(P_{e_1e_2}) = 0\) holds. However, we can show that for any optimal solution \(OPT\), there exists two distinct edges \(e_1\) and \(e_2\), and another optimal solution \(OPT_1\) such that \(OPT_1 = OPT\) and no path in \(OPT_1\) passes through both \(e_1\) and \(e_2\). Then, we use Lemma 4.3.2 to show our algorithm has an approximation ratio of 1.5. In what follows, we will try to find an \(OPT_1\) and two edges \(e_1\) and \(e_2\) that suit for this purpose.

To describe the procedure, we need some more definitions. Each path \(p\) on the ring corresponds to an arc from some node \(x\) to \(y\) along the ring in the clockwise direction. We call \(x\) the left-node and \(y\) the right-node of \(p\), and denote them by \(p^L\) and \(p^R\), respectively. Given an edge \(e\) on the ring, a path \(p\) in a set of paths \(P\) on \(e\) has the left-most left-node (with respect to \(e\)) if \(p^L\) is closest to \(e\) along the counterclockwise direction. Similarly, a path \(p\) in a set of paths \(P\) has the right-most right-node (with respect to \(e\)) if \(p^R\) is closest to \(e\) along the clockwise direction. A node \(x\) is on the left of node \(y\) if \(y\) is on the arc from \(x\) towards \(e\) in the clockwise direction.

Given an optimal solution \(OPT\), the identifying procedure works as follows:

1. Pick up an arbitrary edge \(e_1\) of the ring, and repeat Steps 2-4 until return.
2. If there is no path on \(e_1\), return.
3. If there is only one path \(p\) on \(e_1\), then identify any edge not on \(p\) as \(e_2\), and return.
4. Otherwise, there are at least two paths on \(e_1\). Let \(p_1\) be the path on \(e_1\) with the left-most left-node, and \(p_2\) be the path on \(e_1\) with the right-most right-node (break tie arbitrarily).
Figure 4.10: The procedure for identifying two edges on the ring.

(a) If $p_2$ is on the left of $p_1$ (see Figure 4.10(a)), or $p_1$ is the same path as $p_2$ (Figure 4.10(b)), then identify any edge on the clockwise segment from $p_2$ to $p_1$ as $e_2$, and return.

(b) Otherwise, $p_1$ and $p_2$ cover the whole ring (see Figures 4.10(c) and 4.10(d)). Re-route $p_1$ and $p_2$ (using the only other possible route), then switch the colors of them. Go back to Step 2.

The identifying procedure will clearly terminate, since initially there are a finite number paths on $e_1$, and each time Step 4(b) is executed, the number of paths on $e_1$ is reduced by two. It is easy to see that when the procedure terminates, we have re-routed some of the paths in $OPT$ (but did not change the set of accepted requests) and identified an edge $e_2$ such that no path passes through both $e_1$ and $e_2$. The paths $p_1$ and $p_2$ found in Step 4(b) are called a mutual-support pair. Note that the requests corresponding to $p_1$ and $p_2$ are parallel. However, they were routed in an incompatible manner in $OPT$. (It may seem
strange that \( OPT \) routes requests in such a way. However, we do not make any assumption on how \( OPT \) works as long as the result is optimal.) For any such pair, we have re-routed them avoiding edge \( e_1 \), exchanged their colors, and kept the paths which share colors with them unchanged, without changing the color of any other path. The resulting paths still have valid colorings. The weight of \( OPT \) is not changed, but some of the weight of \( OPT^2_{e_1} \) has been shifted to \( OPT^1_{e_1} \). Let \( OPT1 \) be the set of paths obtained after the procedure. Then \( OPT = OPT1 \). We are now comparing \( OPT \) with \( SOL \) obtained using \( ALG4.8 \).

**Theorem 4.3.3** The algorithm \( ALG4.8 \) achieves an approximation ratio of 1.5.

**Proof** Consider any optimal solution and apply the identifying procedure. If the procedure returns in Step 2, then no path in the resulting \( OPT1 \) passes through edge \( e_1 \), and \( OPT1 = OPT^1_{e_1} \leq SOL_1 \). Thus \( SOL \) is optimal. If the procedure returns in either Step 3 or Step 4(a), then we have identified two edges \( e_1 \) and \( e_2 \) such that no path in \( OPT1 \) passes through both \( e_1 \) and \( e_2 \). Using Lemma 4.3.2, we can show that \( OPT1 \leq 1.5SOL \). Summarizing the above, \( OPT \leq 1.5SOL \) in all cases. Thus, our algorithm achieves an approximation ratio of 1.5.

The algorithm clearly runs in polynomial time, since the Max-PC algorithm for chains and the maximum weight cardinality \( k \) matching algorithm both run in polynomial time, and we call the Max-PC algorithm for chains only a polynomial number of times. 

The approximation ratio of 1.5 achieved by our algorithm for the weighted Max-RPC is worse than the 4/3 ratio achieved for the (unweighted) Max-RPC problem in [33]. The algorithm of [33] used a combination of the cut-one-link method and an advanced version of the iterative greedy method. As mentioned earlier, the iterative greedy method may be more efficient than the maximum matching method. It is not clear whether the approximation ratio of our algorithm can be improved if we use the iterative greedy method instead of the maximum matching method. It seems that the analysis of [33] cannot be extended to the weighted case.

### 4.4 Summary

We have shown that the call control problem is NP-hard and MAX SNP-hard even in depth-2 trees with edge capacities one or two. We give polynomial time optimal algorithms.
for the call control problem in double-stars which are special depth-2 trees. These results suggest that depth-2 trees are a boundary topology for which the call control problem is in P or NP-hard, depending on the node degrees of the trees. We give polynomial time optimal algorithm for the call control problem in spiders. We also give 2-approximation and 3-approximation algorithms for the weighted call control problem in depth-2 and depth-3 trees, respectively. We show that the weighted call control problem is optimally solvable in arbitrary trees if all the paths contain a same node of the tree. We show that the weighted call control problem in any graphs can be solved optimally if all the paths have length at most 2.

We have shown that the Min-PMC and Max-PMC problems are NP-hard in $k$-fiber ($k$ odd) stars. This should be contrasted to the even $k$ case which can be solved optimally. We give optimal algorithms for the following problems: the Min-PMC and Max-PMC problems in non-uniform stars with even fibers, the Min-PMC and Max-PMC problems in $k$-fiber ($k$ even) spiders. We also obtain a 1.58-approximation algorithm for the Max-PMC problem in spiders with non-uniform fibers, using our call control algorithm for spiders as a subroutine.

We have given a 1.5-approximation algorithm for the weighted Max-RPC problem in rings. This improves the previous 1.58-approximation algorithm. Note that the Max-PC, Max-RPC, and the weighted Max-PC problems in rings can all be approximated with ratio better than 1.5. It would be interesting to design approximation algorithm with ratio less than 1.5 for the weighted Max-RPC problem in rings.
Chapter 5

Branch/Carving Decomposition Based Algorithms

In previous chapters, we have given efficient algorithms for the path coloring problem and the call control problem in various networks. Our algorithms only work on the topologies for which they are designed, and the techniques do not seem to be applicable in other topologies, or different problems in the same topology. Algorithms that work for a broader class of problems in a broader class of topologies are highly desirable. Most of our algorithms in previous chapters give only approximate solutions that are constant factors away from optimal solutions. Although approximate solutions can be computed in polynomial time, in some applications, an optimal solution is desired even at the cost of exponential computation time. Recently, there are increased interests in the exact algorithms for optimization problems. Many of the exact algorithms use dynamic programming method based on a tree/branch/carving decomposition of the graph. A graph of small treewidth/branchwidth/carvingwidth admits efficient dynamic programming algorithms for many NP-hard problems on the graph. A key step in these algorithms is to find a tree/branch/carving decomposition of small width for the graph. In this chapter, we propose efficient algorithms for computing optimal branch/carving decomposition of planar graphs. The contents of Sections 5.1 ~ 5.4 in this chapter are joint work with Qian-Ping Gu, Marjan Marzban, Hisao Tamaki, and Yumi Yoshitake, and appeared in the Proc. of the 10th SIAM Workshop on Algorithm Engineering and Experiments (ALENEX'08) [24].

In Section 5.1, we review some related work on optimal branch decompositions of planar
graphs. We give formal definitions of branchwidth and branch decomposition in Section 5.2. All known algorithms for the planar branch decomposition use Seymour and Thomas algorithm (called ST Procedure for short in what follows) which, given an integer $\beta$, decides whether $G$ has the branchwidth at least $\beta$ or not in $O(n^2)$ time. In Section 5.3, we propose efficient implementations of ST Procedure. We first review ST Procedure and give some observations which provide the base of our efficient implementations, then describe our implementations, and finally present the computational results. The computational results of our implementations show that the branchwidth of a planar graph can be computed in a practical time and memory space for some instances of size about one hundred thousand edges. Previous studies report that a straightforward implementation of the algorithm is memory consuming, which could be a bottleneck for solving instances with more than a few thousands edges. Our results suggest that with efficient implementations, the memory space required by the algorithm may not be a bottleneck in practice. Applying our implementations, an optimal branch decomposition of a planar graph of size up to several thousands edges can be computed in a reasonable time, using the edge-contraction method which runs in $O(n^3)$ time [66]. We describe the edge-contraction method and the computational results in Section 5.4.

Although the edge-contraction method can compute an optimal branch decomposition for planar graphs of practical size in a reasonable time, it is still time consuming for graphs with larger size. In Section 5.5, we propose divide-and-conquer based algorithms of using ST Procedure to compute optimal branch decompositions of planar graphs. Our algorithms have time complexity $O(n^3)$. Computational studies show that our algorithms are much faster than the edge-contraction algorithms and can compute an optimal branch decomposition of some planar graphs of size up to 50,000 edges in a practical time.

Branch-decomposition based algorithms have been explored as an approach for solving many NP-hard problems on graphs. Our results suggest that the approach could be practical. We will use the carving-decomposition based method to solve exactly the edge-disjoint paths problem in the next chapter.
5.1 Previous Work

The notions of branchwidth and branch decompositions are introduced by Robertson and Seymour [110] in relation to the more celebrated notions of treewidth and tree decompositions [108, 109]. A graph of small branchwidth (or treewidth) admits efficient dynamic programming algorithms for a vast class of problems on the graph [15, 28]. There are two major steps in a branch/tree-decomposition based algorithm for solving a problem: (1) computing a branch/tree decomposition with a small width and (2) applying a dynamic programming algorithm based on the decomposition to solve the problem. Step (2) usually runs in exponential time in the width of the branch/tree decomposition computed in Step (1). So it is extremely important to decide the branchwidth/treewidth and compute the optimal decompositions. It is NP-complete to decide whether the width of a given general graph is at least an integer $\beta$ if $\beta$ is part of the input, both for branchwidth [113] and treewidth [14]. When the branchwidth (treewidth) is bounded by a constant, both the branchwidth and the optimal branch decomposition (treewidth and optimal tree decomposition) can be computed in linear time [29, 31]. However, the huge constants behind the Big-Oh make the linear time algorithms only theoretically interesting.

One hurdle for applying branch/tree-decomposition based algorithms in practice is the difficulty of computing a good branch/tree decomposition because of the NP-hardness and huge hidden constants problems. Recently, the branch-decomposition based algorithms with practical importance for problems in planar graphs have been receiving increased attention [45, 57]. This is motivated by the fact that an optimal branch decomposition of a planar graph can be computed in polynomial time by Seymour and Thomas algorithm [113] and the algorithm is reported efficient in practice [70, 71]. Notice that it is open whether computing the treewidth of a planar graph is NP-hard or not. The result of the branchwidth implies a 1.5-approximation algorithm for the treewidth of planar graphs. Readers may refer to the recent papers by Bodlaender [30] and Hicks et al. [72] for extensive literature in the theory and application of branch/tree-decompositions.

Given a planar graph $G$ of $n$ vertices and an integer $\beta$, Seymour and Thomas give a decision algorithm which decides if $G$ has a branchwidth at least $\beta$ in $O(n^2)$ time [113]. Using ST Procedure as a subroutine, they also give an edge-contraction algorithm which constructs an optimal branch decomposition of $G$. The edge-contraction algorithm calls ST Procedure $O(n^2)$ times and runs in $O(n^4)$ time. Gu and Tamaki [66] give an improved
algorithm which calls ST Procedure $O(n)$ times and runs in $O(n^3)$ time to construct the branch decomposition. Hicks proposes a divide and conquer heuristic algorithm to reduce the number of calls for ST Procedure [71]. Computational studies show that the heuristic is effective in reducing the calls but has the time complexity of $O(n^4)$ [69, 71]. All known algorithms for computing the optimal branch decomposition of a planar graph rely on ST Procedure; thus, an efficient implementation of the procedure plays a key role in computing the branch decompositions. A straightforward implementation of ST Procedure requires $O(n^2)$ bytes of memory which is reported in [70] a bottleneck for solving large instances with more than 5,000 edges. Hicks proposes memory friendly implementations in the cost of performing re-calculations and increasing the running time of ST Procedure to $O(n^3)$ [70]. The time and memory space required by ST Procedure limit the size of planar graphs for which the optimal branch decompositions can be computed in practice. Hicks reports that the edge-contraction algorithm of [113] can solve some instances of about 2,000 edges and the divide and conquer method can solve some instances of about 7,000 edges in a practical time [69, 71].

5.2 Optimal Branch Decomposition of Planar Graphs

We give formal definitions of several terms that are mainly used in this chapter. Again, terms not defined here may be found in a standard textbook on graph theory. Let $G$ be a graph. A branch decomposition of $G$ is a tree $T_B$ such that the set of leaves of $T_B$ is $E(G)$ and each internal node of $T_B$ has node degree 3. For each edge $e$ of $T_B$, removing $e$ separates $T_B$ into two sub-trees. Let $E'$ and $E''$ be the sets of leaves of the subtrees. The width of $e$ is the number of vertices of $G$ incident to both an edge in $E'$ and an edge in $E''$. The width of $T_B$ is the maximum width of all edges of $T_B$. The branchwidth of $G$ is the minimum width of all branch-decompositions of $G$.

The algorithms of Seymour and Thomas [113] for branchwidth and branch decomposition are based on another type of decompositions called carving decompositions.

A carving decomposition of $G$ is a tree $T_C$ such that the set of leaves of $T_C$ is $V(G)$ and each internal node of $T_C$ has node degree 3. For each edge $e$ of $T_C$, removing $e$ separates $T_C$ into two sub-trees and the two sets of the leaves of the sub-trees are denoted by $V'$ and $V''$. The width of $e$ is the number of edges of $G$ with both an end vertex in $V'$ and an end vertex in $V''$. The width of $T_C$ is the maximum width of all edges of $T_C$. The carvingwidth
of $G$ is the minimum width of all carving decompositions of $G$. Notice that the carving decomposition is defined for more general graphs in [113]. The definition allows positive integer lengths on edges of the graphs. The width of $e$ in $T_C$ for the weighted graph is defined as the sum of lengths of edges with an end vertex in $V'$ and an end vertex in $V''$.

Let $G$ be a planar graph with a fixed embedding. Let $R(G)$ be the set of faces of $G$. The [*medial graph*][113] $M(G)$ of $G$ is a planar graph with an embedding such that $V(M(G)) = \{ u_e | e \in E(G) \}$, $R(M(G)) = \{ r_s | s \in R(G) \} \cup \{ r_v | v \in V(G) \}$, and there is an edge $\{ u_e, u_{e'} \}$ in $E(M(G))$ if the edges $e$ and $e'$ of $G$ are incident to a same vertex $v$ of $G$ and they are consecutive in the clockwise (or counter clockwise) order around $v$. $M(G)$ in general is a multigraph but has $O(|V(G)|)$ edges. Seymour and Thomas [113] show that the carvingwidth of $M(G)$ is exactly twice the branchwidth of $G$ and an optimal carving decomposition of $M(G)$ can be translated into an optimal branch decomposition of $G$ in linear time. To decide whether a planar graph $G$ has the branchwidth at least an integer $\beta$, ST Procedure actually decides whether $M(G)$ has the carvingwidth at least $2\beta$.

**Proposition 5.2.1** (Seymour and Thomas [113]) Given a planar graph $G$ of $n$ vertices and an integer $\beta$, $bw(G) = cw(M(G))/2$, ST Procedure decides if $bw(G) \geq \beta$ by computing if $cw(M(G)) \geq 2\beta$ in $O(n^2)$ time, and an optimal carving decomposition of $M(G)$ can be translated into an optimal branch decomposition of $G$ in $O(n)$ time.

A face $r \in R(G)$ and an edge $e \in E(G)$ are incident to each other if $e$ is a boundary of $r$ in the embedding. Notice that an edge $e$ is incident to exactly two faces. For a face $r \in R(G)$, a vertex $v$ is incident to $r$ if $v$ is an end vertex of an edge incident to $r$. For a face $r \in R(G)$, let $V(r)$ and $E(r)$ be the sets of vertices and edges incident to $r$, respectively. For a vertex $v \in V(G)$, let $E(v)$ be the set of edges incident to $v$.

The [*planar dual*][113] $G^*$ of $G$ is defined as that for each vertex $v \in V(G)$, there is a unique face $r_v^* \in R(G^*)$; for each face $r \in R(G)$, there is a unique vertex $v_r^* \in V(G^*)$; and for each edge $e \in E(G)$ incident to $r$ and $r'$, there is a unique edge $e^* = \{ v_r^*, v_{r'}^* \} \in E(G^*)$ which crosses $e$.

A walk in a graph $G$ is a sequence of edges $e_1, e_2, ..., e_k$ of $G$, where $e_i = \{ v_{i-1}, v_i \}$ for $1 \leq i \leq k$. A walk is closed if $v_0 = v_k$. The length of a walk is the number of edges in the walk. For two vertices $u$ and $v$ in a graph $G$, the distance $d(u, v)$ is the minimum length of all walks between $u$ and $v$. The walk with distance $d(u, v)$ is a shortest path between $u$ and $v$. 

