Lehrstuhl für Effiziente Algorithmen
der Technischen Universität München

Scheduling Connections in Fast Networks

Thomas Erlebach

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. Dr. h.c. Wilfried Brauer
Prüfer der Dissertation:		
	1.	Univ.-Prof. Dr. Ernst W. Mayr
	2.	Univ.-Prof. Dr. Peter Gritzmann
	3.	Univ.-Prof. Dr. Rolf H. Möhring, Technische Universität Berlin (schriftliche Beurteilung)

Die Dissertation wurde am 21.10.1998 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 06.05.1999 angenommen.

Abstract

Fast and reliable exchange of data in world-wide communication networks has acquired essential importance in our society today, and telecommunications is one of the most important growth sectors in all leading economies. New applications like supercomputer networking, multimedia networking, and real-time medical imaging require transfer rates in the range of several gigabits per second, and they become possible only through advances in the technology and architecture of communication networks: enormous amounts of data can be transmitted through optical fiber using laser beams today, and new network protocols, e.g., ATM, allow flexible bandwidth reservation and enable simultaneous transmission of data with various different characteristics (telephone calls, multimedia data, electronic messages), satisfying the different requirements of all these various traffic types at the same time.

The efficient use of modern communication networks is tied to a number of demanding algorithmic problems, in particular concerning the allocation of network resources to individual connections. More specifically, wavelength assignment in all-optical networks and call scheduling in communication networks with bandwidth reservation lead to a number of interesting combinatorial optimization problems. In this thesis, the complexity of such problems, i.e., the difficulty of algorithmic solutions, is determined under various restrictions, and polynomial-time approximation algorithms, i.e., algorithms computing solutions that are provably good (but not always optimal), are presented. The combinatorial optimization problems studied in this thesis pertain to simplified models of real-life communication networks. These combinatorial models capture the essential characteristics of the diverse problems encountered in practice.

In all-optical networks with wavelength-division multiplexing, several connections can use the same fiber link simultaneously if the signals are transmitted on different wavelengths. Connections must use the same wavelength on the whole transmitter-receiver path, if wavelength conversion is not available. The *path coloring* problem is to assign colors (wavelengths) and paths to a given set of connection requests such that paths using the same link are assigned different colors. The goal is to minimize the number of different colors used. This problem is shown to be $\mathcal{NP}$-hard for bidirected trees even in the binary case, for undirected trees of arbitrary degree and for bidirected and undirected rings. A polynomial-time optimal algorithm is given for undirected trees of constant degree. For undirected trees of arbitrary degree, an asymptotic 1.1-approximation is presented. A polynomial-time 5/3-approximation is given for bidirected trees of arbitrary degree.

If the number of available wavelengths is limited, which is the case in practice, the *maximum path coloring* (MaxPC) problem is of interest as well. Given a set P of connection requests and a number W of available wavelengths, the goal is to select a maximum cardinality subset P' of P and to compute an assignment of colors and paths to the connection requests in P' using at most W colors. MaxPC is studied for bidirected trees. Polynomial-time optimal algorithms are shown to exist if the tree has depth one or if both W and the degree of the tree are bounded by a constant. Furthermore, MaxPC is proved $\mathcal{NP}$-hard for bidirected trees of constant degree if W can be arbitrary, and for bidirected trees of arbitrary degree even if $W = 1$. In the case $W = 1$ the problem is equivalent to the maximum edge-disjoint paths problem. For every fixed $\varepsilon > 0$, a polynomial-time $(5/3 + \varepsilon)$-approximation algorithm for the maximum edge-disjoint paths problem (and, therefore, for MaxPC with $W = 1$) is obtained. Using a known reduction from MaxPC with arbitrary number of wavelengths to MaxPC with $W = 1$, a 2.22-approximation for MaxPC with arbitrary number of wavelengths is derived.

While path coloring problems have a mainly graph theoretic flavor, the problem of call scheduling in communication networks with bandwidth reservation is closely related to multiprocessor scheduling and bin packing. When a call is established, the required bandwidth is reserved on all links along a path from sender to receiver for the duration of the call. The communication network is represented by a graph with edge capacities. A call is specified by a pair of vertices, a bandwidth requirement, and a call duration. *Call scheduling* is the problem of assigning paths and starting times to calls in a communication network such that the sum of bandwidth requirements of simultaneously active calls using the same link does not exceed the capacity of that link. Upper and lower bounds on the approximation ratio achieved by variants of the List-Scheduling (LS) algorithm are obtained for call scheduling in star and tree networks. The variants for calls with arbitrary duration work also if the duration of a call is not known in advance; hence, these variants are batch-style on-line algorithms. For unit duration, the approximation ratio of LS in stars is shown to be at most 4.875 for arbitrary lists and at most $2\frac{2}{3}$ if the list of calls is sorted according to non-increasing bandwidth

requirements. For arbitrary, unknown duration, the competitive ratio of LS is proved to be at most 5. In tree networks with n nodes, a variant of LS for calls with unit duration has approximation ratio at most 6, and a variant for calls with arbitrary, unknown duration has competitive ratio at most $5 \log n$.