---

5.3 Empirical Study on Branchwidth of Planar Graphs

In this section, we propose efficient implementations of ST Procedure. Our implementations can be classified into two groups. Group (1) does not perform re-calculations and runs in $O(n^2)$ time. The most memory efficient implementation in this group can compute the branchwidth of some instance of size up to one hundred thousand edges with 500Mbytes memory and in a couple of hours. Group (2) performs re-calculations and can compute the branchwidth of the instance of one hundred thousand edges with 200Mbytes of memory. The implementations in Group (2) may run in $O(n^3)$ time in the worst case. All of our implementations still use $O(n^2)$ bytes of memory. However, the constants behind the Big-Oh are much smaller than those in a straightforward implementation. In contrast, the results of this thesis and those of [70] show that straightforward implementations can only handle instances of size up to about 5,000 edges within 1Gbytes of memory. Our most time efficient implementation is faster than the straightforward one by a factor of $3 \sim 15$. Compared with the previous memory friendly implementations of [70], our most memory efficient implementations of Group (1) and Group (2) use at most $1/4$ memory and $1/8$ memory and run faster by a factor of $100 \sim 400$ and a factor of $100 \sim 200$, respectively. Notice that the CPU used in [70] has frequency 194MHz and the CPU used for testing our implementations has frequency 3.06GHz, so we need to keep in mind this difference of speed when we compare the running time.

The results of this section suggest that the memory size required by ST Procedure may not be a bottleneck for computing the branchwidth and optimal branch decomposition of a planar graph in practice. Our implementations also imply more efficient algorithms which call ST Procedure to find the optimal branch decompositions. We will discuss the decomposition algorithms in Sections 5.4 and 5.5.

5.3.1 Seymour and Thomas Procedure

We give a brief review of ST Procedure and readers may refer to [113] for more details of the decision procedure. ST Procedure is often called the rat-catching algorithm because it can be intuitively described by a rat catching game introduced in [113]. We first review the game and then give a formal description of ST Procedure.
Rat catching game

In this game, there are two players, a rat and a rat-catcher. The game is on a planar graph $G$ of a fixed embedding, with a face and an edge of $G$ interpreted as a room and a wall of a room, respectively. The rules for the game are as follows.

(R1) The rat-catcher selects a room.

(R2) The rat selects a corner of a room (a vertex of $G$).

(R3) The rat-catcher selects a room adjacent to the current room and moves to the wall between the two rooms (the edge of $G$ incident to the current face and the selected face). The rat-catcher generates a noise of a fixed level that may make walls noisy. The condition of making a wall noisy will be given later.

(R4) The rat moves to a different corner via walls or stays at the current corner. The rat can not use a noisy wall but can use as many quiet walls as possible in one move.

(R5) The rat-catcher moves to the room it selected and can not change its mind to move back to the previous room. The rat-catcher keeps making noise.

(R6) If the rat is in a corner, all walls incident to the corner are noisy, and the rat-catcher is in a room with this corner, then the rat-catcher catches the rat and wins the game. Otherwise goto (R3).

Now we give the condition on a wall becoming noisy. For the planar dual $G^*$ of $G$, let $v_*$ and $e^*$ be the vertex and edge of $G^*$ corresponding to the face $r$ and edge $e$ of $G$, respectively. Let $k$ be the noise level produced by the rat-catcher. When the rat-catcher is on edge $e$, edge $f$ is noisy if and only if there is a closed walk of length smaller than $k$ containing edges $e^*$ and $f^*$ in $G^*$. Similarly, when the rat-catcher is in face $r$, edge $f$ is noisy if and only if there is a closed walk of length smaller than $k$ containing vertex $v_*$ and edge $f^*$ in $G^*$. The rat-catcher wins the game if the rat is at a vertex $v$ with node degree smaller than $k$ and the rat-catcher is in a face incident to $v$. The rat wins the game if there is a scheme by which the rat can escape from the rat-catcher for ever. We use $RC(G, k)$ to denote the rat catching game on $G$ and $k$. Seymour and Thomas show that the rat wins the game $RC(G, k)$ if and only if $G$ has carvingwidth at least $k$ and give ST Procedure which, given $G$ and $k$, computes the outcome of the game $RC(G, k)$ [113].
ST Procedure

Now we present ST Procedure using the language of the game \( RC(G, k) \). Our presentation is different from the original one which is based on a notion called antipodality [113]. For a graph \( G \) with maximum node degree at least \( k \), the rat always wins the game \( RC(G, k) \) because the rat will never get caught if it stays at a vertex with node degree at least \( k \). So we assume that \( G \) has maximum node degree smaller than \( k \) in the following discussion. Given \( G \) and \( k \), ST Procedure computes an escaping scheme for the rat or decides no such scheme exists. The escaping scheme is represented by a collection of vertex subsets and subgraphs of \( G \) by which the rat can escape from the rat-catcher for ever. The collection contains a non-empty subset of vertices of \( G \) (a subset of corners) for every face and a non-empty subgraph of \( G \) (a subset of corners and quiet walls) for every edge.

Given \( G \) and \( k \), we define \( G_e \) to be the subgraph of \( G \) obtained by deleting noisy edges from \( G \) when the rat-catcher is on edge \( e \). More specifically, \( V(G_e) = V(G) \) and

\[
E(G_e) = \{ f \mid \text{every closed walk of } G^* \text{ containing } e^* \text{ and } f^* \text{ has length at least } k \}.
\]

Notice that every edge of \( G_e \) is quiet when the rat-catcher is on \( e \). For each face \( r \in R(G) \), we define

\[
S_r = \{(r, v) | v \in V(G)\} \quad \text{and} \quad S = \bigcup_{r \in R(G)} S_r.
\]

For each edge \( e \in E(G) \), we define

\[
T_e = \{(e, C) | C \text{ is a connected component of } G_e \}.
\]

Let \( T = \bigcup_{e \in E(G)} T_e \). Then the game \( R(G, k) \) can be described by a bipartite graph \( H(G, k) \), where the vertex set of \( H(G, k) \) is \( S \cup T \) and there is an edge between \((r, v) \in S \) and \((e, C) \in T \) if face \( r \) is incident to edge \( e \) and \( v \) is a vertex of \( C \). The vertices of \( H(G, k) \) can be interpreted as the states of the game: a \((r, v) \in S \) represents that the rat-catcher is in face \( r \) and the rat is at vertex \( v \), and a \((e, C) \in T \) expresses that the rat-catcher is on edge \( e \) and the rat is at a vertex of \( C \) and can use the edges of \( C \) to move. The edge between \((r, v) \in S \) and \((e, C) \in T \) indicates the possible state transitions of the game: when the rat is at \( v \) and the rat-catcher moves from face \( r \) to edge \( e \), the game state transits from \((r, v) \) to \((e, C) \); or when the rat is at some vertex of \( C \) and moves to \( v \), and the rat-catcher moves from edge \( e \) to face \( r \), the game state transits from \((e, C) \) to \((r, v) \).
A game state of \( R(G, k) \) is called a losing state if the rat will lose the game at the state. To compute an escaping scheme for the rat, ST Procedure deletes the losing states from \( H(G, k) \). For a face \( r \in R(G) \) and a vertex \( v \) incident to \( r \) \( (v \in V(r)) \), \( (r, v) \) is a losing state because the rat gets caught if the rat is at \( v \) and the rat-catcher is in \( r \). For an edge \( e \) incident to face \( r \), \( (e, C) \) is a losing state if for every vertex \( v \) of \( C \), \( (r, v) \) is a losing state. To see this, assume that the rat-catcher is on edge \( e \) and the rat is at some vertex of \( C \). The rat-catcher may move to \( r \) or \( r' \), the other face incident to \( e \), in the next step. In either of the moves, the rat can only move to a vertex \( v \) of \( C \). If the rat-catcher moves to \( r \) then the game transits to \( (r, v) \) at which the rat will get caught. If the rat-catcher moves to \( r' \) then the rat is at some vertex of \( C \). The rat-catcher can move back to \( e \) and then to \( r \), and the rat will get caught. Similarly, if \( (e, C) \) is a losing state then for every face \( r \) incident to \( e \) and every vertex \( v \) of \( C \), \( (r, v) \) is a losing state.

For every edge \( e \in E(G) \), ST Procedure initializes set \( X_e \) to include all states of \( T_e \). For every face \( r \in R(G) \), ST Procedure initializes set \( X_r \) to include all states of \( S_r \) and then deletes \( (r, v) \) from \( X_r \) for every \( v \in V(r) \). After this initial deletion step, for each face \( r \) and each edge \( e \) incident to \( r \), if there is a state \( (e, C) \) such that for every vertex \( v \) of \( C \) state \( (r, v) \) has been deleted, then the state \( (e, C) \) is deleted from \( X_e \). If this deletion is done then for the other face \( r' \) incident to \( e \), state \( (r', v) \) is deleted from \( X_{r'} \) for every vertex \( v \) of \( C \). This deletion may result in further deletions of losing states. The deletion process is repeated until no further deletion is possible. It is shown in [113] that graph \( G \) has carvingwidth at least \( k \) if and only if after the deletion process finishes, \( X_r \) and \( X_e \) are not empty for every \( r \in R(G) \) and every \( e \in E(G) \). The collection of non-empty \( X_r \) and \( X_e \) for every face \( r \) and every edge \( e \) is an escaping scheme for the rat. Below is a simplified version of the formal description of ST Procedure [113]. We remark that ST Procedure decides if the carvingwidth is at least \( k \) for more general planar graphs. It allows weighted input graphs with positive integer lengths on edges.

**ST Procedure**

**Input:** A non-null connected planar graph \( G \) with a fixed embedding, a planar dual \( G^* \) of \( G \), an integer \( k \geq 0 \).

**Output:** Decides if \( G \) has carvingwidth at least \( k \).

1. If the maximum node degree of \( G \) is at least \( k \) then output \( G \) has carvingwidth at least \( k \) and terminate.
2. For each face \( r \in R(G) \), let \( X_r = S_r \).

For each edge \( e \in E(G) \), compute \( G_e \) and let \( X_e = T_e \). For each \( (e, C) \in X_e \) and the faces \( r \) and \( r' \) incident to \( e \), let \( c(r, e, C) = |V(C)| \) and \( c(r', e, C) = |V(C)| \), where \( V(C) \) is the set of vertices of \( C \).

3. For each face \( r \) and each state \( (r, v) \in X_r \) with \( v \in V(r) \), put \( (r, v) \) to a stack \( L \) and delete \( (r, v) \) from \( X_r \).

4. If \( L \) is empty then goto the next step.

Otherwise, remove a state \( x \) from \( L \).

Assume that \( x = (r, v) \) is a state for a face \( (x \in S) \). For each edge \( e \) incident to \( r \), find the state \( (e, C) \in X_e \) such that \( C \) contains \( v \). Decrease \( c(r, e, C) \) by one. If \( c(r, e, C) \) becomes 0 and \( (e, C) \in X_e \) then put \( (e, C) \) to \( L \) and delete \( (e, C) \) from \( X_e \).

Assume that \( x = (e, C) \) is a state for an edge \( (x \in T) \). If there is a face \( r \) incident to \( e \) such that \( c(r, e, C) > 0 \) then for each vertex \( v \) of \( C \) and \( (r, v) \in X_r \) put \( (r, v) \) to \( L \) and delete \( (r, v) \) from \( X_r \).

Repeat this step.

5. If \( X_r \) is non-empty for every \( r \in R(G) \) and \( X_e \) is non-empty for every \( e \in E(G) \) then output \( G \) has carvingwidth at least \( k \); otherwise output \( G \) has carvingwidth smaller than \( k \).

Notice that we can stop ST Procedure and conclude that the rat loses the game when some \( X_r \) becomes empty. The reason is that if all states of \( X_r \) are deleted, all states of \( X_e \) for \( e \) incident to \( r \) will be deleted; then all states for face \( r' \) incident to \( e \) will be deleted; and finally all states for every face and edge will be deleted. Similarly, the rat loses the game if some \( X_e \) becomes empty.

To compute \( G_e \) for each \( e \), ST Procedure needs to find the quiet edges when the rat-catcher is on edge \( e \). An edge \( f \) is quiet and will be included in \( G_e \) if every closed walk in \( G^* \) that contains \( e^* \) and \( f^* \) has length at least \( k \). More specifically, let \( e^* = \{u^*, v^*\} \) and \( f^* = \{x^*, y^*\} \). Edge \( f \) is included in \( G_e \) if and only if \( d(u^*, x^*) + d(v^*, y^*) + 2 \geq k \) and \( d(u^*, y^*) + d(v^*, x^*) + 2 \geq k \). A solution for the all-pairs shortest path problem of \( G^* \) will suffice for the distances required in computing \( G_e \) for all \( e \in E(G) \).
**Theorem 5.3.1** *(Seymour and Thomas [113])*

Given a planar graph $G$ of $n$ vertices and integer $k \geq 0$, ST Procedure decides if $G$ has carvingwidth at least $k$ or not using graph $H(G, k)$ in $O(n^2)$ time and $O(n^2)$ bytes of memory.

To decide the branchwidth of $G$, the input to ST Procedure is the medial graph $M(G)$ and the branchwidth of $G$ is $k/2$ if the carvingwidth of $M(G)$ is $k$.

**Observations for efficient implementations**

We give some observations on the game $RC(G, k)$ that can be used for efficient implementations of ST Procedure. By the definition of the game $RC(G, k)$, a state $(r, v)$ is a losing state if $v \in V(r)$ for $G$ with maximum node degree smaller than $k$. ST Procedure makes use of this sufficient condition to delete the losing states at the initial step of the deletion process for each face $r$. We observe that if we can find and delete more losing states at the initial step for each face $r$, then ST Procedure may run faster and use less memory. We prove the following sufficient condition for finding more losing states.

**Lemma 5.3.2** For a face $r$ and a vertex $v$ in graph $G$ with maximum node degree smaller than $k$, $(r, v)$ is a losing state if there exist two faces $s$ and $t$ incident to $v$ such that there are

1. a closed walk $W_1$ in $G^*$ with length smaller than $k$ that consists of the shortest path from $v_r^*$ to $v_s^*$, the clockwise walk from $v_s^*$ to $v_t^*$ around $r_s^*$, and the shortest path from $v_t^*$ to $v_r^*$; and
2. a closed walk $W_2$ in $G^*$ with length smaller than $k$ that consists of the shortest path from $v_r^*$ to $v_s^*$, the counter-clockwise walk from $v_s^*$ to $v_t^*$ around $r_s^*$, and the shortest path from $v_t^*$ to $v_r^*$.

**Proof** Assume that $W_1$ and $W_2$ exist. Then for every edge $e$ incident to $v$ in $G$, $e^*$ is either in $W_1$ or $W_2$ and $e$ is noisy when the rat-catcher is in $r$. Let $e_1^*, ..., e_j^*$ be the edges in the shortest path from $v_r^*$ to $v_s^*$. Assume that the rat is at $v$ and the rat-catcher is in $r$. Since all edges incident to $v$ are noisy, the rat can not move away from $v$. Next, the rat-catcher can move to edge $e_1$. Since all edges incident to $v$ are noisy when the rat-catcher is on $e_1$, the rat has to stay at $v$. Similarly, the rat has to stay at $v$ when the rat-catcher is on edge $e_i, 1 \leq i \leq j$. So the rat-catcher can move to face $s$ using edges $e_1, ..., e_j$ and catch the rat at $v$. \qed
Once the shortest paths from \( v^* \) to all other vertices of \( G^* \) have been computed, it is easy to see the time for checking if \((r, v)\) is a losing state by the condition of Lemma 5.3.2 is proportional to the node degree of \( v \). Therefore, it takes \( O(n) \) time to check \((r, v)\) for a face \( r \) and all \( v \in V(G) \). For each face \( r \), let \( U(r) \) be the set of vertices that for every \( v \in U(r) \), \((r, v)\) is a losing state computed by the sufficient condition of Lemma 5.3.2. From Theorem 5.3.1, we have the following result.

**Theorem 5.3.3** Given a planar graph \( G \) of \( n \) vertices and \( k \geq 0 \), \( ST \) Procedure decides if \( G \) has carvingwidth at least \( k \) in \( O(n^2) \) time and \( O(n^2) \) bytes of memory when the losing states \((r, v)\), \( v \in U(r) \), are deleted at the initial step of the deletion process for each face \( r \).

For each face \( r \in R(G) \), we define \( G_r \) to be the subgraph of \( G \) obtained by deleting the noisy edges from \( G \) when the rat-catcher is in face \( r \). That is, \( V(G_r) = V(G) \) and

\[
E(G_r) = \{ f \mid \text{every closed walk of } G^* \text{ containing } v^*_r \\
\text{and } f^* \text{ has length at least } k \}.
\]

Notice that every edge of \( G_r \) is quiet when the rat-catcher is in \( r \). Recall that \( G_e \) is the quiet subgraph of \( G \) when the rat-catcher is on edge \( e \). Our next observation is that for every edge \( e \) incident to face \( r \), \( E(G_r) \subseteq E(G_e) \), because \( v^*_r \) is an end vertex of \( e^* \) and therefore the set of closed walks of \( G^* \) containing vertex \( v^*_r \) and edge \( f^* \) includes all closed walks of \( G^* \) containing edges \( e^* \) and \( f^* \). From this, a component of \( G_r \) is a subgraph of some component of \( G_e \). Hence, when the rat-catcher moves from face \( r \) to edge \( e \) and the rat is at any vertex of some component \( D \) of \( G_r \), the component of \( G_e \) on which the rat can move is the same one which contains \( D \) as a subgraph. Thus, when the rat-catcher is in face \( r \), the states of the game can be expressed by

\[
S'_r = \{(r, D) | D \text{ is a connected component of } G_r \}.
\]

Let \( S' = \bigcup_{r \in R(G)} S'_r \). The game \( RC(G, k) \) can be described by a bipartite graph \( H'(G, k) \), where the vertex set of \( H'(G, k) \) is \( S' \cup T \) and there is an edge between \((r, D) \in S' \) and \((e, C) \in T \) if face \( r \) is incident to edge \( e \) and \( D \) is a subgraph of \( C \). For a face \( r \) and a component \( D \) of \( G_r \), \((r, D)\) is a losing state if for every vertex \( v \) of \( D \), \((r, v)\) is a losing state. For an edge \( e \) incident to face \( r \), state \((e, C)\) is a losing state if for every component \( D \) of \( G_r \) that is a subgraph of \( C \), \((r, D)\) is a losing state. Similarly, if \((e, C)\) is a losing state then for every face \( r \) incident to \( e \) and every component \( D \) of \( G_r \) that is a subgraph of \( C \), \((r, D)\) is a losing state. Summarizing the above and from Theorem 5.3.1, the following result holds.
**Theorem 5.3.4** Given a planar graph $G$ of $n$ vertices and $k \geq 0$, ST Procedure decides if $G$ has carvingwidth at least $k$ using graph $H'(G, k)$ in $O(n^2)$ time and $O(n^2)$ bytes of memory.

When graph $H'(G, k)$ is used for the game $RC(G, k)$, $X_r$ is initialized as $S'_r$ for each face $r \in R(G)$ in ST Procedure. Compared with $S_r$, $S'_r$ may have less game states and thus require less memory.

During the deletion process of ST Procedure, losing states are deleted from sets $X_r$ and $X_e$. Our another observation is that the elements of $X_e$ for an edge $e$ incident to faces $r$ and $r'$ at a step of ST Procedure can be computed in $O(n)$ time from the elements of $X_r$ and $X_{r'}$ at that step. This gives an option for implementing ST Procedure that does not keep but dynamically computes $X_e$ from $X_r$ and $X_{r'}$ during the deletion process.

**Theorem 5.3.5** Given a planar graph $G$ of $n$ vertices and $k \geq 0$, ST Procedure can decide if $G$ has carvingwidth at least $k$ or not in $O(n^3)$ time and $O(n^2)$ bytes of memory if for each edge $e$, $X_e$ is not kept but dynamically computed during the deletion process.

**Proof** For an edge $e$ incident to faces $r$ and $r'$, the set $X_e$ is needed when an element of $X_r$ or $X_{r'}$ is deleted during the computation and when ST Procedure terminates. So, we can compute $X_e$ in $O(n)$ time once there is an element deleted from $X_r$ or $X_{r'}$. Since there are $O(n)$ elements in $X_r \cup X_{r'}$, $X_e$ is computed $O(n)$ times. The total time for computing $X_e$ for all $e \in E(G)$ is $O(n^3)$. From Theorem 5.3.1, the theorem holds.  

The re-calculation of edge data used in our implementations is different from the re-calculation in the previous study of [70], where face data are re-calculated for some faces and each re-calculation for a face $r$ involves a computation of $T_e$ for each $e$ incident to $r$.

Finally, it is easy to see that if all states of $S_r$ (or $S'_r$) for some face $r$ are losing states then for every face $r'$, all states of $S_{r'}$ ($S'_{r'}$) are losing states and the rat loses the game.

**Observation 5.3.6** If $X_r$ becomes empty for some face $r$ during the deletion process then graph $G$ has carvingwidth smaller than $k$.

By this observation we can terminate ST Procedure when some $X_r$ becomes empty. This may save the computation time when the rat loses the game.
CHAPTER 5. BRANCH/CARVING DECOMPOSITION BASED ALGORITHMS

5.3.2 Efficient Implementations

Let $G$ be a connected planar graph with a given embedding and $V(G) = \{v_1, \ldots, v_n\}$. We first describe a straightforward implementation (called Naive) of ST Procedure and then propose several improvements on the implementations of ST Procedure. Those improvements try to reduce both the memory space and running time of ST procedure.

Naive implementation

A straightforward implementation of ST Procedure would use graph $H(G, k)$ for deciding the outcome of the game $RC(G, k)$. We use the following data structure for graph $H(G, k)$ in Naive.

- For each face $r \in R(G)$, a Boolean array $B_r$ (of $n$ elements) is assigned such that $B_r[i]$ is used to indicate if $(r, v_i) \in X_r$ or not. A list of $|E(r)|$ elements is used to keep the edges incident to $r$.

- For each edge $e \in E(G)$, the two faces $r$ and $r'$ incident to $e$ are kept. All components of $G_e$ are kept in a list. Each component of $G_e$ is given an index and component $C_j$ is kept in the $j$th element of the list. The element of the list for $C_j$ contains the set of vertices of $C_j$, $c(r, e, C_j)$, $c(r', e, C_j)$, and a Boolean variable indicating if $(e, C_j)$ has been deleted from $X_e$ or not. An integer array $I_e$ (of $n$ elements) is used to indicate which component a vertex is in. If $v_i$ is a vertex of $C_j$ then $I_e[i]$ is set to $j$.

- In addition to the face and edge data, a stack $L$ is used and a distance matrix is kept for the all pairs shortest distances in the dual graph $G^*$ of $G$.

It is easy to check that the Naive implementation runs in $O(n^2)$ time. A simple calculation shows that Naive implementation requires about $40n^2$ bytes of memory when $G$ is a medial graph. Since there are many single vertex components in $G_e$ and the operating system may have a minimum memory allocation size of 16 bytes, the memory usage in practice is close to $50n^2$ bytes.

Common improvements

We first describe two common improvements which are used in all of our efficient implementations. When we say processing a face $r$ or an edge $e$, we mean deleting a losing state from $X_r$ or $X_e$. 
The first common improvement is that we define a processing order of the faces in our implementations. We put losing states \((r, v)\) to the stack for only one face at a time. When there are losing states of multiple faces to be included to the stack, we group the losing states according to the faces, and give an order on the groups to be put to the stack. Only the group at the top of the order is put to the stack at a time. A face which has been processed is given a higher priority to be put to the stack. The processing order on the faces is used to define a subset \(Q \subseteq R(G)\) and to restrict the rat-catcher moving within the faces of \(Q\). Given a subset \(Q\) of \(R(G)\), let \(S_Q = \cup_{r \in Q} S_r\), \(S'_Q = \cup_{r \in Q} S'_r\), and \(T_Q = \cup_{e \in E(r), r \in Q} T_e\).

We start with a small \(Q\) and perform the deletion process for the subgraph of \(H(C, k)\) induced by the vertices of \(S_Q \cup T_Q\) (or the subgraph of \(H'(C, k)\) induced by the vertices of \(S'_Q \cup T_Q\)) until no deletion is possible. Then we enlarge \(Q\) by including a new face and repeat the deletion process. \(Q\) is enlarged gradually until \(Q = R(G)\). By Observation 5.3.6, ST Procedure may stop at a small \(Q\) when the rat-catcher wins the game. Also, for a given subset \(Q\), the losing states are deleted from \(X_r\) and \(X_e\) \((r \in Q, e \in E(r))\), and after the deletion, the data for \(X_r\) and \(X_e\) can be compressed before \(Q\) is enlarged. This helps in reducing the time and memory of ST Procedure.

The second common improvement is that we use a parsimonious data structure for edge data. We observe that there are many single vertex components in edge data. This makes the list of components for each edge very big. We keep the same face data as those in Naive. For each edge \(e\), a component of \(G_e\) is called non-trivial if it has at least one edge otherwise called trivial. We only assign an index to a non-trivial component and keep a list of non-trivial components. We decide the integer type for \(I_e\) based on the number of non-trivial components in \(G_e\). The length of the integer type for \(I_e\) is just big enough to encode the indices of non-trivial components of \(G_e\). A trivial component \(C = \{v_i\}\) is not kept in the list and \(I_e[i]\) is used to indicate if \((e, C)\) has been deleted from \(X_e\) or not. Further, if a non-trivial \(C_j\) has at least a constant fraction of \(n (\delta n)\) vertices then the set of vertices of \(C_j\) is not kept in the list. If there are at most a constant number \((c)\) of non-trivial components then the sets of vertices of the components are not kept in the list. In these cases, when an access to vertices of a non-trivial component is needed, we check \(I_e\) to find the vertices of the component. It is easy to see that this does not increase the order of the time complexity of the implementation. A smaller \(\delta\) saves more memory but may give a larger running time. Similarly, a larger \(c\) saves more memory but may increase the running time. We have chosen \(\delta = 1/100\) and \(c = 100\) in this study. A distance matrix is used to keep the all pairs shortest
distances. We decide the integer type for the distance matrix based on the input integer \( k \) to ST Procedure. When \( G \) is a medial graph, we can reduce the required memory size to about \( 4n^2 \) bytes if one-byte integer arrays are used for each \( I_e \) and the distance matrix, and to about \( 7n^2 \) bytes if two-byte integer arrays are used.

More improvements

**Improvement A\(_1\)** This improvement is based on Theorem 5.3.3. In \( A\(_1\) \), the elements \((r, v), v \in U(r)\), are deleted from \( X_r \) and put to the stack at the initial step of the deletion process for face \( r \). From Lemma 5.3.2, \( U(r) \supseteq V(r) \) and computational studies show that \( |U(r)| \) is usually much larger than \( |V(r)| \). Therefore, \( A\(_1\) \) gives a room for improving both the running time and memory space.

**Improvement D\(_1\)** The features of \( D\(_1\) \) can be expressed by dynamic data creation and data compression. In \( D\(_1\) \) the data for a face (edge) are created only when ST Procedure starts to process the face (edge). When some losing states are deleted, the face/edge data are compressed. More specifically, when ST Procedure is to perform the first deletion for a face \( r \), \( U(r) \) is computed and array \( B_r \) of \( n \) elements is created. After the losing states \((r, v_i), v_i \in U(r)\), are deleted from \( X_r \), vertices of \( V(G) \setminus U(r) \) are re-indexed and array \( B_r \) is compressed to indicate if \((r, v_i)\) has been deleted for vertices \( v_i \) of \( V(G) \setminus U(r) \) only. Similarly, when ST Procedure is to perform the first deletion for an edge \( e \), \( G_e \) is computed and the edge data are created. Let \( r \) and \( r' \) be the two faces incident to \( e \). We create two integer arrays \( I_e \) and \( I'_e \) for \( e \). If the vertices of \( V(G) \setminus U(r) \) have been re-indexed and \( B_r \) has been compressed then \( I_e \) is compressed accordingly. Similarly, array \( I'_e \) is compressed for the vertices of \( V(G) \setminus U(r') \).

To calculate \( U(r) \) and \( G_e \), the shortest distances from vertex \( v^*_i \) to all other vertices in the planar dual graph \( G^* \) of \( G \) are needed for each face \( r \) in \( G \). When a distance matrix is used to keep the shortest distances, we need to solve \(|R(G)|\) single source shortest path problems. In \( D\(_1\) \), the distance matrix is discarded. When we process a face \( r \), we create the data for \( r \) and the data for \( I_e \) for all \( e \) incident to \( r \). Since each edge is incident to two faces \( r \) and \( r' \), the total number of single source shortest path calculations is bounded by \( 2|E(G)| \). When \( G \) has \( n \) vertices and is a medial graph, \(|R(G)| = n + 2\) and \(|E(G)| = 2n\). From this, if the distance matrix is used, we need to solve \( n + 2 \) single source shortest path problems while we need to solve at most \( 4n \)
such path problems if $D_1$ is applied.

Combining $D_1$ with $A_1$, the required memory size is now about $5n \times q$ bytes if one-byte integer arrays are used for $I_e$ and $I'_e$ and about $9n \times q$ if two-byte integer arrays are used, where $q$ is the average of $|V(G)\setminus U(r)|$. For the Delaunay triangulation instances tested, $q$ is less than $0.3n$ (instances dependent).

**Improvement A$_2$** This improvement is based on Theorem 5.3.4. For each face $r$, instead of $S_r$, $A_2$ initializes $X_r$ to include all states of $S'_r$.

**Improvement A$_3$** This improvement is based on Theorem 5.3.5 and performs re-calculation for edge data. $A_3$ keeps the face data once they are created but keeps the edge data for only a pre-defined maximum number of edges. Once this number is reached $A_3$ starts to delete the entire $X_e$ for some edge $e$. If a deleted $X_e$ is needed again, $X_e$ is re-computed from $X_r$, where $r$ is incident to $e$.

**Improvement D$_2$** In $D_2$, we use a bit vector $B_r$ for the data of face $r$, with one bit for one element of $X_r$. The memory size for face data is $1/8$ of that when a one-byte Boolean array is used. But more complex bit operations have to be used.

It is easy to check that all improvements except $A_3$ do not change the order of running time of ST Procedure. However, applying $A_3$, the running time of ST Procedure may become $O(n^3)$.

### 5.3.3 Computational Results

All of our efficient implementations use common improvements. In our implementations with any of improvements $A_2, A_3$ and $D_2$, improvements $A_1$ and $D_1$ are always used. We do not mention $A_1$ and $D_1$ explicitly in those implementations. We test Naive and Implementations $A_1, A_1D_1, A_2, A_2D_2, A_3, A_3D_2, A_2A_3$, and $A_2A_3D_2$. To show that our implementations work well for a broad class of planar graphs, three classes of instances are used in the test: one class is the benchmark instances from previous studies, and the other two classes are random planar graphs generated by two well-used software libraries, LEDA and PIGALE, respectively. More specifically, Class (1) of instances includes Delaunay triangulations of point sets taken from TSPLIB [107]. Those instances are used as test instances in the previous studies [70, 71]. The instances in Class (2) are generated by the LEDA
library [1, 90]. LEDA generates two types of planar graphs. One type of the graphs are the randomly generated maximal planar graphs and their subgraphs obtained from deleting some edges. Since the maximal planar graphs generated by LEDA always have branchwidth four, the subgraphs obtained by deleting edges from the maximal graphs have branchwidth at most four. The graphs of this type are not interesting for the study of branchwidth and branch decompositions. The other type of planar graphs are those generated based on some geometric properties, including Delaunay triangulations and triangulations of points uniformly distributed in a two-dimensional plane, and the intersection graphs of segments uniformly distributed in a two-dimensional plane. We will report the results on the intersection graphs. The instances in Class (3) are generated by the PIGALE library [4]. PIGALE randomly generates one of all possible planar graphs with a given number of edges based on the algorithms of [112]. We use Naive and our implementations to compute the carving-width of the medial graphs of the instances (i.e., the input graph to ST Procedure is not an instance itself but the medial graph of the instance). Our implementations are tested on a computer with Intel(R) Xeon(TM) 3.06GHz CPU, 2Gbytes physical memory and 8Gbytes swap memory. The operating system is SUSE LINUX 10.0, and the programming language we used is C++.

We compute an upper bound on the carvingwidth as the initial guessed input integer $k$ to call ST Procedure. It is known that the branchwidth of a planar graph of $n$ vertices is at most $\sqrt{4.5n}$ [57]. From this, $2\sqrt{4.5n}$ is an upper bound on the carvingwidth of the medial graph of an instance of $n$ vertices. We follow a similar approach in [70] to compute another upper bound $l$: Let $M(G)$ be a medial graph of a planar graph $G$ of $n$ vertices. For each face $r$ of $M(G)$ which corresponds to a vertex in $G$, we compute the eccentricity of $v_r^*$ (the length of the longest shortest paths from $v_r^*$ to all other vertices) in the planar dual $M(G)^*$. We initialize $l$ as twice as the minimum eccentricity among all $v_r^*$'s. Finally, we take $k = \min\{2\sqrt{4.5n}, l\}$. Either the linear search or the binary search can be used to find the carvingwidth starting from the initial guessed $k$. In the linear search, when the rat-catcher wins, $k$ is decreased by two and ST Procedure is called again until the rat wins the game. In the binary search, we call ST Procedure to search for the carvingwidth between $k$ (upper bound) and the node degree of $M(G)$ (which is four and a lower bound). For the instances in Classes (1) and (2), the eccentricity-based guess is very close to the carvingwidth and $k$ always takes the value of $l$. The linear search uses a smaller number of iterations to find the carvingwidth than the binary search. For instances in Class (3), the eccentricity-based
guess could be very large and $k$ may take $2\sqrt{4.5n}$ for large instances. Since $2\sqrt{4.5n}$ is still far away from the carvingwidth, the binary search does a better job. One may run the linear search and binary search in parallel and take the results from the one which finishes earlier.

**Computation time and memory**

Table 5.1 shows the computation time of Naive and efficient implementations for the carvingwidth of the medial graphs of the instances in Class (1). In the table, $Itr$ is the number of iterations in the linear search. Table 5.2 shows the memory size (in megabytes) of those implementations. Only the data for relatively large instances are given in the tables.

For the instances in Class (1), one-byte integer arrays are used for each edge and the distance matrix. The most time efficient implementation is $A_1$ which is faster than Naive by a factor of at least 10 and uses at most $1/10$ memory of Naive. With more improvements, the memory requirement is further reduced but the running time is slightly increased. The effect of data compression in Improvement $D_1$ is significant. The memory used by $A_1D_1$ is only about $1/3 \sim 1/4$ of that by $A_1$. Improvement $A_2$ is effective in reducing the memory size. In general, the number of non-trivial components is small for both faces and edges, and thus the memory saving is big. The memory used by Improvement $D_2$ for face data is $1/8$ of that when one-byte Boolean arrays are used. When the memory for face data becomes dominating, Improvement $D_2$ reduces memory requirement significantly. The most memory efficient implementation without re-calculation is $A_2D_2$ which is faster than Naive by a factor of $8 \sim 9$. For Instance $pla33810$ which has 101,367 edges (corresponding to 101,367 vertices in the input medial graph), $A_2D_2$ uses about 500Mbytes memory, which is about $\frac{1}{20}n^2$ bytes, where $n$ is the number of vertices in the input medial graph. Compared with Naive, the memory saving is by a factor of about 1000.