On the one hand, the obtained results show the difficulty of computing optimal solutions for a number of combinatorial optimization problems concerning the allocation of resources in communication networks. These results resolve substantial open questions regarding the boundary between tractable and intractable versions of the problems under consideration. On the other hand, several efficient approximation algorithms are presented and analyzed; they do not always compute an optimal solution, but a solution that can be worse than the optimal solution only by a small (often constant) factor. These algorithms and their analysis give important insight into the practical benefit of different network architectures and the network utilization achievable in the worst case. Furthermore, the algorithms can be used as a basis for the implementation of resource allocation methods that perform well in practice for optical networks and for networks with bandwidth reservation. Finally, the analysis of the studied list-scheduling variants provides a justification for the use of simple heuristics, which are often preferred in practice due to ease of implementation, under certain conditions.

Acknowledgments

This work would have been impossible without the help from a number of people. First of all, I am deeply indebted to my advisor Ernst Mayr for invaluable guidance and support throughout the past four years, for teaching me insight into a number of advanced concepts and ideas, for giving me the motivation and opportunity to investigate challenging problems and try harder to solve them, and for many things more. I would also like to thank all my colleagues and ex-colleagues at the Chair for Efficient Algorithms and at TUM in general for assistance with numerous issues and for making the working environment here very pleasant and enjoyable at all times.

Furthermore, I want to express my sincere gratitude to Klaus Jansen for the many stimulating ideas, helpful discussions, and fruitful collaborations. He deserves a substantial share of the credits for the results presented in this thesis; many of them were derived only after he initiated to study the problems, suggested promising ways to tackle them, or came up with the right ideas that led to a solution.

I am also grateful to numerous other colleagues in the field, who have been extremely helpful by readily providing preliminary versions of their own results when they were not yet published, by sharing ideas regarding possible improvements of my work, or by engaging in stimulating discussions. Among many others, I want to thank Anja Feldmann, Vijay Kumar, Christos Kaklamanis, Stefano Leonardi, Aris Pagourtzis, Pino Persiano, Adi Rosén, and Alexander Schrijver.

Besides, I would like to gratefully acknowledge the support from the German Research Foundation (DFG), who has been funding my position at TUM under grant SFB 342 TP A7.

Finally, I want to dedicate this thesis to my parents, who have always supported me in the best of all ways, and, above all, to my wife Lai-Chong, whose love brings sunshine and happiness into my life on every single day and without whom everything else would be worthless.

Contents

Chapter 1

Introduction

The demand for communication networks with ever-increasing capacity and service quality is the driving force behind ongoing fruitful research on physical network technology, flexible protocols, and efficient resource management. Over the past 30 years, users' telecommunication needs have exploded. In the early 1970s, the highest-speed private lines were 9.6 kbps (kilobits per second); today, private lines at 45 Mbps (megabits per second) are widely available. Within the public network in the USA, the 1.544 Mbps facilities of the 1970s have been replaced by 2.4 Gbps (gigabits per second) Synchronous Optical Network (SONET[1]) facilities after the advent of Broadband ISDN (Broadband Integrated Services Digital Network, B-ISDN) [RS92]. Now, applications that require transfer rates of several Gbps are emerging. Such potential gigabit applications include supercomputer networking, remote visualization, medical imaging, collaborative group work, HDTV (high-definition television), virtual reality, and telepresence. Fast and powerful networks are required on the LAN (local area network), MAN (metropolitan-area network), and WAN (wide area network) levels. The most promising technology for the implementation of gigabit networks are so-called *all-optical networks*, i.e., networks in which data is transmitted on lightwaves through optical fiber and no optical-to-electrical conversions are required at intermediate nodes on the way from sender to receiver.

Changing characteristics of applications have also led to a shift from packet switching to (virtual) circuit switching for the transmission of digital data. Traditionally, telephone networks were realized by circuit switching (a connection is established by reserving a channel on all links of a path between the communication endpoints), but digital data was transmitted using packet switching (a message is split into small packets that are forwarded

[1]The SONET optical transmission standard is referred to internationally as the Synchronous Digital Hierarchy (SDH).

from node to node until they reach the receiver). While packet switching is appropriate for file transfers or electronic messages, today more and more applications require connections with guaranteed available bandwidth and delay characteristics. For example, if moving image sequences (animations, movies) are transmitted in multimedia applications (e.g., multimedia teleconferencing), it is essential that all data arrive at the receiver in a timely fashion. Because of such applications, networks with *guaranteed quality of service* are becoming increasingly important. In such networks, a connection request can specify its required bandwidth and other service parameters, and the network guarantees that the requested bandwidth and service quality will be available to the connection as long as it remains active. This is achieved by allocating resources along a path from sender to receiver; it can be viewed as establishing a *virtual circuit*.