Improvement $A_3$ performs re-calculation for edge data. The performance depends on the maximum number of edges that are kept. This maximum number can be chosen based on the size of available memory. In general, a larger maximum number gives an implementation which uses more memory but runs faster (less re-calculations). The maximum number of kept edges is 500 for the results presented unless otherwise stated explicitly. Among all implementations, $A_2A_3D_2$ is the most memory efficient one. $A_2A_3D_2$ is faster than Naive by a factor of $6 \sim 7$. For Instance $pla33810$, $A_2A_3D_2$ uses less than 200Mbytes memory, which is about $\frac{1}{50}n^2$ bytes. Compared with Naive, the memory saving is by a factor of about 2500. The memory used by Implementation $A_2A_3D_2$ can be further reduced to about
Table 5.1: Computation time (in seconds) of Naive and efficient implementations for Class (1) instances. An X in the table indicates that the implementation runs out of 2Gbyte memory for that instance.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Number of edges</th>
<th>bw</th>
<th>Itr</th>
<th>Computation time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Naive</td>
</tr>
<tr>
<td>pr1002</td>
<td>2,972</td>
<td>21</td>
<td>2</td>
<td>51.2</td>
</tr>
<tr>
<td>rl1323</td>
<td>3,950</td>
<td>22</td>
<td>2</td>
<td>95.7</td>
</tr>
<tr>
<td>d1655</td>
<td>4,890</td>
<td>29</td>
<td>2</td>
<td>158</td>
</tr>
<tr>
<td>rl1889</td>
<td>5,631</td>
<td>22</td>
<td>2</td>
<td>195</td>
</tr>
<tr>
<td>u2152</td>
<td>6,312</td>
<td>31</td>
<td>3</td>
<td>X</td>
</tr>
<tr>
<td>pr2392</td>
<td>7,125</td>
<td>29</td>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>pcb3038</td>
<td>9,101</td>
<td>40</td>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>fl3795</td>
<td>11,326</td>
<td>25</td>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>fnl4461</td>
<td>13,359</td>
<td>48</td>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>rl5934</td>
<td>17,770</td>
<td>41</td>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>pla7397</td>
<td>21,865</td>
<td>33</td>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>usa13509</td>
<td>40,503</td>
<td>63</td>
<td>4</td>
<td>X</td>
</tr>
<tr>
<td>brd14051</td>
<td>42,128</td>
<td>68</td>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>d15112</td>
<td>45,310</td>
<td>78</td>
<td>3</td>
<td>X</td>
</tr>
<tr>
<td>d18512</td>
<td>55,510</td>
<td>88</td>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>pla38310</td>
<td>101,367</td>
<td>100</td>
<td>5</td>
<td>X</td>
</tr>
</tbody>
</table>
Table 5.2: Memory usage (in megabytes) of Naive and efficient implementations for Class (1) instances. An *X* in the table indicates that the implementation runs out of 2Gbyte memory for that instance.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Number of edges</th>
<th>Naive</th>
<th>$A_1$</th>
<th>$A_1D_1$</th>
<th>$A_2$</th>
<th>$A_2D_2$</th>
<th>$A_3$</th>
<th>$A_3D_2$</th>
<th>$A_2A_3$</th>
<th>$A_2A_3D_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr1002</td>
<td>2,972</td>
<td>413</td>
<td>39</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>rl1323</td>
<td>3,950</td>
<td>734</td>
<td>66</td>
<td>23</td>
<td>11</td>
<td>10</td>
<td>15</td>
<td>13</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>d1655</td>
<td>4,890</td>
<td>1,188</td>
<td>99</td>
<td>30</td>
<td>14</td>
<td>11</td>
<td>17</td>
<td>14</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>rl1889</td>
<td>5,631</td>
<td>1,424</td>
<td>130</td>
<td>46</td>
<td>18</td>
<td>14</td>
<td>28</td>
<td>21</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>u2152</td>
<td>6,312</td>
<td>X</td>
<td>161</td>
<td>41</td>
<td>17</td>
<td>14</td>
<td>26</td>
<td>20</td>
<td>16</td>
<td>13</td>
</tr>
<tr>
<td>pr2392</td>
<td>7,125</td>
<td>X</td>
<td>204</td>
<td>66</td>
<td>21</td>
<td>17</td>
<td>35</td>
<td>26</td>
<td>19</td>
<td>15</td>
</tr>
<tr>
<td>pcb3038</td>
<td>9,101</td>
<td>X</td>
<td>328</td>
<td>76</td>
<td>25</td>
<td>21</td>
<td>36</td>
<td>27</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>fl3795</td>
<td>11,326</td>
<td>X</td>
<td>504</td>
<td>132</td>
<td>58</td>
<td>42</td>
<td>66</td>
<td>63</td>
<td>40</td>
<td>23</td>
</tr>
<tr>
<td>fnl4461</td>
<td>13,359</td>
<td>X</td>
<td>698</td>
<td>158</td>
<td>39</td>
<td>32</td>
<td>66</td>
<td>43</td>
<td>33</td>
<td>26</td>
</tr>
<tr>
<td>rl5934</td>
<td>17,770</td>
<td>X</td>
<td>1,226</td>
<td>358</td>
<td>67</td>
<td>51</td>
<td>155</td>
<td>86</td>
<td>50</td>
<td>35</td>
</tr>
<tr>
<td>pla7397</td>
<td>21,865</td>
<td>X</td>
<td>1,850</td>
<td>436</td>
<td>123</td>
<td>85</td>
<td>238</td>
<td>144</td>
<td>83</td>
<td>44</td>
</tr>
<tr>
<td>usa13509</td>
<td>40,503</td>
<td>X</td>
<td>X</td>
<td>1,534</td>
<td>220</td>
<td>153</td>
<td>498</td>
<td>271</td>
<td>149</td>
<td>79</td>
</tr>
<tr>
<td>brd14051</td>
<td>42,128</td>
<td>X</td>
<td>X</td>
<td>1,600</td>
<td>215</td>
<td>149</td>
<td>580</td>
<td>283</td>
<td>149</td>
<td>82</td>
</tr>
<tr>
<td>d15112</td>
<td>45,310</td>
<td>X</td>
<td>X</td>
<td>1,795</td>
<td>227</td>
<td>156</td>
<td>508</td>
<td>256</td>
<td>156</td>
<td>86</td>
</tr>
<tr>
<td>d18512</td>
<td>55,510</td>
<td>X</td>
<td>X</td>
<td>284</td>
<td>198</td>
<td>706</td>
<td>328</td>
<td>194</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>pla33810</td>
<td>101,367</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>814</td>
<td>508</td>
<td>X</td>
<td>876</td>
<td>507</td>
<td>198</td>
</tr>
</tbody>
</table>
155Mbytes for Instance pla33810 with a slightly increase in the running time, if we keep at most 50 edges.

The instances in Class (2) are generated by the LEDA function random_planar_graph [1]. We have tested our implementations on instances of Delaunay triangulations and triangulations of points randomly distributed in a two-dimensional plane, and intersection graphs of segments. Our implementations have similar performances for the Delaunay triangulations and triangulations instances as those for the instances in Class (1). Table 5.3 gives the computation time and memory of Naive and Implementations $A_1, A_2 D_2$, and $A_2 A_3 D_2$ for instances of intersection graphs of segments. In the table, $Itr$ is the number of iterations in the linear search. The instances of intersection graphs of segments may have a large number of non-trivial components for edges and faces, and two-byte integer arrays are used to represent the edge data. Therefore, the memory usage is considerably larger than the Delaunay instances of the same size. As shown in the table, our efficient implementations are faster and use much less memory than Naive.

Instances of Class (3) are generated by the PIGALE library [4]. PIGALE provides a number of planar graph generators. Since 2-connected planar graphs are the most interesting class of graphs in the study of branchwidth and branch decompositions, we selected the function for generating 2-connected planar graphs. The function, given the number $m$ of edges, randomly generates one of all possible 2-connected planar graphs of $m$ edges. The output graph is usually a multi-graph with parallel edges. Since parallel edges are not interesting for branchwidth finding, we specify the function to produce simple 2-connected graphs. With a given $m$, the function outputs a 2-connected random planar graph with $m'$ edges. Normally $m'$ is smaller than $m$, since parallel edges are not kept and there are performance considerations [4]. Table 5.4 gives the computation time and memory of Naive and Implementations $A_1, A_2 D_2$, and $A_2 A_3 D_2$. In the table, $Itr$ is the number of iterations in the binary search. The instances in Class (3) may have a small number of non-trivial edge components, but we still use two-byte integer arrays for the edge data. For this class of instances, the eccentricity-based guess is usually bad. For example, the medial graph of Instance P137730 has carvingwidth 12, but the eccentricity-based guess is 8974. For large instances tested, the binary search always finishes earlier than the linear search. The number of iterations in the binary search is about 10 for large instances, and this prohibits us from solving very large instances in a reasonable time. The memory usage for the PIGALE instances is very small, compared to the instances of Classes (1) and (2) even two-byte
integer arrays are used for edge data.

For the instances in Class (3), Implementation $A_2D_2$ is very memory efficient. This indicates that the numbers of non-trivial components for faces in those instances are small. The gap between the running time of Implementation $A_1$ and that of Implementation $A_2D_2$ is a little bigger for instances of this class than the gap for instances in the other two classes. This can be explained by the following reasons. Naive and Implementation $A_1$ keep the shortest distance matrix, while $A_2D_2$ discards the matrix. As analyzed in Improvement $D_1$, Naive and $A_1$ need to solve $n + 2$ single source shortest path problems while $A_2D_2$ may need to solve $4n$ such problems, where $n$ is the number of vertices of the input medial graph. $A_2D_2$ also needs to calculate the components of $G_r$ for every face $r$ and there is no such computation in Naive and $A_1$. Notice that each of the shortest distances calculation, the computation of components of $G_r$ for all $r$, and the deletion process takes $O(n^2)$ time. For instances of Classes (1) and (2), the time of deletion process is larger than the sum of the other two. However, the deletion process runs faster for the instances of Class (3) than for instances of Classes (1) and (2). In this case, the shortest distances calculation and the computation of components of $G_r$ may become a dominating part of the total running time.

Among all implementations, the most time efficient one is $A_1$. Compared with Naive, $A_1$ is faster by a factor of $3 \sim 15$. The memory saving of $A_1$ is also significant. $A_1$ can solve an instance of about 20,000 edges in Class (1) and instances of about 15,000 edges in Classes (2) and (3), while Naive can only solve instances of size up to about 5,000 edges for all three classes. The most memory efficient implementation without re-calculation is $A_2D_2$. It can solve an instance of about 100,000 edges in Class (1) by about 3.5 hours and 500 Mbytes, an instance of about 60,000 edges in Class (2) by about 1.5 hours and 1.5Gbytes memory, and an instance of about 100,000 edges in Class (3) by about 14 hours and 200 Mbytes. Implementation $A_2A_3D_2$ is the most memory efficient one among all implementations. It can solve an instance of about 100,000 edges in Class (1) by about 6 hours and 200 Mbytes, an instance of about 100,000 edges in Class (2) by about 6 hours and 1.4Gbytes, and an instance of about 70,000 edges in Class (3) by by about 14 hours and 160 Mbytes. All implementations without using $A_3$ have time complexity $O(n^2)$ since no re-calculation for edge data is performed. In the worst case, Implementation $A_2A_3D_2$ may perform re-calculation repeatedly for some edges and has time complexity $O(n^3)$. However, the worst case scenario has not been observed and the running time of Implementation $A_2A_3D_2$ is at most as twice as that of Implementation $A_2D_2$ for most instances.
Table 5.3: Computation time and memory of intersection graphs of segments generated by LEDA. An $X$ in the table indicates that the implementation runs out of 2Gbyte memory for that instance.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Number of edges</th>
<th>$bw$</th>
<th>$Itr$</th>
<th>Time (seconds)</th>
<th>Memory (MByte)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Naive</td>
<td>$A_1$</td>
</tr>
<tr>
<td>rand1300</td>
<td>2,030</td>
<td>7</td>
<td>6</td>
<td>51.1</td>
<td>10.4</td>
</tr>
<tr>
<td>rand1900</td>
<td>3,029</td>
<td>8</td>
<td>5</td>
<td>102</td>
<td>15.7</td>
</tr>
<tr>
<td>rand3050</td>
<td>5,032</td>
<td>9</td>
<td>4</td>
<td>283</td>
<td>32.5</td>
</tr>
<tr>
<td>rand6000</td>
<td>10,261</td>
<td>12</td>
<td>2</td>
<td>$X$</td>
<td>95.5</td>
</tr>
<tr>
<td>rand8700</td>
<td>15,990</td>
<td>14</td>
<td>3</td>
<td>$X$</td>
<td>292</td>
</tr>
<tr>
<td>rand11500</td>
<td>20,279</td>
<td>13</td>
<td>2</td>
<td>$X$</td>
<td>$X$</td>
</tr>
<tr>
<td>rand17000</td>
<td>30,433</td>
<td>14</td>
<td>2</td>
<td>$X$</td>
<td>$X$</td>
</tr>
<tr>
<td>rand22500</td>
<td>40,622</td>
<td>18</td>
<td>2</td>
<td>$X$</td>
<td>$X$</td>
</tr>
<tr>
<td>rand28000</td>
<td>50,901</td>
<td>18</td>
<td>3</td>
<td>$X$</td>
<td>$X$</td>
</tr>
<tr>
<td>rand33000</td>
<td>60,398</td>
<td>20</td>
<td>2</td>
<td>$X$</td>
<td>$X$</td>
</tr>
<tr>
<td>rand54000</td>
<td>100,037</td>
<td>22</td>
<td>2</td>
<td>$X$</td>
<td>$X$</td>
</tr>
</tbody>
</table>
Table 5.4: Computation time and memory of random instances generated by PIGALE. An $X$ in the table indicates that the implementation runs out of 2Gbyte memory for that instance.

<table>
<thead>
<tr>
<th>m</th>
<th>Instances</th>
<th>Number of edges</th>
<th>$bw$</th>
<th>$litr$</th>
<th>Time (seconds)</th>
<th>Memory (MByte)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Naive $A_1$ $A_2D_2$ $A_2A_3D_2$</td>
<td>Naive $A_1$ $A_2D_2$ $A_2A_3D_2$</td>
</tr>
<tr>
<td>2,400</td>
<td>PI1180</td>
<td>2,022</td>
<td>7</td>
<td>5</td>
<td>23 6.73 9.08 10.4</td>
<td>196 32 8 7</td>
</tr>
<tr>
<td></td>
<td>PI1182</td>
<td>2,016</td>
<td>7</td>
<td>5</td>
<td>22.6 7.44 10.4 12.1</td>
<td>190 32 8 7</td>
</tr>
<tr>
<td></td>
<td>PI1186</td>
<td>2,029</td>
<td>6</td>
<td>5</td>
<td>23.5 6.56 9.24 10</td>
<td>205 32 7 7</td>
</tr>
<tr>
<td></td>
<td>PI1193</td>
<td>2,019</td>
<td>6</td>
<td>5</td>
<td>19.1 7.39 10.5 11.1</td>
<td>156 32 7 7</td>
</tr>
<tr>
<td></td>
<td>PI1207</td>
<td>2,029</td>
<td>9</td>
<td>5</td>
<td>30.6 5.41 6.64 7.04</td>
<td>167 32 7 7</td>
</tr>
<tr>
<td>6,000</td>
<td>PI2995</td>
<td>5,043</td>
<td>7</td>
<td>6</td>
<td>156 58.7 93.3 96.5</td>
<td>1,034 178 14 13</td>
</tr>
<tr>
<td></td>
<td>PI2996</td>
<td>5,015</td>
<td>8</td>
<td>7</td>
<td>210 72.1 115 117</td>
<td>1,119 176 16 14</td>
</tr>
<tr>
<td></td>
<td>PI2998</td>
<td>5,049</td>
<td>7</td>
<td>6</td>
<td>163 58.3 92.4 102</td>
<td>1,090 178 17 15</td>
</tr>
<tr>
<td></td>
<td>PI3017</td>
<td>5,063</td>
<td>8</td>
<td>7</td>
<td>197 70.9 111 117</td>
<td>1,112 179 15 14</td>
</tr>
<tr>
<td></td>
<td>PI3018</td>
<td>5,074</td>
<td>7</td>
<td>6</td>
<td>158 57.4 88.2 95.5</td>
<td>986 180 15 14</td>
</tr>
<tr>
<td>12,000</td>
<td>PI5940</td>
<td>10,016</td>
<td>7</td>
<td>8</td>
<td>X 289 522 563</td>
<td>X 683 28 26</td>
</tr>
<tr>
<td></td>
<td>PI5992</td>
<td>10,101</td>
<td>8</td>
<td>7</td>
<td>X 286 558 580</td>
<td>X 695 27 25</td>
</tr>
<tr>
<td></td>
<td>PI5998</td>
<td>10,144</td>
<td>7</td>
<td>8</td>
<td>X 304 555 583</td>
<td>X 701 26 24</td>
</tr>
<tr>
<td></td>
<td>PI6043</td>
<td>10,146</td>
<td>8</td>
<td>8</td>
<td>X 299 550 576</td>
<td>X 701 30 27</td>
</tr>
<tr>
<td></td>
<td>PI6067</td>
<td>10,173</td>
<td>7</td>
<td>8</td>
<td>X 312 555 584</td>
<td>X 705 26 25</td>
</tr>
<tr>
<td>18,000</td>
<td>PI8950</td>
<td>15,097</td>
<td>10</td>
<td>9</td>
<td>X 907 1,771 2,089</td>
<td>X 1,541 48 39</td>
</tr>
<tr>
<td></td>
<td>PI8977</td>
<td>15,065</td>
<td>9</td>
<td>9</td>
<td>X 913 1,791 1,971</td>
<td>X 1,535 43 38</td>
</tr>
<tr>
<td></td>
<td>PI8986</td>
<td>15,058</td>
<td>8</td>
<td>8</td>
<td>X 765 1,559 1,643</td>
<td>X 1,533 42 39</td>
</tr>
<tr>
<td></td>
<td>PI8995</td>
<td>15,080</td>
<td>9</td>
<td>9</td>
<td>X 885 1,787 1,911</td>
<td>X 1,538 41 38</td>
</tr>
<tr>
<td></td>
<td>PI9020</td>
<td>15,053</td>
<td>8</td>
<td>8</td>
<td>X 807 1,582 1,688</td>
<td>X 1,532 44 37</td>
</tr>
<tr>
<td>24,000</td>
<td>PI11974</td>
<td>20,071</td>
<td>9</td>
<td>9</td>
<td>X X 3,646 3,702</td>
<td>X X 46 44</td>
</tr>
<tr>
<td>35,000</td>
<td>PI17495</td>
<td>30,003</td>
<td>7</td>
<td>9</td>
<td>X X 8,597 8,610</td>
<td>X X 70 67</td>
</tr>
<tr>
<td>46,000</td>
<td>PI22640</td>
<td>40,074</td>
<td>5</td>
<td>9</td>
<td>X X 14,163 14,210</td>
<td>X X 94 89</td>
</tr>
<tr>
<td>56,000</td>
<td>PI27671</td>
<td>50,095</td>
<td>8</td>
<td>10</td>
<td>X X 24,684 24,702</td>
<td>X X 118 114</td>
</tr>
<tr>
<td>66,000</td>
<td>PI32943</td>
<td>60,634</td>
<td>8</td>
<td>10</td>
<td>X X 36,909 37,076</td>
<td>X X 156 138</td>
</tr>
<tr>
<td>76,000</td>
<td>PI37730</td>
<td>70,022</td>
<td>6</td>
<td>10</td>
<td>X X 49,136 49,180</td>
<td>X X 188 160</td>
</tr>
</tbody>
</table>
CHAPTER 5. BRANCH/CARVING DECOMPOSITION BASED ALGORITHMS

Computation time of one iteration

To find the carvingwidth of a planar graph, ST Procedure is usually called multiple times. The number of calls (iterations) is instance dependent. In computing the branch decompositions, the computation time of one iteration is an important measure for the time efficiency. Table 5.5 shows the computation time of Naive and Implementations $A_1$, $A_2D_2$, and $A_2A_3D_2$ in the iteration when the rat wins the game and the iteration when the rat-catcher wins the game with the noisy level $k$ closest to the carvingwidth for some instances in Classes (1), (2), and (3). From Observation 5.3.6, the deletion process of ST Procedure may terminate earlier when the rat-catcher wins the game. It can be seen from the table that ST Procedure generally uses much less time when the rat-catcher wins for instances of Classes (1) and (2). For instances in Class (3), the computation time when the rat wins is not much different from that when the rat-catcher wins because the time of the deletion process is not a dominating part of the total running time.