In all-optical networks and in networks with guaranteed quality of service, resource management is an important issue. Resources must be reserved for every established connection, and a bad resource management strategy may lead to reduced utilization and throughput of the network. In particular, call admission (deciding whether to accept or reject a connection request), call scheduling (establishing all connections in a set of connection requests and completing them in minimum time), and wavelength routing (assigning paths and wavelengths to connection requests in all-optical networks with wavelength-division multiplexing) are problems that must be addressed. We study several combinatorial optimization problems related to call admission, call scheduling, and wavelength routing. The focus of the work is on simplified combinatorial models that capture the essential characteristics of the diverse problems encountered in practice. The types of results that we will derive are complexity results (determining which problems are $\mathcal{NP}$-hard [GJ79] and which can be solved in polynomial time) and algorithms (polynomial-time optimal algorithms and, predominantly, approximation algorithms with provable performance guarantees).

1.1 Technological Background

In this section, we briefly explain certain aspects regarding the technology of all-optical networks and of networks with guaranteed quality of service. The material in this section is not a prerequisite for understanding the remaining chapters of this thesis; it should rather serve as motivation for studying the combinatorial optimization problems defined in Section 2.2.

1.1.1 All-Optical Networks

Three generations of physical-level technology for communication networks can be defined [Gre91]. For networks of the first generation, up to the early 1980s, fiber-optic technology was not available, and copper was the main medium for transmission of data. Networks of the second generation, still widely in use today, are mainly characterized by upgrading individual copper links to optical fiber. These networks take advantage of the higher bandwidth and smaller bit-error rate of optical fiber on individual links, but all switching in the network nodes is still done electronically. The topologies and protocols are the same as those used in the first generation. Finally, networks of the third generation, which have begun to emerge at the time of this writing, will use fiber for its unique properties. The bottleneck caused by electronic switching is overcome by doing all switching optically (photonic switching). The key advantage of optical switching is that it avoids multiple optical-to-electrical conversions and electronic switching operations at intermediate nodes along a connection.

A single fiber-optic cable offers bandwidth of about 25,000 to 30,000 GHz. It can potentially carry information at the rate of several terabits per second. No electronic device can process data at such speeds. In order to utilize the potential of optical fiber, wavelength-division multiplexing (WDM) is used [CNW90, CHK+96]. The bandwidth is partitioned into a number of channels at different wavelengths. A single channel supplies bandwidth in the range of gigabits per second, and several signals can be transmitted through a fiber link simultaneously on different channels. The typical number of channels (wavelengths) available in WDM systems today ranges from as few as two to as many as several hundred [VD93, Ram93], with numbers from 40 to 60 appearing practical. Tunable lasers or arrays of fixed-wavelength lasers are used to generate the laser beams that are to be transmitted on the optical channels [Bra90]. Add/drop multiplexers are employed at the network nodes to insert lightwaves into the fiber or to extract them. Fixed-wavelength or tunable filters and receivers are used at the receiving side of a transmission. The electronic equipment is not required to operate faster than a single optical channel; thus, WDM allows existing electronic equipment to fully use the enormous potential of optical fiber.

Several architectures for all-optical networks have been proposed. We assume an architecture with pairs of unidirectional fiber links or with single bidirectional links between adjacent nodes of the network and with *reconfigurable wavelength-selective* switches (also known as *generalized* switches) in the nodes. No distinction between access nodes and routing nodes is made. A reconfigurable wavelength-selective switch is capable of splitting the signals on incoming links according to their wavelengths and switching them onto arbitrary outgoing links. While setting up the configuration of such a switch is costly (it takes around 50ms for switches manufactured today), it can switch incoming signals onto outgoing links virtually without any delay. The routing of a signal depends only on its wavelength (wavelength routing); no inspection of packet headers etc. is required. The latency of an all-optical network is therefore limited only by the propagation delay of light in fiber. Light travels in fiber at a speed of about 2×10^8m/s; hence, a distance of, say, 5,000 meters cannot be traveled faster than 25 microseconds. All-optical networks with reconfigurable wavelength-selective switches have been proposed and studied, for example, in [CGK92, VD93, RS95].

Most optical switches that are built today do not offer wavelength conversion; a signal that enters the switch on one wavelength must leave the switch on the same wavelength. All-optical networks with reconfigurable wavelength-selective switches, but without wavelength conversion are called *wavelength-selective* (WS) networks. In WS networks, a connection must use the same wavelength on the whole path from transmitter to receiver. Due to interference, no two signals may be transmitted through a fiber link on the same wavelength. Hence, a wavelength must be reserved for a connection on all links on a path from transmitter to receiver; the path is then called a *lightpath*.

Switches with wavelength conversion capabilities are a field of ongoing research. At the time of this writing, wavelength conversion technology is still considered relatively immature and expensive [BDO+98]. However, it is conceivable that technological progress will make these devices practical in the future. A switch with full wavelength conversion capabilities can switch any incoming signal onto any outgoing link and at the same time change its wavelength into any other wavelength. All-optical networks with full wavelength conversion switches are called *wavelength-interchanging* (WI) networks. In a WI network, a connection can use different wavelengths on different links of its transmitter-receiver path.

The complexity of building switches with full wavelength conversion capabilities is high; therefore, switches with limited wavelength conversion are also considered. In particular, switches that can convert every wavelength only to a subset of the other wavelengths or even only to adjacent wavelengths may be more practical than switches with full wavelength conversion capabilities.

In all-optical WDM networks, the number of channels (wavelengths) is a scarce resource. The cost and complexity of the switches and add/drop multiplexers grow substantially with increasing number of wavelengths.

A set of connections in a WS network can be established if each connection is assigned a transmitter-receiver path and one of the available wavelengths such that connections sharing a link receive different wavelengths. This gives rise to two kinds of optimization problems. First, it is desirable to establish a given set of connection requests with a minimum number of wavelengths. This problem is relevant, for example, when the provider designs the network and decides which add/drop multiplexing devices should be employed. We view wavelengths as colors and model this routing and wavelength assignment problem as the *path coloring* problem (all optimization problems mentioned in this section will be defined formally in Section 2.2). The second problem arises when the capacity of an existing network is not sufficient for establishing all requests in a given set of connection requests simultaneously. If the network supports a certain number of wavelengths, the goal is to establish as many of the connection requests as possible simultaneously, while rejecting or deferring the remaining requests. This problem is modelled by the *MaxPC* problem. Solutions to these problems of efficient wavelength allocation have substantial importance for the future development of WDM technology.

In a WI network, a set of connections can be established if they are assigned transmitter-receiver paths such that no link is used by more connections than the number of available wavelengths. For WI networks, we encounter the same two kinds of optimization problems as for WS networks; here, we refer to them as *path packing* and *MaxPP*, respectively.

At the WAN level, all-optical networks would typically be employed by big telecommunications providers. The network connects nodes located in different parts of a country or in different countries. The connection requests correspond to connectivity requirements predicted by the provider from information collected about telephone calls and data transmissions over a period of time. The availability of optical amplifiers (using Erbium-doped fiber) ensures that large distances do not cause a problem for all-optical networks.

The application of all-optical networks for distributed computing seems very promising as well [VD93]. Lightpaths can be used to establish a virtual topology on top of the underlying physical topology of the network. For example, a regular topology like hypercube or grid may be imposed on nodes interconnected by an irregular all-optical network; it suffices to establish a lightpath for each connection of the virtual topology. Different virtual topologies can be achieved by reconfiguring the switches of the network. Once a certain virtual topology is established, the communication between the nodes can take place as if the links of the virtual topology were physically present; no delay beyond the propagation delay of light in fiber is introduced by the fact that a link of the virtual topology is realized by a lightpath in the physical topology.

A different architecture for all-optical WDM networks is based on the broadcast-and-select paradigm. For example, the passive optical star topology has received a lot of attention. A number of nodes are connected to a central hub. All the transmissions from the various nodes are combined in the central hub (a WDM passive star coupler), and the mixed optical information is broadcast to all nodes. Besides the star topology, linear bus and tree structures have been proposed. Broadcast-and-select networks can be classified into single-hop systems and multihop systems. The former are surveyed in [Muk92a], the latter in [Muk92b]. Unlike the all-optical networks we study, broadcast-and-select networks do not allow spatial reuse of wavelengths. [Ram93] explains the technology and compares the respective advantages and disadvantages of broadcast-and-select networks and wavelength-routing networks.

While WDM appears the most promising alternative for employing all-optical networks, there is a competing *space-division multiplexing* (SDM) technology. Here, several parallel single-wavelength links are installed between adjacent nodes. All data is transmitted on the same wavelength. The advantage is that there are no wavelength assignment problems for SDM networks; a disadvantage is the cost for the additional fiber cables, especially for MAN and WAN distances. It is also conceivable that SDM and WDM are combined in an all-optical network. Architectural and technological concepts for corporate optical backbone networks using low-cost components and a combination of WDM rings and SDM rings are developed in the COBNET project [BDO+98], for example. One of the factors that determine which of the proposed variants of all-optical networks (with respect to architecture, topology, wavelength conversion capabilities, multiplexing technique, etc.) will become the standard technology for high-performance networks of the future is the quality of resource allocation strategies for the individual variants.