Comparison with previous works

Hicks proposes a straightforward implementation $rat$ and two memory friendly implementations $comprat$ and $memrat$ of ST Procedure [70]. The implementations are tested using instances of Class (1) on a SGI Power Challenge with $6 \times 194$ MHz processors, 1Gbytes of physical memory, and 1Gbytes of swap space. To compare our results with Hicks', we quote some data of [70] in Table 5.6. An $M$ in the table indicates that the implementation runs out of 2Gbyte memory for that instance. From the table, $rat$ runs out of 2Gbyte memory for instances of r11889 (5,631 edges) and larger. Our Naive implementation can solve r11889 but runs out of 2Gbyte memory for instances of u2152 (6,312 edges) and larger. This confirms that straightforward implementations of ST Procedure are memory consuming. The memory used by $memrat$ for Instance brd14051 (the largest one reported in [70]) is about 600Mbytes. For the same instance, $A_2D_2$ uses about 1/4 and $A_2A_3D_2$ uses about 1/8 of the memory of $memrat$. Implementation $A_1$ is faster by a factor of 200 ~ 500, Implementation $A_2D_2$ is faster by a factor of 100 ~ 400, and Implementation $A_2A_3D_2$ is faster by a factor of 100 ~ 200 than $comprat$ and $memrat$ for large instances. Notice that the CPU used in [70] has frequency 194MHz and the CPU used our studies has frequency 3.06GHz, so we need to keep in mind this difference of speed when we compare the running time.
Table 5.5: Computation time (in seconds) of several implementations for \( k \) close to the carvingwidth \( 2bw \). An \( X \) in the table indicates that the implementation runs out of 2Gbyte memory for that instance.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Number of edges</th>
<th>( k = 2(bw + 1) )</th>
<th>( k = 2bw )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Naive</td>
<td>( A_1 )</td>
<td>( A_2D_2 )</td>
</tr>
<tr>
<td>rl1889</td>
<td>5,631</td>
<td>87.6</td>
<td>0.196</td>
</tr>
<tr>
<td>usa13509</td>
<td>40,503</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>d15112</td>
<td>45,310</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>rand3050</td>
<td>5,032</td>
<td>59.4</td>
<td>4.33</td>
</tr>
<tr>
<td>rand22500</td>
<td>40,622</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>rand33000</td>
<td>60,398</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PI2995</td>
<td>5,043</td>
<td>12.1</td>
<td>10.2</td>
</tr>
<tr>
<td>PI22640</td>
<td>40,074</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PI32943</td>
<td>60,634</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 5.6: Computation time (in seconds) of \( \text{rat} \), \( \text{comrat} \), and \( \text{memrat} \) quoted from Table 1 of [70]. An \( \mathcal{M} \) in the table indicates that the implementation runs out of 2Gbyte memory for that instance.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Number of edges</th>
<th>( bw )</th>
<th>( Itr )</th>
<th>Computation time (in seconds).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( \text{rat} )</td>
</tr>
<tr>
<td>pr1002</td>
<td>2,972</td>
<td>21</td>
<td>2</td>
<td>338</td>
</tr>
<tr>
<td>rl1323</td>
<td>3,950</td>
<td>22</td>
<td>3</td>
<td>876</td>
</tr>
<tr>
<td>d1655</td>
<td>4,890</td>
<td>29</td>
<td>3</td>
<td>1,318</td>
</tr>
<tr>
<td>rl1889</td>
<td>5,631</td>
<td>22</td>
<td>3</td>
<td>M</td>
</tr>
<tr>
<td>u2152</td>
<td>6,312</td>
<td>31</td>
<td>4</td>
<td>M</td>
</tr>
<tr>
<td>pr2392</td>
<td>7,125</td>
<td>29</td>
<td>3</td>
<td>M</td>
</tr>
<tr>
<td>pcb3038</td>
<td>9,101</td>
<td>40</td>
<td>4</td>
<td>M</td>
</tr>
<tr>
<td>fl3795</td>
<td>11,326</td>
<td>25</td>
<td>3</td>
<td>M</td>
</tr>
<tr>
<td>fnl4461</td>
<td>13,359</td>
<td>48</td>
<td>4</td>
<td>M</td>
</tr>
<tr>
<td>rl5934</td>
<td>17,770</td>
<td>41</td>
<td>3</td>
<td>M</td>
</tr>
<tr>
<td>pla7397</td>
<td>21,865</td>
<td>33</td>
<td>2</td>
<td>M</td>
</tr>
<tr>
<td>usa13509</td>
<td>40,503</td>
<td>63</td>
<td>1/2</td>
<td>M</td>
</tr>
<tr>
<td>brd14051</td>
<td>42,128</td>
<td>68</td>
<td>3</td>
<td>M</td>
</tr>
</tbody>
</table>
5.4 The Edge-contraction Method for Branch Decompositions

Seymour and Thomas give an algorithm, which is known as edge contraction method, for computing an optimal branch decomposition of a planar graph [113]. The contraction of an edge $e$ in a graph $G$ is to remove $e$ from $G$, identify the two end vertices of $e$ by a new vertex, and make all edges incident to $e$ incident to the new vertex. We denote by $G/e$ the graph obtained by contracting $e$ in $G$. Given a 2-connected planar graph $G$, the algorithm of Seymour and Thomas computes an optimal branch decomposition of $G$ by a sequence of edge contractions of the medial graph $M(G)$ of $G$ as follows: First the carvingwidth $cw$ of $M(G)$ is computed by ST Procedure. An edge $e$ of $M(G)$ is contractible if the carvingwidth of $M(G)/e$ is at most $cw$ and $M(G)/e$ is 2-connected. Next, a contractible edge $e$ of $M(G)$ is found by ST Procedure and $M(G)$ is contracted to graph $M(G)/e$. The contraction is repeated on $M(G)/e$ until the graph becomes one with three vertices. A optimal carving decomposition of $M(G)$ with width at most $cw$ is constructed based on the sequence of edge contractions.

**Proposition 5.4.1** (Seymour and Thomas [113]) Let $e = \{x, y\}$ be a contractible edge of $M(G)$, $x_e$ be the new vertex identifying $\{x, y\}$ in $M(G)/e$, and $T'_C$ be an optimal carving decomposition of $M(G)/e$. Then the carving decomposition $T_C$ obtained by adding links $\{x_e, x\}$ and $\{x_e, y\}$ to $T'_C$ is an optimal carving decomposition of $M(G)$.

Finally, the branch decomposition of $G$ is obtained from the carving decomposition of $M(G)$ in linear time (Proposition 5.2.1). It is proved in [113] that for any 2-connected planar graph there is a contractible edge and for a 2-connected planar graph $G$, $M(G)$ is 2-connected. To check if an edge is contractible, ST Procedure is used to test if $M(G)/e$ has carvingwidth at most $cw$. In the worst case, all edges may be checked to find a contractible one and for a graph of $n$ vertices, the algorithm of Seymour and Thomas may call ST Procedure $O(n)$ times for one contraction and $O(n^2)$ times in total. So the time complexity of the algorithm is $O(n^4)$.

We call a contractibility test on an edge a positive one if the edge is tested contractible, otherwise a negative one. Gu and Tamaki give an algorithm which uses a better strategy to find positive tests [66]. When a negative test is obtained on an edge then the edge will not be tested again unless a necessary condition for that edge to be contractible is satisfied. By
this improvement, the algorithm of Gu and Tamaki avoids the repeated negative tests on a same edge, calls ST Procedure $O(n)$ times, and has time complexity $O(n^3)$ for computing an optimal branch decomposition of a planar graph.

We test the $O(n^4)$ time algorithm of Seymour and Thomas and the $O(n^3)$ time algorithm of Gu and Tamaki for instances in three classes using a number of heuristics to select edges for testing the contractibility. Implementation $A_1$, the most time efficient one, is used as ST Procedure. Both the algorithms have the minimum number of negative calls and running time when the round robin edge selection heuristic is used. Our computational results show that optimal branch decompositions of planar graphs of a few thousands edges can be computed in a practical time. For most instances tested, repeated negative tests are not observed on any edge in the algorithm of Seymour and Thomas. So the advantage of the algorithm of Gu and Tamaki is not shown by those instances when the round robin edge selection heuristic is used. On some other edge selection heuristic, more repeated negative tests are observed in the algorithm of Seymour and Thomas. In this case, the algorithm of Gu and Tamaki has much less negative calls and runs faster than the algorithm of Seymour and Thomas. The details can be found in [24].

5.5 Branch Decomposition of Large Planar Graphs

For large instances, computing optimal branch decompositions is still time consuming by the edge contraction method. Hicks reports that the divide-and-conquer approach is more practical to compute the branch decomposition of planar graphs [69, 71]. In this approach, first the branchwidth $\beta$ of a graph $G$ is computed. Let $S$ be a set of vertices that separates $G$ into two subgraphs. Roughly speaking, a partition by $S$ is valid if $|S| \leq \beta$, and each subgraph has branchwidth at most $\beta$ (a formal definition on the valid partition will be given later). Next a valid partition of $G$ is found. In this step, ST Procedure is used to test if each subgraph has branchwidth at most $\beta$. If a valid partition is found, then the branch decomposition of each subgraph is computed recursively. The branch decomposition of $G$ is constructed from the decompositions of the subgraphs. How to find a valid partition efficiently is a key for this approach. Hicks proposes the cycle method for computing a valid partition [69, 71]. Notice that there is no guarantee on the existence of a valid partition in a recursive step. The edge-contraction method is used to make a progress in the cycle method when a valid partition can not be found. In the worst case, the cycle method has
time complexity $O(n^4)$. Computational results show that the cycle method is faster than the edge-contraction method by a factor of about $10 \sim 30$ on average for the Delaunay triangulation instances [71].

In this section, we propose divide-and-conquer based algorithms for computing planar branch decompositions. Our algorithms are similar to the cycle method in finding a valid partition but make effort to balance the sizes of subgraphs. Our algorithms also use the edge-contraction method to make a progress when a valid partition can not be found, as is done in previous study [71]. In the worst case, our algorithms run in $O(n^3)$ time. We tested our algorithms and the $O(n^3)$ time edge-contraction algorithm [66] on several classes of planar graphs. Computational results show that our algorithms are faster than the edge-contraction algorithm by a factor of about $10 \sim 30$ on average for the Delaunay triangulation instances of more than 5,000 edges. Using the more efficient implementations of ST Procedure of [24], our algorithms can compute optimal decompositions for some instances of size up to 50,000 edges in a practical time. Previous results of the cycle method [69, 71] are obtained by a slower computer and a less efficient implementation of ST Procedure than those in this study. To compare our algorithms with the cycle method on a same platform, we implemented the unaltered cycle method [71] using the more efficient implementation of ST Procedure. Computational results show that our algorithms are faster than the unaltered cycle method by a factor of more than 10 for the Delaunay triangulation instances. Notice that our implementation of the unaltered cycle method is a straightforward one based on the information available in the published literature [69, 71].

Our results suggest that the optimal branch decompositions of large planar graphs can be computed in practice. Our divide-and-conquer algorithms are efficient tools for finding such branch decompositions. This may make the branch-decomposition based algorithms more attractive for many problems in planar graphs.

5.5.1 Divide-and-conquer Based Algorithms

Following the divide-and-conquer approach used in the cycle method [69, 71], we first describe a framework for our algorithms. Given a planar graph $H$ with carvingwidth $k$, let $C$ be a set of edges (cut set) that partitions $H$ into subgraphs $H_1$ and $H_2$. For each $H_i$ ($i = 1, 2$), define $H'_i$ to be the graph obtained by adding a new vertex $v'_i$ and the edge set $\{\{u, v'_i\}| u \in V(H_i) \cap V(C)\}$ to $H_i$ (see Figure 5.1). Intuitively, $H'_i$ is the graph of $H_i$ and a vertex $v'_i$ representing the part of $H$ other than $H_i$. The partition by $C$ is valid if $|C| \leq k$,
and each of $H'_i$ has carvingwidth at most $k$. Below is the framework for our algorithms.

1. Given a planar graph $G$, compute the medial graph $M(G)$ and the carvingwidth $k$ of $M(G)$ by ST Procedure and let $H = M(G)$.

2. If $|E(H)| > c$ ($c$ is a constant)
   - then try to find a valid partition of $H$,
     Partition $H$ into subgraphs $H_i$ ($i = 1, 2$) by a set $C$ of edges with $|C| \leq k$. If every $H'_i$ has carvingwidth at most $k$ for $i = 1, 2$, then a valid partition is found.
   - else compute the carving decomposition of $H$ by enumeration.

3. If a valid partition is found
   - then goto Step 2 to compute the carving decomposition of every subgraph $H'_i$ recursively; and construct the carving decomposition of $H$ from the carving decompositions of the subgraphs.
   - else call an edge-contraction algorithm to contract an edge $e$ of $H$ such that the contracted graph $H/e$ has carvingwidth at most $k$; goto Step 2 to compute the carving decomposition of $H/e$; and construct the carving decomposition of $H$ by Proposition 5.4.1.

4. Construct the branch decomposition of $G$ from the carving decomposition of $M(G)$ (Proposition 5.2.1).

**Lemma 5.5.1** An optimal branch decomposition of $G$ can be computed by the framework.

**Proof** By Proposition 5.4.1, if an optimal carving decomposition of $H/e$ has been found then an optimal carving decomposition of $H$ can be constructed. Assume that a valid partition of $H$ is found and optimal carving decompositions $T_1$ and $T_2$ have been constructed for subgraphs $H'_1$ and $H'_2$ in the valid partition. We assume that $T_1$ has a leaf node $v_1$ corresponding to $v'_1$ and $T_2$ has a leaf node $u_2$ corresponding to $v'_2$, added in Step 2. Let $e_1 = \{u_1, w_1\}$ be the link of $T_1$ and $e_2 = \{u_2, w_2\}$ be the link of $T_2$. We get a carving decomposition $T_C$ of $H$ by first connecting $T_1$ and $T_2$ using a new link $\{w_1, w_2\}$ and then discarding links $\{v_1, w_1\}$ and $\{u_2, w_2\}$. Obviously, each internal node of $T_C$ has degree three. Each link of $E(T_C) \setminus \{w_1, w_2\}$ has the same width as that of the corresponding link in $T_1$. 
or $T_2$. The width of link $\{w_1, w_2\}$ is $|C|$. Thus, $T_C$ has width at most $k$ and is an optimal carving decomposition of $H = M(G)$. By Proposition 5.2.1, $T_C$ can be converted to an optimal branch decomposition of $G$.

How to find a valid partition is a key on the efficiency of the divide-and-conquer algorithms. An obvious approach for finding such a partition is to compute a closed walk (cycle) $W^*$ of length at most $k$ in the planar dual $M(G)^*$ of $M(G)$. Let $E^*(W^*)$ be the set of edges in $W^*$. Let $R_{W^*}$ and $V_{W^*}$ be the sets of faces and vertices of $M(G)^*$ enclosed by $W^*$, respectively (see Figure 5.1). Then the set of edges of $M(G)$ corresponding to the edges of $E^*(W^*)$ is a cut set between the subgraph of $M(G)$ with the vertex set and face set corresponding to $R_{W^*}$ and $V_{W^*}$, respectively, and the rest part of $M(G)$.

In the cycle method [69, 71], a closed walk is computed as follows. First, a face $r^*$ of $M(G)^*$ is selected. Let $E^*_{r^*}$ be the set of edges incident to $r^*$. Next, a pair of vertices $s^*$ and $t^*$ incident to $r^*$ is selected and a shortest path $P^*$ that does not contain any edge of $E^*_{r^*}$ between $s^*$ and $t^*$ in $M(G)^*$ is computed. A path $Q^*$ between $s^*$ and $t^*$ formed by edges of $E^*_{r^*}$ and path $P^*$ give a closed walk $W^*$ of $M(G)^*$. For a selected face $r^*$, the cycle method tries every pair of vertices $s^*$ and $t^*$ incident to $r^*$. If a valid partition is found, then the method is applied recursively, otherwise the edge-contraction method is called.