A number of books cover the application of optics to computer science: the book by McAulay [McA91] presents the basics of optics in a comprehensive way; the books by Green [Gre93] and Mukherjee [Muk97] discuss optical communication networks.

1.1.2 ATM Networks

One example of networks with guaranteed quality of service are ATM networks. A comprehensive introduction to the concepts and architecture of ATM networks can be found in [Vet95]. ATM has become increasingly popular over the past few years, and it has been accepted as a standard for B-ISDN. ATM stands for *asynchronous time-division multiplexing* or *asyn-*

chronous transfer mode [ATM95]. In an ATM network, every connection request specifies a certain traffic type (constant bit-rate, variable bit-rate, or available bit-rate), the required bandwidth (sustained bandwidth, peak bandwidth, and burst length), and the preferred quality of service (mean cell delay, cell delay variation and cell loss ratio). If the connection is established, the network guarantees that the required resources will be available to the connection as long as it remains active.

Although several problems related to their efficient application have not yet found satisfactory solutions, it seems clear that ATM networks will be among the high-speed networks of the future. Communication in wide area networks as well as in local area networks can be done with ATM. Hence, WANs and LANs can use the same basic communication protocol, and it will no longer be necessary to have expensive devices that transform LAN traffic into WAN traffic and vice versa. ATM is currently being used for high-speed backbone networks on the MAN and WAN level, and is becoming increasingly important for high-performance LANs as well.

In ATM networks all data is transmitted in small, fixed-length packets called *cells*. Each cell contains 48 bytes of user data and 5 bytes of header information. Typical transmission speeds of physical links in ATM networks are 155 Mbps, 622 Mbps, or 2.4 Gbps. This bandwidth can be shared by different virtual connections with asynchronous time-division multiplexing: The cells belonging to different connections are interleaved and transmitted over the same link one by one. ATM switches can receive cells on many incoming links simultaneously and forward them to outgoing links with very little delay. Routing in ATM networks is based on the concepts of *virtual channels* and *virtual paths*; for details, please refer to one of the introductory textbooks on ATM networks (e.g., [Cla97]).

The most striking advantages of ATM networks over conventional networks include scalability, high bandwidth, guaranteed quality of service, statistical multiplexing, and the flexibility to support a variety of traffic types with different service requirements in one network (traffic integration) [KW95]. The high bandwidth and guaranteed quality of service in ATM networks are essential for emerging applications like multimedia networking [IMI+95], high-definition distance learning (HDDL) [SDRF95], or real-time medical imaging [RS92]. In addition, the high bandwidth ATM brings into LANs makes workstation clusters an attractive alternative to dedicated parallel computers for parallel and distributed computing.

The problem that still needs to be solved is how to utilize an ATM network efficiently in an environment where heterogeneous connection requests compete for the resources in the network.

We consider a simplified model of ATM networks (or other networks with similar characteristics) that we call *networks with bandwidth reservation*. In such networks, every link has a certain capacity (bandwidth), and every connection request (call) specifies a certain bandwidth requirement. The network guarantees that once the connection is established, the requested bandwidth is available to it as long as it remains active. Consequently, this bandwidth must be reserved on all links along a path that connects the endpoints of the call in the network.

Note that our model of networks with bandwidth reservation applies to ATM networks directly if only constant bit-rate connections are allowed. (In the context of ATM, a constant bit-rate connection is a connection that specifies a certain bit-rate in advance and then transmits data at exactly that bit-rate as long as the connection is active.) In practice, one wants to allow available bit-rate connections as well; these are connections that have no timing constraints and can tolerate arbitrary delays. Such available bit-rate connections are appropriate for electronic messages and file transfers, for example. They can be handled without interfering with constant bit-rate connections by an appropriate congestion control mechanism; hence, we can ignore available bit-rate connections in our model.

Another type of connections in ATM networks are variable bit-rate connections. They require the peak bandwidth only for short time intervals (bursts), while most of the time data is transmitted at a lower rate (sustained bandwidth). Connections with such characteristics arise from transmission of compressed audio or video, for example. If many such variable bit-rate connections share the bandwidth of a link, statistical multiplexing is effective: it is not necessary to reserve the peak bandwidth for each individual connection, because it is unlikely that all the connections have their bursts at the same time. In our model, we do not deal with variable bit-rate connections.

Given a set of calls in a network with bandwidth reservation, it is desirable to complete all the calls in minimum time. This can help utilize the network efficiently and speed up distributed applications. For each call, it must be determined when it should be established and which path it should take in the network. Once a call is established, it remains active for a period of time that depends on the call and that may be unknown in advance. Preemption (interrupting a call after it has been established) and rerouting (changing the path for a call after it has been established) are not allowed. While a call is active, it occupies its requested bandwidth on all links along its path; the sum of the bandwidth requirements of different active calls using a link must not exceed the capacity of that link. The problem of assigning paths and starting times to calls under these constraints with the goal of minimizing the latest completion time is modelled as *call scheduling*.

1.2 Thesis Outline

In Chapter 2, *Preliminaries*, the notation and concepts used throughout this thesis are introduced. Precise definitions of the optimization problems considered in subsequent chapters are given here as well. This chapter also includes an extensive review of related work.

Chapters 3 to 6 present our results with rigorous proofs. Chapter 3, *Complexity of Path Coloring and Call Scheduling*, investigates the computational complexity of path coloring and call scheduling in different restricted settings. It is shown that most variants are $\mathcal{NP}$-hard, and polynomial-time optimal algorithms are presented for some special cases.

The wavelength assignment problem in all-optical tree networks is tackled in Chapter 4, *Path Coloring in Trees*. For undirected tree networks of arbitrary degree, an approximation algorithm with absolute approximation ratio 4/3 and asymptotic approximation ratio 1.1 is derived from a known approximation algorithm for edge coloring of multigraphs. For bidirected tree networks of arbitrary degree, an approximation algorithm requiring at most $\lceil (5/3)L \rceil$ colors for sets of directed paths with maximum load L is presented. This is optimal within the class of local greedy algorithms, to which the algorithm and all other known algorithms for the problem belong.

Chapter 5, *MaxPC and MaxPP in Bidirected Trees*, deals with the call-admission problem in all-optical networks. MaxPC is a variant of the path coloring problem in which the number of available colors (wavelengths) is limited and the goal is to color a maximum cardinality subset of the given paths using the available colors. MaxPP is another variant that applies to optical networks with full wavelength conversion. It is shown that MaxPC and MaxPP can be solved optimally in polynomial time if the degree of the tree and the number of available wavelengths are both bounded by a constant. If one of these two parameters can be arbitrary, both problems are proved $\mathcal{NP}$-hard. For MaxPP, a 2-approximation algorithm is presented for arbitrary number of available wavelengths. In the case of one available wavelength, MaxPC and MaxPP are equivalent and correspond to the maximum edge-disjoint paths problem. For every fixed value of $\varepsilon > 0$, a polynomial-time $(5/3+\varepsilon)$-approximation algorithm for the maximum edge-disjoint paths problem in bidirected trees is presented. Using a known reduction from MaxPC with many wavelengths to MaxPC with one wavelength, an algorithm with approximation ratio 2.22 is obtained for MaxPC with arbitrary number of wavelengths.

Approximation algorithms for call scheduling in networks with bandwidth reservation are studied in Chapter 6, *Analysis of List-Scheduling Variants*. Calls with variable bandwidth requirements and either unit duration or arbitrary duration are considered. It is shown that variants of List-Scheduling, which are simple and easy to implement, have small, constant approximation ratio in stars and trees for calls with unit duration. For calls with arbitrary duration, batch-style on-line algorithms, which do not require advance knowledge of the duration of a call, with competitive ratio 5 for stars and competitive ratio $5 \log n$ for trees with n nodes are presented.

Finally, Chapter 7, *Conclusion*, summarizes the obtained results and discusses their implications. A number of possible extensions and directions for future research are suggested.

Most of the results presented in Chapters 3 to 6 have been obtained in joint work with Klaus Jansen. Preliminary versions of these results were announced at various conferences. In particular, the complexity results for path coloring and call scheduling from Chapter 3 were presented in part at PASA'96 [EJ97c] and in part at HICSS-30 [EJ97a]. From the results of Chapter 4, the approximation algorithm for path coloring in undirected trees was also presented at PASA'96 [EJ97c], and two preliminary versions of the approximation algorithm for path coloring in bidirected trees were presented as joint papers with Klaus Jansen, Christos Kaklamanis, and Pino Persiano at a DIMACS Workshop on Network Design [EJKP98] and at ICALP'97 [KPEJ97]. The modifications that allow a more efficient implementation, which are integrated into the presentation in Section 4.2, as well as experimental results were reported at WAE'98 [EJ98a]. The complexity and approximation results for MaxPC and MaxPP in bidirected trees from Chapter 5 are announced at ISAAC'98 [EJ98b]. Preliminary results regarding the analysis of list-scheduling variants in Chapter 6 were presented at WG'97 [EJ97b].