Similar to the cycle method, our algorithms compute a closed walk $W^*$ formed by paths $Q^*$ and $P^*$ between $s^*$ and $t^*$. Our algorithms select the vertices $s^*$ and $t^*$ with the consideration on the sizes of subgraphs. Notice that the edges of $E^*_{r^*}$ is a closed walk. For vertices
CHAPTER 5. BRANCH/CARVING DECOMPOSITION BASED ALGORITHMS

$s^*$ and $t^*$ incident to $r^*$, there are two paths $Q_1^*$ and $Q_2^*$ formed by the edges of $E_i^*$. between $s^*$ and $t^*$. The partition may be balanced if there is a small difference between the lengths of $Q_1^*$ and $Q_2^*$. Our first algorithm chooses the vertices $s^*$ and $t^*$ in an order that a smaller difference between the lengths of $Q_1^*$ and $Q_2^*$ is selected with a higher priority. We call this procedure the length-priority algorithm.

The cut set corresponding to a closed walk $W^*$ partitions the input graph in a recursive step into two subgraphs. The size of a subgraph is the number of vertices in the subgraph. The partition is balanced if there is a small difference between the sizes of the two subgraphs. Our second algorithm chooses $s^*$ and $t^*$ in an order that a smaller difference between the sizes of the two subgraphs is selected with a higher priority. We call this procedure the size-priority algorithm.

In both algorithms, we try a constant number of pairs of vertices $s^*$ and $t^*$ incident to $r^*$ in the order defined above. If a valid partition is found then the algorithms are applied recursively, otherwise an edge-contraction method is called. In both algorithms, the constant $c$ in the framework is set to 3 and a subgraph in each partition has at least two vertices.

In the divide-and-conquer algorithms, we partition $H$ into $H_i$ ($i = 1, 2$) and test if $H_i$ has carvingwidth at most $k$ by ST Procedure. A test is called positive if $H_i$ has carvingwidth at most $k$, otherwise negative. Similarly, in the edge contraction method, we contract an edge $e$ and test if $H/e$ has carvingwidth at most $k$ by ST Procedure. A test is called positive if $H/e$ has carvingwidth at most $k$, otherwise negative.

**Theorem 5.5.2** Both the length-priority and size-priority algorithms compute an optimal branch decomposition of a planar graph $G$ of $n$ vertices in $O(n^3)$ time.

**Proof** By Lemma 5.5.1, the algorithms compute an optimal branch decomposition of $G$. The medial graph $H = M(G)$ has $|E(G)| = O(n)$ vertices. The carvingwidth of $H$ can be computed in $O(n^2 \log n)$ time by ST Procedure (using a binary search). Because the branchwidth of $G$ is $\beta = O(\sqrt{n})$ [57], the carvingwidth of $H$ is $k = 2\beta = O(\sqrt{n})$. Since the carvingwidth of a graph is at least the maximum node degree of the graph, $H$ and the subgraphs in each recursive step have node degree $O(\sqrt{n})$. Therefore, there are $O(n)$ pairs of $s^*$ and $t^*$ incident to a face $r^*$ when we try to find a valid partition. It takes $O(n)$ time to compute a partition for each pair of $s^*$ and $t^*$. Ordering $O(n)$ partitions takes $O(n \log n)$ time. Thus, both algorithms take $O(n^2)$ time to find and order the partitions for the $O(n)$ pairs of $s^*$ and $t^*$. ST Procedure takes $O(n^2)$ time to test if a graph of $n$ vertices has
carvingwidth at least \( k \). Since a constant number of partitions are tested by ST Procedure, the total time for deciding whether a valid partition can be found is \( O(n^2) \). If a valid partition is not found, the edge contraction method is used to make a progress. This takes \( O(n^2) \) time [66]. Let \( T(n) \) be the time for computing an optimal carving decomposition of \( H \) with \( n \) vertices. Then

\[
T(n) = \max\{T(n_1) + T(n_2) + O(n^2), T(n-1) + O(n^2)\},
\]

where \( T(n_i) \) \((i = 1, 2)\) and \( T(n - 1) \) are the time for computing optimal carving decompositions of \( H'_i \) and \( H/e \), respectively. Since \( n_1 \leq n - 1, n_2 \leq n - 1, \) and \( n_1 + n_2 = n + 2 \), \( T(n) = O(n^3) \). It takes \( O(n) \) time to get a branch decomposition of \( G \) from the carving decomposition of \( H \) (Proposition 5.2.1).

The bound of Theorem 5.5.2 is the worst case time complexity of the divide-and-conquer algorithms. If a valid partition is always found and sizes of the two subgraphs differ only in a constant factor in every recursive step, then the divide-and-conquer algorithms run in \( O(n^2 \log n) \) time which is faster than the \( O(n^3) \) time edge-contraction algorithm.

We call the length-priority and size-priority algorithms the 2-component method because, the input graph in each recursive step is partitioned into two subgraphs and ST Procedure is used to test the carvingwidth of each subgraph. The 2-component method can be generalized to the \( 2^i \)-component method \((i \geq 1)\). Given an input graph, we first choose one pair of \( s^* \) and \( t^* \) to partition the graph into two subgraphs. We call the subgraphs level-1 subgraphs. A subgraph is called a level-\((j + 1)\) subgraph if it is obtained from a partition of a level-\( j \) \((j \geq 1)\) subgraph. In the \( 2^i \)-component method, we compute the level-\( i \) subgraphs (there are \( 2^i \) such graphs) by a sequence of partitions of the input graph. During the sequence of partitions, only one pair of \( s^* \) and \( t^* \) is used for each subgraph. We only check the sizes of the cut sets but do not check the carvingwidth for the level-\( j \) subgraphs for \( j < i \). We use ST Procedure to check the carvingwidth for every level-\( i \) subgraph. If all level-\( i \) subgraphs have carvingwidth at most \( k \), then the method is recursively applied to each level-\( i \) subgraph. If one level-\( j \) \((1 < j \leq i)\) subgraph \( H' \) has carvingwidth greater than \( k \) then we test the level-\((j - 1)\) subgraph from which \( H' \) is obtained. If all level-\((j - 1)\) subgraphs have carvingwidth at most \( k \) then the method is applied recursively (notice that a level-\((j - 1)\) subgraph \( H \) has carvingwidth at most \( k \) if all level-\( j \) subgraphs obtained from \( H \) have carvingwidth at most \( k \)). If a level-1 subgraph has carvingwidth greater than \( k \),
then we give up the current pair of \( s^* \) and \( t^* \) and apply the method to the input graph on a different pair of \( s^* \) and \( t^* \).

This generalization is motivated by the fact that testing the carvingwidth of large graphs by ST Procedure is the most time consuming part in finding the branch decompositions and some observations from the computational study: in most cases, a valid partition can be found in the first try and partitioning the input graph into smaller subgraphs can save the time used by ST Procedure. For constant \( i \), the \( 2^i \)-component algorithms have time complexity \( O(n^3) \).

The branch decomposition of a graph \( G \) which is not 2-connected can be easily constructed from the branch decompositions of its 2-connected components. So, the study of branch decomposition may be concentrated on 2-connected graphs.

### 5.5.2 Computational Results

We implemented our algorithms and the unaltered cycle method [69, 71]. A number of efficient implementations of ST Procedure are reported in [24]. The implementations of ST Procedure with the best practical performances are used in our algorithms and the cycle method. The implementation of the cycle method is a straightforward one: The pair of vertices \( s^* \) and \( t^* \) is selected in an arbitrary order. If there are multiple shortest paths \( P^* \)'s between \( s^* \) and \( t^* \) in \( M(G)^* \), an arbitrary one is used. Similarly, an arbitrary shortest path \( P^* \) is used for the length-priority and size-priority algorithms. We test our implementations on three classes of instances. Class (1) instances include Delaunay triangulations of point sets taken from TSPLIB [107]. The instances are provided by Hicks and are used as test instances in the previous studies [69, 71]. The instances in Class (2) are generated by the LEDA library [1, 90]. LEDA generates two types of planar graphs. One type of the graphs are the randomly generated maximal planar graphs and their subgraphs obtained from deleting some edges. Since the maximal planar graphs generated by LEDA always have branchwidth four, the subgraphs obtained by deleting edges from the maximal graphs have branchwidth at most four. The graphs of this type are not interesting for the study of branch decompositions. The other type of planar graphs are those generated based on some geometric properties, including Delaunay triangulations and triangulations of points uniformly distributed in a two-dimensional plane, and the intersection graphs of segments uniformly distributed in a two-dimensional plane. We report the results on the 2-connected intersection graphs. The instances in Class (3) are generated by the PIGALE library [4].
PIGALE randomly generates one of all possible planar graphs with a given number of edges based on the algorithms of [112]. We report the results on the 2-connected graphs generated by the PIGALE library.

We run the implementations on a computer with Intel(R) Xeon(TM) 3.06GHz CPU, 2GB physical memory and 4GB swap memory. The operating system is SUSE LINUX 10.0, and the programming language we used is C++.

**Results for Instances in Class (1)**

The computational results for Class (1) instances are reported in Table 5.7. In the table, $|E(G)|$ is the number of edges in the instance and thus the number of vertices in the medial graph $M(G)$ which is the input to the algorithms, $bw$ is the branchwidth of the graph $G$, $NT$ is the number of negative tests, $Cycle$ is the unaltered cycle method, $L.P$ is the length-priority algorithm, $S.P$ is the size-priority algorithm, and $S4$ is the 4-component algorithm with size-priority. For comparison, we include the running time of the $O(n^3)$ time edge-contraction method in column $EC_GT$ (the data is taken from [24] which uses a computer of similar performance to the one we use for the divide-and-conquer algorithms, and the $O(n^3)$ algorithm itself is given in [66]). In the table, an “X” indicates that it requires more than 70,000 seconds to solve the instance and a blank indicates that we did not test the algorithms for that instance.

The data show that all divide-and-conquer algorithms ($Cycle$, $L.P$, $S.P$, and $S4$) are much faster than the edge-contraction algorithm. The length-priority and size-priority algorithms are faster than the edge-contraction method by a factor of 200 $\sim$ 300 for instances of more than 5,000 edges in this class. It is difficult to compare the data of our algorithms with those of the cycle method reported in previous studies [69, 71], because computers of different speeds and different implementations of ST Procedure are used. To compare our algorithms with the cycle method on a same platform, we give a straightforward implementation of the unaltered cycle method using the same efficient ST Procedure used in our algorithms. Our algorithms are faster than the cycle method by a factor of at least 10 for instances of more than 5,000 edges. Notice that on average the cycle method is faster than the edge-contraction method by a factor of about 10 which is slightly smaller than that ($10 \sim 30$ on average) reported in previous studies [71]. Considering the fact that a more efficient edge-contraction algorithm is used in this study, our implementation of the cycle method has a similar performance as that used in the previous studies and our new
Table 5.7: Computation time (in seconds) of several decomposition algorithms for Class (1) instances. An \(X\) in the table indicates that it requires more than 70,000 seconds to solve the instance and a blank indicates that we did not test the algorithms for that instance.

| Graphs \(G\) | \(|E(G)|\) | \(bw\) | \(EC, GT\) time | Cycle time | \(LP\) time | \(SP\) time | \(S4\) time |
|-------------|-----------|--------|----------------|------------|-------------|-------------|-------------|
|             |           |        | time | NT    | time | NT    | time | NT    | time | NT    | time | NT    |
| pr1002      | 2972      | 21     | 2667 | 102   | 369  | 37    | 155  | 34    | 150  | 63    | 271  | 129   |
| rl1323      | 3950      | 22     | 6879 | 136   | 441  | 0     | 63   | 5     | 189  | 97    | 336  | 200   |
| d1655       | 4890      | 29     | 13529| 171   | 5958 | 806   | 295  | 34    | 218  | 28    | 402  | 59    |
| rl1889      | 5631      | 22     | 29096| 178   | 1896 | 527   | 130  | 0     | 115  | 1     | 90   | 2     |
| u2152       | 6312      | 31     | 26092| 192   | 2394 | 92    | 156  | 0     | 140  | 0     | 119  | 0     |
| pr2392      | 7125      | 29     | 45728| 271   | 5595 | 210   | 173  | 0     | 153  | 0     | 118  | 0     |
| pcb3038     | 9101      | 40     |      | 6265  | 53   | 490   | 8    | 998   | 17   | 1899  | 36   |
| fl3795      | 11326     | 25     | 8954 | 52    | 863  | 3     | 902  | 11    | 1190 | 22    |
| fnl4461     | 13359     | 48     |      | X     | X    | 3795  | 31   | 2479  | 16   | 2441  | 16   |
| rl5934      | 17770     | 41     |      | 2348  | 2    | 2585  | 6    | 3296  | 12   |
| pla7397     | 21865     | 33     |      | 10291 | 88   | 3026  | 10   | 3376  | 21   |
| usa13509    | 40503     | 63     |      | 25956 | 29   | 29539 | 79   | 50376 | 160  |
| brd14051    | 42128     | 68     |      | 10536 | 19   | 31554 | 129  | 64802 | 263  |
| d18512      | 55510     | 88     |      | 22378 | 44   | X     | X    | X     | X    |

algorithms are faster than the cycle method. For all instances which are solved within the 70,000 seconds time limit, the edge-contraction method is never used by any divide-and-conquer algorithm to make a progress, that is, a valid partition is always found in every recursive step.

There are two factors improving the running time of our algorithms. Both the length-priority and size-priority algorithms find more balanced partitions than the cycle method. This reduces the total running time in the divide-and-conquer approach. The other factor is that our algorithms have a smaller number of negative tests. In finding a valid partition, once a negative test happens, all divide-and-conquer algorithms try a different pair of \(s^*\) and \(t^*\) and the running time is increased. Also it takes more time for a negative test than a positive one. For Class (1) instances, the length-priority algorithm runs faster than the size-priority algorithm for large graphs while the size-priority algorithm does a better job for smaller graphs. Because the running time of the algorithms depends on both the size of the graphs and the number of negative tests, it may take a longer time to solve some instances than that for a larger graph. For example, Instance usa13509 requires a longer time than Instance brd14051 by the length-priority algorithm.
For Class (1) instances, the number of negative tests is non-trivial, especially for large graphs. This makes the $2^i$-component ($i > 1$) algorithms less efficient, because using more than two components generally increases the number of negative tests and thus the total running time. As shown in Table 5.7, the 4-component algorithm is slower than the 2-component algorithms for most instances in this class.

Results for Instances in Classes (2) and (3)

Computational results for Classes (2) and (3) instances are given in Tables 5.8 and 5.9, respectively. In the tables, S8 is the 8-component algorithm with the size-priority. An “X” in the tables indicates that it takes more than 150,000 seconds to solve that instance. Similar to results for Class (1) instances, the edge-contraction method is never used by any divide-and-conquer algorithm to make a progress for Classes (2) and (3) instances.

It takes more time to find the branch-decomposition of a Class (2) instance than a Class (1) instance with a similar size by divide-and-conquer algorithms. This may be caused by the fact that Class (2) instances have smaller branchwidth than that of Class (1) instances. A larger branchwidth implies that a longer cycle is used in a valid partition and a longer cycle usually gives a more balanced partition. For Class (2) instances, the size-priority algorithm runs faster than the length-priority algorithm and is faster than the edge-contraction algorithm by a factor of about 50 ~ 150. Both the length-priority and size-priority algorithms are faster than the cycle method. Since the number of negative tests in the divide-and-conquer algorithms for Class (2) instances is small, the $2^i$-component ($i > 1$) algorithms are more efficient than the 2-component ones. Especially, the 8-component algorithm is faster than the edge-contraction, the cycle, and the 2-component size-priority algorithms by factors of about 100 ~ 200, 5 ~ 8, and 2, respectively.

It takes more time to find the branch-decomposition of a Class (3) instance than a Class (1) or Class (2) instance with a similar size by the divide-and-conquer algorithms, because Class (3) instances have a smaller branchwidth. As shown in the table, the branchwidth of the instances is small constants and does not increase in the size of the instances. In each valid partition of the divide-and-conquer algorithms, we get a small subgraph of a constant size and a large subgraph for most instances. This limits the speed-up by the 2-component divide-and-conquer algorithms to a constant factor. Similar to the results for Class (2) instances, the number of negative tests in the divide-and-conquer algorithms
Table 5.8: Computation time (in seconds) of several decomposition algorithms for Class (2) instances. An X in the table indicates that it requires more than 150,000 seconds to solve the instance and a blank indicates that we did not test the algorithms for that instance.

| Graphs $G$ | $|E(G)|$ | $bw$ | $E_{\text{CGT}}$ | Cycle | $L_P$ | $S_P$ | $S_4$ | $S_8$ |
|------------|---------|------|------------------|--------|-------|-------|-------|-------|
| rand1160   | 2081    | 8    | 1749             | 34     | 53.2  | 0     | 29.9  | 23.1  | 18.4  |
| rand1672   | 3047    | 10   | 4695             | 103    | 137   | 2     | 54.6  | 39.7  | 29.7  | 25.7  |
| rand2780   | 5024    | 10   | 29073            | 147    | 2059  | 0     | 727   | 471   | 312   | 249   |
| rand3857   | 7032    | 11   | 82409            | 281    | 1503  | 0     | 810   | 493   | 351   | 292   |
| rand5446   | 10093   | 11   |                  |        | 11474 | 17    | 3283  | 2361  | 1532  | 1205  |
| rand8098   | 15031   | 13   |                  |        | 12022 | 76    | 2783  | 1864  | 1465  | 1159  |
| rand10701  | 20044   | 13   |                  |        | 11782 | 9     | 4368  | 3699  | 2884  | 2475  |
| rand15902  | 30010   | 14   |                  |        | 68809 | 125   | 32409 | 19127 | 13240 | 11744 |
| rand21178  | 40190   | 17   |                  |        | X     | X     | 93897 | 54557 | 33429 | 26910 |
| rand26304  | 50032   | 19   |                  |        |       |       |       | 149570| 85207 | 59907 | 47039 |
Table 5.9: Computation time (in seconds) of several decomposition algorithms for Class (3) instances. An $X$ in the table indicates that it requires more than 150,000 seconds to solve the instance.

| Graphs $G$ | $|E(G)|$ | $bw$ | $EC_GT$ | Cycle | $L_P$ | $S_P$ | $S4$ | $S8$ |
|------------|--------|------|---------|--------|-------|-------|------|------|
|            |        |      | time    | NT     | time  | NT     | time | NT   |
| PI855      | 1434   | 6    | 565     | 61     | 22.7  | 0      | 14.3 | 0    |
| PI1277     | 2128   | 9    | 1563    | 101    | 107   | 1      | 47.5 | 1    |
| PI1467     | 2511   | 6    | 3135    | 74     | 304   | 1      | 183  | 0    |
| PI2009     | 3369   | 7    | 8127    | 90     | 253   | 0      | 142  | 0    |
| PI2518     | 4266   | 8    | 17807   | 105    | 663   | 0      | 369  | 0    |
| PI2968     | 5031   | 6    | 26230   | 162    | 2244  | 9      | 1235 | 0    |
| PI3586     | 6080   | 8    | 49108   | 176    | 2340  | 1      | 1182 | 0    |
| PI4112     | 6922   | 7    | 70220   | 132    | 10808 | 1      | 10817| 0    |
| PI5940     | 10016  | 7    | $X$     | $X$    | 19770 | 0      | 18807| 0    |
| PI8950     | 15097  | 10   | $X$     | $X$    | 33862 | 13     | 19216| 1    |
| PI11974    | 20071  | 9    | $X$     | $X$    | $X$   | $X$    | 111747| 0    |

Note: An $X$ indicates that the solution requires more than 150,000 seconds.
is small and the $2^i$-component algorithms are faster than the 2-component ones. The 8-component algorithm is faster than the edge-contraction, the cycle, and the 2-component size-priority algorithms by factors of about $30 \sim 150$, $5 \sim 8$, and $2$, respectively.

5.6 Summary

We give efficient implementations of the Seymour and Thomas procedure which, given an integer $\beta$, decides whether a planar graph $G$ has the branchwidth at least $\beta$ or not. We tested our implementations on instances of size up to one hundred thousand edges. The results show that the branchwidth of those instances can be computed within a reasonable time and memory space. This suggests that the required memory may not be a bottleneck for computing branchwidth and optimal branch decompositions of planar graphs in practice. Our implementations without edge re-calculations require $O(n^2)$ bytes memory, although the constant behind the Big-Oh may be small. We have an upper bound $O(n^3)$ on the time complexity of the implementations with re-calculations for edge data. Let $p$ be the maximum number of edges kept in those implementations. This bound is true for any $p \geq 1$. In general, a larger $p$ results in a faster running time of the implementations. It is interesting to prove a better upper bound related to $p$, say $O(n^2(n/p))$, on the time complexity of those implementations.

We propose divide-and-conquer based algorithms of using ST procedure to compute optimal branch decompositions of planar graphs. Our algorithms have time complexity $O(n^3)$. Computational studies show that our algorithms are much faster than the edge-contraction algorithms and can compute the optimal branch decompositions for some instances of about 50,000 edges in a practical time. This provides useful tools for applying the branch decomposition based algorithms to practical problems.
Chapter 6

Edge Disjoint Paths Problem in Planar Graphs

Given a set $P$ of pre-routed paths in a graph $G$, the maximum edge-disjoint paths (MEDP) problem is to find a maximum subset $P' \subseteq P$ of paths such that no two paths in $P'$ share a common edge of $G$. As we already reviewed in Chapter 2, the MEDP problem has received much attention in the past decades. Most of the previous studies focus on developing performance guaranteed approximation algorithms. These algorithms are, although very important theoretically, often far from optimal. Their practical performances are not known, since there are very little efforts on the implementation of these algorithms. The only implementation we are aware of is given in [50] which implemented the $(3/3+\epsilon)$-approximation algorithm for the MEDP problem in directed trees, where $\epsilon > 0$ is any fixed constant [49].

In this chapter, we study the maximum edge-disjoint paths problem. We are mainly interested in exact algorithms for the problem. We show in Section 6.1 that the maximum edge-disjoint paths problem in planar graphs can be solved optimally, if the carvingwidth of the planar graph is bounded by a small constant. The running time is exponential in the carvingwidth but is polynomial in the number of nodes and edges of the graph, and polynomial in the number of given paths. Our algorithm has two steps: (I) computing an optimal carving decomposition of the planar graph, and (II) computing a maximum set of edge-disjoint paths, using a dynamic programming method based on the carving decomposition computed in Step I. Our algorithm works also for graphs that are close to planar graphs, by first computing a carving decomposition of the planar subgraphs. In
Section 6.2, we show the practical performance of our algorithm. We implement the optimal algorithm and test the implementation on both practical and random generated networks. Our experimental results show that the maximum edge-disjoint paths problem can be solved exactly for graphs with small carvingwidth in a practical time and memory space, when the load of the given set of paths is not too large. We also give an approximation algorithm for the maximum path coloring problem, for which an exact algorithm may not be practical.

### 6.1 Optimal Algorithm for the MEDP Problem

In this section, we give an algorithm which solves the maximum edge-disjoint paths problem optimally in planar graphs. The input to an instance of the maximum edge-disjoint paths problem consists of two parts: a set \( P \) of paths, and a graph \( G \). In Step I of our algorithm, we compute an optimal carving decomposition of the given planar graph, using the divide-and-conquer algorithm given in Section 5.5. For a graph \( G \) that is not planar but close to planar, one may compute an optimal carving decomposition \( T_C \) for a planar subgraph \( G' \) of \( G \), and then use \( T_C \) as the carving decomposition of \( G \). (For small graphs, the planar subgraphs may be found by hand. However, for large graphs, one may need to rely on some heuristics to find planar subgraphs.) Recall that the width of a link \( e \) in \( T_C \) is the number of edges between the two subgraphs obtained by removing \( e \) in \( T_C \). Since \( G \) has more edges than \( G' \), \( T_C \) may not be an optimal carving decomposition for the graph \( G \). However, \( T_C \) is very close to optimal if \( G \) is close to planar. Note that the dynamic programming part of our algorithm does not require the carving decomposition to be optimal, but the running time is exponential in the width of the decomposition. A carving decomposition which is optimal or close to optimal helps to reduce the total running time.

In order to describe Step II of our algorithm, we need some more definitions. Recall that for each link \( e \) of a carving decomposition \( T_C \), removing \( e \) separates \( T_C \) into two subtrees. Let \( V' \) and \( V'' \) be the sets of leaves of the subtrees. Let \( C_e \) be the set of edges of \( G \) incident to both a node of \( V' \) and a node of \( V'' \). Then \( |C_e| \) is bounded by the width of \( T_C \). We call a link \( e = \{x, y\} \) of \( T_C \) a leaf link if one of \( x \) and \( y \) is a leaf node of \( T_C \), otherwise an internal link.

In Step II, we compute a maximum set of edge-disjoint paths as follows. We first convert the carving decomposition \( T_C \) of \( G \) into a rooted binary tree by replacing a link \( \{x, y\} \) of \( T_C \) by three links \( \{x, z\}, \{z, y\}, \) and \( \{z, r\} \), where \( z \) and \( r \) are new nodes added to \( T_C \), \( r \) is
the root, and \( \{z, r\} \) is an internal link. We call \( \{z, r\} \) the root link. For every internal link \( e \) of \( T_C \), \( e \) has two child links incident to \( e \). For every link \( e \) of \( T_C \), let \( T_e \) be the subtree of \( T_C \) consisting of all descendant links of \( e \). Let \( G_e \) be the subgraph of \( G \) induced by the nodes at leaf nodes of \( T_e \). For an internal link \( e \) of \( T_C \), we use \( \mathcal{P}_e \) to denote the set of all subsets of edge-disjoint paths in \( P \) on \( C_e \). For a set of edge-disjoint paths \( P'_e \subseteq \mathcal{P}_e \), we define \( f(e, P'_e) \) as \( |Q_e| \), where \( Q_e \) is a maximum subset of paths in \( G_e \) such that \( P'_e \cup Q_e \) is edge-disjoint.

To compute a maximum edge-disjoint paths in \( G \), we find all sets of edge-disjoint paths (solutions, i.e., values \( f(e, P'_e) \)) of \( G_e \) from which a maximum edge-disjoint paths may be constructed for every link \( e \) of \( T_C \) by a dynamic programming method: the solutions of \( G_e \) for each leaf link \( e \) is empty and the solutions for an internal link \( e \) is computed by merging the solutions for the child links of \( e \).

Initially, \( f(e, P'_e) \) is set to 0 for all links \( e \) of \( T_C \) and for every possible subset \( P'_e \subseteq \mathcal{P}_e \). For a leaf link \( e \), no computation is needed. An internal link \( e \) has child edges \( e_1 \) and \( e_2 \) in \( T_C \). The values \( f(e, P'_e) \) is computed from \( f(e_1, P'_{e_1}) \) and \( f(e_2, P'_{e_2}) \). More specifically, we enumerate all possible subsets \( P'' \) of paths such that \( P'' \) is edge-disjoint and each path of \( P'' \) is on some edges in \( C_{e_1} \cup C_{e_2} \). Let \( P'_e \subseteq P'', P'_{e_1} \subseteq P'', \) and \( P'_{e_2} \subseteq P'' \) be the sets of paths on some edges in \( C_e \), \( C_{e_1} \), \( C_{e_2} \), respectively. Let \( P'_{e_1e_2} = (P'_{e_1} \cup P'_{e_2}) \setminus P''. \) For every \( P'' \), the values \( f(e_1, P''_{e_1}) \), \( f(e_2, P''_{e_2}) \), and \( |P''_{e_1e_2}| \) are added up. If this value is greater than the previous value for \( f(e, P''_e) \), then \( f(e, P''_e) \) is updated. At the root link \( e = \{z, r\} \), the maximum value \( f(e, P'_e) \) over all \( P'_e \) is the solution for the maximum edge-disjoint paths problem.

The running time of the algorithm can be estimated as follows. Step I of our algorithm runs in \( O(n^3) \) time (see Section 5.5). Step II is the dominant part for the time complexity. For each internal link \( e \) of \( T_C \), \( C_e \) is bounded by \( cw(G) \), the carvingwidth of \( G \). There are at most \((L + 1)^{cw(G)} \) possible subsets \( Q_e \) of partial solutions to store, since each edge of \( C_e \) has load bounded by \( L \), and each subset \( Q_e \) contains at most one path on each edge of \( C_e \). During the merging process, there are at most \((L + 1)^{1.5 \cdot cw(G)} \) possible cases to consider: we only need to consider paths on \( C_e \) and paths on \( C_{e_1} \cap C_{e_2} \), while \(|C_e \cup (C_{e_1} \cap C_{e_2})| \leq 1.5 \cdot cw(G)\). Thus, the time complexity for processing one link of \( T_C \) is \( O((L + 1)^{1.5 \cdot cw(G)}) \) and the memory requirement is \( O((L + 1)^{cw(G)} \cdot |V(G)|) \). The total time complexity for processing all links of \( T_C \) is \( O((L + 1)^{1.5 \cdot cw(G)} \cdot |V(G)|) \) and the memory requirement is \( O((L + 1)^{cw(G)} \cdot |V(G)|) \). When \( cw(G) \) is bounded by a constant, the running time and memory requirement are both polynomial in the input parameters.
CHAPTER 6. EDGE DISJOINT PATHS PROBLEM IN PLANAR GRAPHS

Notice that similar approach has been used in [46] to solve the call control problem in bounded degree trees. The highly simplified structure of bounded degree trees makes the problem easier.

The maximum path coloring (Max-PC) problem can be solved in a similar way. Recall that for the Max-PC problem, we are given a set $P$ of paths in a graph $G$, and we want to color a maximum subset of paths in $P$ using a given number $w$ of colors, such that no two overlapping paths are given the same color. The time complexity would be $O((L + 1)^{1.5\cdot cw(G)} \cdot |V(G)|)$ and the memory requirement would be $O((L + 1)^{w\cdot cw(G)} \cdot |V(G)|)$. Thus, the maximum path coloring problem can be solved optimally in polynomial time, if the number of colors and the carvingwidth are both bounded by a small constant. The running time and memory space are huge even for very small $L$, $cw(G)$, and $w$. Thus, this approach is not practical. One can have a 1.58-approximation iterative greedy algorithm for the Max-PC problem as follows: call the MEDP algorithm to select a set of edge-disjoint paths and color the selected set using one color; remove the colored paths and repeat the procedure on the remaining paths until the colors are used up or there is no remaining path. Obviously, the time complexity of this approach is similar to that of the MEDP algorithm, subject to a polynomial factor of $w$. It was proved in [120] that this iterative Max-PC algorithm has an approximation ratio of 1.58 if the MEDP algorithm is optimal. If we call the MEDP procedure until all paths are colored, we have an approximation algorithm for the minimum path coloring (Min-PC) problem.

6.2 Computational Results

We implemented our algorithms and tested our implementations on three classes of instances. Class (1) instances are real networks deployed in the US and in Europe. They include a 16-node NSFNET backbone (Figure 6.1), a 20-node European Optical Network (Figure 6.2), a 24-node ARPANET-like network (Figure 6.3), and a 30-node UK Network (Figure 6.4). Note that the 16-node NSFNET backbone, the 24-node ARPANET-like network, and the 30-node UK Network are not planar but very close to planar. In particular, removing the dark edges in Figures 6.1, 6.3, and 6.4 makes these graphs planar. For these non-planar graphs, an optimal carving decomposition $T_C$ is first computed for a planar subgraph (by removing the dark edges in the figure), using the divide-and-conquer algorithm given in Section 5.5. $T_C$ is then used as the carving decomposition of the original
6. EDGE DISJOINT PATHS PROBLEM IN PLANAR GRAPHS

(non-planar) graph $G$. $T_C$ may not be an optimal carving decomposition for the original graph $G$ because the dark edges are included. The instances in Class (2) are the intersection graphs of segments uniformly distributed in a two-dimensional plane, generated by the LEDA library [1, 90]. The instances in Class (3) are generated by the PIGALE library [4]. PIGALE randomly generates one of all possible planar graphs with a given number of edges based on the algorithms of [112]. We report the results on the 2-connected graphs generated by the PIGALE library. The intersection graphs generated by LEDA and random graphs generated by PIGALE have been used in the branch decomposition studies in Chapter 5.

We generate sets of paths as follows, given a positive integer $k$ and an allowable maximum load $L$ of the $k$ paths. We first generate $k$ source-destination pairs randomly in the given graph, and then connect them using shortest paths. We do not use any pair for which the shortest path has length one. If there are multiple shortest paths between the two end-nodes of a pair, an arbitrary one is used. If the generated set of paths has load more than $L$, then we discard the generated paths and start over again. Note that for given integers $k$ and $L$, it may not be possible to generate a set of $k$ paths with maximum load $L$, if $L$ is small and $k$ is large.

To compute an optimal carving decomposition $T_C$ of a planar graph, we use the length-priority algorithm given in Section 5.5. In Step II, to save memory, we compute the partial solutions for each link $e$ of $T_C$ in the postorder. Once the partial solutions are computed for a link $e$, the solutions for the child links of $e$ are discarded. The memory requirement may be reduced to $O((L + 1)^{cw(G)})$, if the binary tree obtained from $T_C$ is well balanced. Notice that Steps I and II have time complexities $O(|V(G)|^3)$ and $O((L+1)^{1.5\cdot cw(G)}\cdot |V(G)|)$, respectively.

We run the implementations on a computer with Intel(R) Xeon(TM) 3.06GHz CPU, 2GB physical memory and 4GB swap memory. The operating system is SUSE LINUX 10.0, and the programming language we used is C++.

6.2.1 Results for Instances in Class (1)

The computational results for Class (1) instances are reported in Tables 6.1 - 6.4. In the tables, $cw$ is the width of the carving decomposition for the graph $G$, $n$ and $m$ are the number of nodes and edges in the instances, respectively. The carving decompositions of the input graphs are computed using the length-priority algorithm given in Section 5.5. The time for computing the carving decompositions is small, thus we do not report it here. $|P|$ is
the number of given paths, $L(P)$ is the maximum load of the paths, $L(P)_\text{ave}$ is the average load of the edges, $\omega(P)$ is the clique size of the conflict graph of the paths. Notice that $L(P)$ is a lower bound on $\omega(P)$. We compute $\omega(P)$ using the well known $dfmax$ program [13]. $OPT_{MEDP}$ is the number of paths in an optimal solution for the maximum edge-disjoint paths problem, $SOL_{PC}$ is the number of colors used by the iterative greedy path coloring algorithm, $T_{MEDP}$ is the time used by the MEDP algorithm (in seconds), $T_{PC}$ is the time used by the iterative greedy path coloring algorithm (in seconds), and $Mem$ is the memory used by the algorithms (in megabytes, or MB).

From the tables, we can see that when the width of the carving decomposition is small (for example, 5 for the 16-node NSFNET backbone), our algorithms can handle a set of paths with load as large as 40. However, when the width of the carving decomposition is large (for example, 10 for the ARPANET-like network), the algorithms can only handle a set of paths with load at most 5. The number of paths cannot be too large. Otherwise the load would be too large since the graphs are small. The maximum set of edge-disjoint paths can usually be computed within several minutes for the tested instances. For larger instances, the program runs out of memory very quickly. The clique sizes of the conflict graphs are usually close to the load of the paths (in many cases, the clique size is equal to the load). This might be due to the simple structure of the networks, and the use of shortest paths routing. The number of colors used by the iterative path coloring algorithm is at most 1.67 times the clique size.

6.2.2 Results for Instances in Classes (2) and (3)

Computational results for Classes (2) and (3) instances are given in Table 6.5 and Table 6.6. The columns are named in the same way as in Class (1) instances. Our algorithms can handle large graphs in these two classes and large number of paths, providing the load is not too large. The maximum set of edge-disjoint paths can usually be computed within several hours for the tested instances. Again, for larger instances, the program runs out of memory very quickly. For some instances, the clique sizes of the conflict graphs can be twice the load of the paths (but within an additive constant of 10). The number of colors used by the iterative greedy path coloring algorithm is at most 2.2 times the clique size.