Chapter 2

Preliminaries

2.1 Notation and Basic Concepts

The set of natural numbers (excluding zero) is denoted by $\mathbb{N}$, and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. Furthermore, $\mathbb{Q}$ is the set of rational numbers. By $\mathbb{Q}_+$ we denote the set of positive rational numbers. The cardinality of a set M is denoted by $|M|$. The power-set of a set M, i.e., the set of all subsets of M, is denoted by 2^M. All logarithms are base 2.

An optimization problem P is given by a set $\mathcal{I}$ of instances (inputs), a set $\mathcal{S}$ of solutions (outputs), a function $s : \mathcal{I} \to 2^{\mathcal{S}}$ mapping instances to sets of feasible solutions, a function $val : \mathcal{I} \times \mathcal{S} \to \mathbb{N}_0$ measuring the quality $val(I, S)$ of solution $S \in s(I)$ for instance $I \in \mathcal{I}$, and a goal *min* or *max*. If the goal is *min*, one wants to find a solution $S \in s(I)$ that minimizes $val(I, S)$; if the goal is *max*, a solution $S \in s(I)$ that maximizes $val(I, S)$ is desired. We consider only optimization problems for which it is easy to check whether $I \in \mathcal{I}$ for a given I, to check whether $S \in s(I)$ for given I and S, and to compute $val(I, S)$ for given I and S. Further details regarding the formal definition of optimization problems (and a comprehensive coverage of the interrelation between approximation algorithms and probabilistically checkable proofs) can be found in [MPS98].

When we define particular optimization problems later on, we will present them in a more informal way, but it should always be clear how they can fit in the framework outlined above.

For considerations regarding the time complexity of optimization problems or decision problems, we use the model of a standard Turing machine as defined, e.g., in the book by Papadimitriou [Pap94]. We assume that instances of the optimization problems under consideration are encoded using an alphabet and any reasonable encoding; in particular, numbers are encoded using a logarithmic number of bits. However, reductions will be described at a more abstract level, keeping in mind that it would be straightforward to derive Turing machines executing the reductions in polynomial time.

For the presentation of algorithms, we use the RAM model of computation. The time complexity of problems in the RAM model and in the Turing machine model is polynomially related. For a discussion of the RAM model and for further prerequisites regarding the specification and analysis of algorithms the reader is referred to one of the available introductory textbooks on the design and analysis of algorithms, e.g., the book by Aho, Hopcroft, and Ullman [AHU76].

A decision problem P is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and there is a polynomial-time many-one reduction from every other problem in $\mathcal{NP}$ to P. The decision version of an optimization problem P is the decision problem obtained by adding a value $K \in \mathbb{N}_0$ to every instance I and asking the *yes/no* question "Is there an $S \in s(I)$ such that $val(I,S) \geq K$?" in case of a maximization problem or "Is there an $S \in s(I)$ such that $val(I,S) \leq K$?" in case of a minimization problem. We call an optimization problem P $\mathcal{NP}$-hard if its decision version is $\mathcal{NP}$-complete. For all optimization problems considered in this thesis it is easy to see that the decision version is in $\mathcal{NP}$; hence, we will prove $\mathcal{NP}$-hardness by reducing known $\mathcal{NP}$-complete problems to the decision versions of our optimization problems. We refer the reader to the book [GJ79] by Garey and Johnson for an introduction to the theory of $\mathcal{NP}$-completeness.

A polynomial-time algorithm that always produces an optimal solution for a given optimization problem is called an *exact* algorithm. The existence of an exact algorithm for an $\mathcal{NP}$-hard optimization problem would imply $\mathcal{P} = \mathcal{NP}$. Therefore, one is interested in polynomial-time approximation algorithms for $\mathcal{NP}$-hard optimization problems.

An approximation algorithm A for an optimization problem P is a deterministic algorithm whose running-time is polynomial in the size of the input and that always computes a feasible solution. Denote by $A(I)$ the value of the output of A on input I, i.e., $A(I) = val(I,y)$ where $y \in s(I)$ is the output produced by A on input I. Denote by $OPT(I)$ the value of an optimal solution to I, i.e., $OPT(I) = \max_{S\in s(I)} val(I,S)$ for maximization problems and $OPT(I) = \min_{S\in s(I)} val(I,S)$ for minimization problems. If P is a maximization problem, A has (absolute) approximation ratio ρ if $OPT(I)/A(I) \leq \rho$ for all inputs I. Similarly, if P is a minimization problem, A has (absolute) approximation ratio ρ if $A(I)/OPT(I) \leq \rho$ for all inputs I. If A has approximation ratio ρ, we call A a *ρ-approximation algorithm.* The approximation ratio of an algorithm is often referred to as its *performance ratio* or its *performance guarantee.*

Sometimes we are also interested in the *asymptotic approximation ratio* of an algorithm. For a minimization problem, an algorithm A has asymptotic approximation ratio ρ if $\limsup_{OPT(I)\to\infty} A(I)/OPT(I) \leq \rho$. For a maximization problem, an algorithm A has asymptotic approximation ratio ρ if $\limsup_{OPT(I)\to\infty} OPT(I)/A(I) \leq \rho$. An overview of approximation algorithms for $\mathcal{NP}$-hard problems can be found in the book [Hoc97] edited by Hochbaum.