For a graph with large carvingwidth, Step II is both time and memory consuming, because this step runs exponentially in the carvingwidth. The time and memory increase very quickly for large load and carvingwidth. The memory requirement seems to be the main
hurdle for solving instances with large carvingwidth and large load. The computational results suggest that it may not be practical to solve the MEDP problem for instances in which $(L + 1)^{cw} > 10^7$ on a PC with 2GB memory.

6.3 Summary

We have given an optimal algorithm for the maximum edge-disjoint paths problem in planar graphs, and an approximation algorithm for the maximum path coloring problem. Our algorithms use dynamic programming method based on carving decompositions of the input graphs. We also tested the practical performances of the algorithms on both real and randomly generated planar graphs (or graphs close to planar). The computational results coincide with the theoretical analysis of the algorithms: they are efficient for graphs with small carvingwidth when the load is not too large, but may not be practical for graphs with large carvingwidth and for large load.
Figure 6.1: A 16-node NSFNET backbone.

Table 6.1: Computation results for a 16-node NSFNET backbone ($n = 16$, $m = 25$, $cw = 5$).

| $|P|$ | $L(P)_{ave}$ | $L(P)$ | $\omega(P)$ | $OPT_{MEDP}$ | $SOL_{PC}$ | $T_{MEDP}$ | $T_{PC}$ | Mem (MB) |
|-----|-------------|--------|-------------|---------------|-------------|------------|----------|---------|
| 100 | 10          | 15     | 18          | 10            | 19          | 1.25       | 3.97     | 17      |
|     | 10          | 20     | 20          | 11            | 22          | 2.94       | 9.06     | 60      |
| 150 | 14          | 20     | 22          | 11            | 25          | 13.4       | 37.6     | 112     |
|     | 16          | 25     | 25          | 11            | 28          | 18         | 57.2     | 325     |
|     | 15          | 30     | 30          | 10            | 32          | 20         | 70.8     | 659     |
|     | 16          | 35     | 35          | 10            | 35          | 22         | 99.8     | 1192    |
| 200 | 19          | 26     | 28          | 11            | 31          | 77         | 249      | 394     |
|     | 20          | 30     | 30          | 11            | 35          | 74         | 286      | 780     |
|     | 20          | 35     | 36          | 11            | 37          | 91         | 374      | 1634    |
|     | 20          | 40     | 40          | 11            | 40          | 85         | 346      | 1600    |
| 250 | 25          | 35     | 35          | 10            | 43          | 246        | 1145     | 1672    |
Figure 6.2: A 20-node European Optical Network.

Figure 6.3: A 24-node ARPANET-like network.
Table 6.2: Computation results for a 20-node European Optical Network \((n = 20, m = 38, cw = 8)\).

<table>
<thead>
<tr>
<th>(P)</th>
<th>(L(P)_{ave})</th>
<th>(L(P))</th>
<th>(\omega(P))</th>
<th>(OPT_{MEDP})</th>
<th>(SOL_{PC})</th>
<th>(T_{MEDP})</th>
<th>(T_{PC})</th>
<th>Mem (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>2.5</td>
<td>4</td>
<td>5</td>
<td>13</td>
<td>6</td>
<td>0.36</td>
<td>0.41</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2.7</td>
<td>6</td>
<td>6</td>
<td>11</td>
<td>8</td>
<td>1.47</td>
<td>2.75</td>
<td>244</td>
</tr>
<tr>
<td></td>
<td>2.8</td>
<td>8</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>1.63</td>
<td>2.41</td>
<td>297</td>
</tr>
<tr>
<td>60</td>
<td>3.7</td>
<td>6</td>
<td>7</td>
<td>16</td>
<td>7</td>
<td>5.66</td>
<td>6</td>
<td>245</td>
</tr>
<tr>
<td></td>
<td>3.9</td>
<td>8</td>
<td>8</td>
<td>13</td>
<td>9</td>
<td>10.6</td>
<td>18.8</td>
<td>1791</td>
</tr>
<tr>
<td>80</td>
<td>4.9</td>
<td>6</td>
<td>7</td>
<td>15</td>
<td>9</td>
<td>104</td>
<td>121</td>
<td>255</td>
</tr>
<tr>
<td></td>
<td>4.9</td>
<td>7</td>
<td>7</td>
<td>16</td>
<td>9</td>
<td>196</td>
<td>233</td>
<td>713</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>11</td>
<td>58</td>
<td>75.6</td>
<td>1796</td>
</tr>
<tr>
<td>100</td>
<td>6.1</td>
<td>8</td>
<td>9</td>
<td>17</td>
<td>12</td>
<td>421</td>
<td>484</td>
<td>1819</td>
</tr>
</tbody>
</table>
CHAPTER 6. EDGE DISJOINT PATHS PROBLEM IN PLANAR GRAPHS

Table 6.3: Computation results for a 24-node ARPANET-like network \((n = 24, m = 49, cw = 10)\).

| \(|P|\) | \(L(P)_{ave}\) | \(L(P)\) | \(\omega(P)\) | \(OPT_{MEDP}\) | \(SOL_{PC}\) | \(T_{MEDP}\) | \(T_{PC}\) | Mem (MB) |
|---|---|---|---|---|---|---|---|---|
| 30 | 1.5 | 2 | 2 | 15 | 3 | 0.1 | 0.1 | 3 |
| | 1.7 | 3 | 3 | 12 | 4 | 0.14 | 0.15 | 17 |
| | 1.7 | 4 | 4 | 12 | 5 | 0.85 | 0.99 | 26 |
| | 1.8 | 5 | 5 | 11 | 6 | 4 | 5.03 | 944 |
| 40 | 2 | 3 | 3 | 17 | 5 | 1.18 | 1.33 | 17 |
| | 2.2 | 4 | 4 | 13 | 6 | 1.35 | 1.43 | 158 |
| | 2.4 | 5 | 6 | 11 | 7 | 4.2 | 5.16 | 944 |
| 50 | 2.7 | 4 | 4 | 17 | 6 | 4.38 | 4.83 | 159 |
| | 2.8 | 5 | 6 | 15 | 6 | 5.61 | 6.39 | 944 |
| 60 | 3.1 | 5 | 6 | 18 | 7 | 13.7 | 15.1 | 945 |

Table 6.4: Computation results for a 30-node UK Network \((n = 30, m = 57, cw = 8)\).

| \(|P|\) | \(L(P)_{ave}\) | \(L(P)\) | \(\omega(P)\) | \(OPT_{MEDP}\) | \(SOL_{PC}\) | \(T_{MEDP}\) | \(T_{PC}\) | Mem (MB) |
|---|---|---|---|---|---|---|---|---|
| 60 | 3.1 | 4 | 5 | 20 | 6 | 1.7 | 2.09 | 7 |
| | 3.4 | 8 | 8 | 16 | 9 | 3.1 | 6 | 121 |
| | 3.6 | 10 | 10 | 15 | 10 | 6.5 | 9.32 | 577 |
| 80 | 4.2 | 6 | 7 | 20 | 9 | 4.3 | 5.96 | 82 |
| | 4.4 | 8 | 8 | 20 | 9 | 12.2 | 17.4 | 591 |
| | 4.7 | 10 | 10 | 16 | 13 | 23.2 | 39 | 1337 |
| 100 | 4.7 | 6 | 6 | 23 | 8 | 25.2 | 39.7 | 82 |
| | 5.2 | 8 | 9 | 23 | 11 | 71.1 | 95.6 | 591 |
| 120 | 5.7 | 7 | 8 | 23 | 10 | 90 | 123 | 233 |
| | 6.1 | 8 | 9 | 22 | 11 | 524 | 673 | 591 |
| 140 | 6.7 | 8 | 8 | 22 | 11 | 270 | 392 | 591 |
Table 6.5: Computation results for instances generated by LEDA.

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
<th>cw</th>
<th></th>
<th>P</th>
<th></th>
<th>L(P)_{ave}</th>
<th>L(P)</th>
<th>\omega(P)</th>
<th>OPT_{MEDP}</th>
<th>SOL_{PC}</th>
<th>T_{MEDP}</th>
<th>T_{PC}</th>
<th>Mem (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>177</td>
<td>7</td>
<td>200</td>
<td>4.3</td>
<td>5</td>
<td>6</td>
<td>55</td>
<td>9</td>
<td>20.6</td>
<td>25.8</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.7</td>
<td>6</td>
<td>7</td>
<td>52</td>
<td>10</td>
<td>33.7</td>
<td>50</td>
<td>55</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>48</td>
<td>12</td>
<td>106</td>
<td>154</td>
<td>138</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5.4</td>
<td>8</td>
<td>9</td>
<td>47</td>
<td>12</td>
<td>87.5</td>
<td>132</td>
<td>300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5.4</td>
<td>9</td>
<td>11</td>
<td>44</td>
<td>14</td>
<td>159</td>
<td>273</td>
<td>607</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5.7</td>
<td>10</td>
<td>12</td>
<td>40</td>
<td>15</td>
<td>186</td>
<td>343</td>
<td>1125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>214</td>
<td>365</td>
<td>8</td>
<td>300</td>
<td>3.2</td>
<td>4</td>
<td>5</td>
<td>101</td>
<td>8</td>
<td>51.6</td>
<td>57.8</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.6</td>
<td>5</td>
<td>7</td>
<td>88</td>
<td>11</td>
<td>186</td>
<td>209</td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.1</td>
<td>6</td>
<td>9</td>
<td>78</td>
<td>12</td>
<td>297</td>
<td>412</td>
<td>402</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.3</td>
<td>7</td>
<td>9</td>
<td>80</td>
<td>11</td>
<td>828</td>
<td>1132</td>
<td>1149</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>531</td>
<td>922</td>
<td>10</td>
<td>400</td>
<td>1.7</td>
<td>2</td>
<td>3</td>
<td>218</td>
<td>5</td>
<td>9.9</td>
<td>10.4</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.1</td>
<td>3</td>
<td>4</td>
<td>173</td>
<td>7</td>
<td>89.4</td>
<td>100</td>
<td>79</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.4</td>
<td>4</td>
<td>5</td>
<td>154</td>
<td>8</td>
<td>171</td>
<td>205</td>
<td>578</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>815</td>
<td>1431</td>
<td>10</td>
<td>600</td>
<td>1.7</td>
<td>2</td>
<td>3</td>
<td>332</td>
<td>5</td>
<td>42.7</td>
<td>44.6</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.1</td>
<td>3</td>
<td>4</td>
<td>268</td>
<td>7</td>
<td>121</td>
<td>141</td>
<td>137</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.4</td>
<td>4</td>
<td>5</td>
<td>220</td>
<td>10</td>
<td>1904</td>
<td>2087</td>
<td>1187</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 6.6: Computation results for instances generated by PIGALE.

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
<th>cw</th>
<th>134</th>
<th>1893</th>
<th>1853</th>
<th>2000</th>
<th>2761</th>
<th>4610</th>
<th>5571</th>
<th>6016</th>
<th>6969</th>
<th>9000</th>
</tr>
</thead>
<tbody>
<tr>
<td>154</td>
<td>3000</td>
<td>5</td>
<td>13</td>
<td>16</td>
<td>23</td>
<td>32</td>
<td>45</td>
<td>61</td>
<td>84</td>
<td>104</td>
<td>127</td>
<td>150</td>
</tr>
<tr>
<td>5000</td>
<td>67</td>
<td>23</td>
<td>35</td>
<td>39</td>
<td>56</td>
<td>71</td>
<td>100</td>
<td>139</td>
<td>194</td>
<td>239</td>
<td>276</td>
<td>334</td>
</tr>
<tr>
<td>10000</td>
<td>112</td>
<td>67</td>
<td>100</td>
<td>94</td>
<td>127</td>
<td>175</td>
<td>233</td>
<td>315</td>
<td>415</td>
<td>515</td>
<td>615</td>
<td>730</td>
</tr>
<tr>
<td>20000</td>
<td>233</td>
<td>23</td>
<td>43</td>
<td>36</td>
<td>49</td>
<td>62</td>
<td>84</td>
<td>113</td>
<td>147</td>
<td>187</td>
<td>227</td>
<td>276</td>
</tr>
<tr>
<td>30000</td>
<td>353</td>
<td>67</td>
<td>68</td>
<td>59</td>
<td>82</td>
<td>105</td>
<td>138</td>
<td>186</td>
<td>234</td>
<td>283</td>
<td>336</td>
<td>402</td>
</tr>
<tr>
<td>40000</td>
<td>473</td>
<td>127</td>
<td>94</td>
<td>79</td>
<td>104</td>
<td>135</td>
<td>174</td>
<td>226</td>
<td>283</td>
<td>336</td>
<td>402</td>
<td>473</td>
</tr>
</tbody>
</table>

- | | | | | | | | | | | | |

Note: L(P) - Lower Bound, OPTMEDP - Optimal MEDP, TMEDP - Total MEDP, Tpc - Time of Computation, P - Problem, w(P) - Weight of Path, OPTMEDP - Optimal MEDP, TMEDP - Total MEDP, Tpc - Time of Computation, P - Problem, w(P) - Weight of Path.
Chapter 7

Conclusion and Future Work

In this thesis, we have studied fundamental routing and channel assignment problems in WDM optical networks with specific topologies and general topologies. Our study on specific topologies includes the Min-PC problem on trees of rings, the Max-RPC problem on rings, the Min-PMC and Max-PMC problems on multifiber trees, and the call control problem on bounded depth trees. We developed a carving-decomposition based method to solve exactly the edge-disjoint paths problem, and to approximate the maximum path coloring problem for networks with more general topologies. The carving-decomposition based approach works for graphs with bounded carvingwidths. In this chapter, we summarize the contributions of this work, and give a few directions for future work.

7.1 Summary of Contributions

We have given an efficient algorithm which solves the Min-PC problem on a tree of rings with an arbitrary (node) degree using at most $3L$ colors and achieves an approximation ratio of 2.75 asymptotically. The $3L$ upper bound is tight even on a tree of rings with degree four. We also give a $3L$ and 2-approximation (resp. 2.5-approximation) algorithm for the Min-PC problem on a tree of rings with degree at most six (resp. eight and ten).

We have shown that the call control problem is NP-hard and MAX SNP-hard even in depth-2 trees. We give optimal algorithms for the call control problem in double-stars and in spiders. We also give 2- and 3-approximation algorithms for the weighted call control problem in depth-2 and depth-3 trees, respectively. We show that the weighted call control problem is solvable in arbitrary trees if all the paths contain a same node of the tree.
We have shown that the Min-PMC and Max-PMC problems are NP-hard in $k$-fiber ($k$ odd) stars. We give optimal algorithms for the following problems: the Min-PMC and Max-PMC problems in non-uniform stars with even fibers and in $k$-fiber ($k$ even) spiders. We have given a 1.5-approximation algorithm for the weighted Max-RPC problem in rings.

We give efficient implementations of the Seymour and Thomas procedure which, given an integer $\beta$, decides whether a planar graph $G$ has the branchwidth at least $\beta$ or not. We tested our implementations on instances of size up to one hundred thousand edges. The experimental results show that our implementations run much faster and use less memory than previous implementations, even considering the speed difference of the computers used (see page 127 for details).

We propose divide-and-conquer based algorithms of using Seymour and Thomas procedure to compute optimal branch decompositions of planar graphs. Our algorithms have time complexity $O(n^3)$. Computational studies show that our algorithms are much faster than the edge-contraction algorithms and can compute the optimal branch decompositions for some instances of about 50,000 edges in a practical time. This provides useful tools for applying the branch decomposition based algorithms to practical problems.

We have given an optimal algorithm for the maximum edge-disjoint paths problem in planar graphs, and an approximation algorithm for the maximum path coloring problem. We also tested the practical performances of the algorithms on both real and randomly generated planar graphs (or graphs close to planar). The computational results coincide with the theoretical analysis of the algorithms: they are efficient for graphs with small carvingwidth when the load is not too large, but may not be practical for graphs with large carvingwidth and for large load.

### 7.2 Future Work

Many research efforts have been devoted to the research problems we studied in this thesis. However, there are still many open problems. In this section, we give a few directions for the future work.

We have given a 2.75-approximation algorithm for the Min-PC problem on trees of rings with arbitrary degrees. An interesting problem is to improve the 2.75-approximation ratio. Our results imply a 3-approximation algorithm for the Min-RPC problem on a tree of rings. It would be challenging to improve the approximation ratio of 3 for the Min-RPC problem.
Our 3L algorithm also implies a 6L algorithm for the Min-PC problem on directed trees of rings. It would be interesting to improve the approximation ratio for the Min-PC problem on directed trees of rings.

For the optimal branch decomposition of planar graphs, the following problems are unsettled and may be studied in the future:

- Our implementations of ST procedure may require at least $n^2/8$ bytes of memory for a graph of $n$ edges. So they may not be able to solve extremely large instances with a few hundred thousands or more edges within a practical memory space. How to compute the branchwidth of extremely large planar graphs is an interesting open problem.

- Our divide-and-conquer based algorithms can compute an optimal branch decomposition of planar graphs of size up to 50,000 edges in a practical time on a PC with 3GHz CPU. It is still time consuming to compute optimal branch decompositions for planar graphs with more than 50,000 edges. An interesting future work is to design more efficient algorithms for very large planar graphs. Using better approaches to make balanced partitions is one possible direction to get such algorithms.

- All divide-and-conquer algorithms use the edge-contraction method to guarantee the branch decomposition can be found. However, the edge-contraction algorithm has never been called in our computational study. It would be interesting to prove that a valid partition can always be found efficiently in those algorithms.

- It would be interesting to develop a performance guaranteed and yet efficient heuristic for computing a good approximation of the carvingwidth, and for computing a carving decomposition that is close to optimal.

Our optimal algorithm for the maximum edge-disjoint paths problem and approximation algorithm for the maximum path coloring problem are not practical when the carvingwidth of the input graph is large, or when the load of the paths is large. Whether the performance of the algorithms can be improved is an interesting open problem. It is also worth to design heuristic algorithms which can compute a solution for larger instances within a practical time and memory, even if the solution may not be optimal.
Bibliography