As approximation algorithms usually require that the complete input is given to the algorithm in advance, they can be classified as *off-line* algorithms. For the optimization problems considered in this thesis, the input will (in part) consist of a set of connection requests. For some of the applications, it is realistic to assume that the connection requests arrive dynamically and the algorithm must process each request without knowledge about future requests; algorithms that work in such a scenario are called *on-line* algorithms. For other applications, it is realistic to assume that all connection requests are known to the algorithm in advance, but the duration of a connection is unknown until its completion. Such algorithms that work for connection requests with unknown duration are called *batch-style on-line* algorithms, because they receive a batch of connection requests with unknown duration as input. Please refer to the book by Borodin and El-Yaniv [BEY98] and the book edited by Fiat and Woeginger [FW98] for an introduction to on-line algorithms and an overview of the current state of the art.

The *competitive ratio* measures the worst-case approximation ratio of an on-line algorithm or batch-style on-line algorithm [ST85]. For a scheduling problem where the makespan (schedule length) is to be minimized, an on-line algorithm has competitive ratio ρ if it always produces a schedule whose makespan is at most a factor ρ longer than the makespan of the optimal (off-line) schedule.

We will mainly be concerned with off-line algorithms. The batch-style on-line scenario will be studied only in Chapter 6, where we assume that all calls are given to the call scheduling algorithm in advance, but that the duration of a call is unknown. As observed by Feldmann et al. [FMS$^+$95, Fel95], a batch-style on-line algorithm for call scheduling can easily be converted into a fully on-line algorithm, i.e., an algorithm that can deal with additional calls that arrive on-line while other calls have already been scheduled. A result due to Shmoys, Wein, and Williamson [SWW91] implies that this conversion increases the competitive ratio by no more than a factor 2. However, the makespan is not the only important criterion for the quality of a schedule in the fully on-line scenario; in addition, criteria like response time or network throughput must be considered. We will not deal with the fully on-line scenario.

2.1.1 Graphs

Graphs are a very important concept that is used throughout this thesis. For definitions that cannot be found here, we refer to any introductory text on graphs or graph algorithms, e.g., the books by Berge [Ber76] and Golumbic [Gol80].

An *undirected graph* $G = (V, E)$ consists of a finite set V of vertices (nodes) and a finite set E of edges, where each edge $e \in E$ joins two distinct vertices and is represented by a subset of V with cardinality 2. Two vertices joined by an edge are called *adjacent*. All vertices adjacent to a given vertex v are the *neighbors* of v. An edge $e = \{u, v\}$ is said to be *incident* to u and to v, and u and v are called its *endpoints*. The *degree* $\delta(v)$ of a vertex v is the number of edges incident to it. The set of edges incident to vertex v is denoted $\Gamma(v)$. A vertex with degree 0 is called an *isolated vertex*. An undirected graph is called *simple* if it does not contain parallel edges, i.e., if every pair of vertices is joined by at most one edge. A class of graphs has *bounded degree* if there is a constant c such that all vertices in graphs of the class have degree at most c.

A *path* from u to v in $G = (V, E)$ is a sequence

$$\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{r-1}, v_r\}$$

of edges such that $v_0 = u$ and $v_r = v$. The *length* of a path is the number of its edges. Unless specified otherwise, paths are assumed to be *simple*, i.e., we have $v_i \neq v_j$ for $0 \leq i < j \leq r$. A path with $v_i \neq v_j$ for $1 \leq i < j \leq r$ and $v_0 = v_r$ is called a (simple) *cycle*. We say that a path *runs from u to w* if it contains a path from u to w as a subpath. Two paths *intersect* if they share at least one edge. An undirected graph is *connected* if there is a path from any vertex to any other vertex. A *mesh* is a two-dimensional array of vertices such that a vertex is adjacent to its neighbors in the same column and in the same row. This definition can be extended to d-dimensional meshes in the obvious way. A d-dimensional mesh with $d = O(1)$ is called a *mesh of bounded dimension*.

Connected undirected graphs without cycles are called *trees*. In a tree, the vertices with degree 1 are called *leaves*, and the remaining vertices are called *internal vertices*. A tree is a *binary tree* if all its vertices have degree at most 3. A subset $E' \subseteq E$ of the edge set of an undirected, connected graph $G = (V, E)$ is a *spanning tree*, if $G' = (V, E')$ is a tree. A *star* is a special case of a tree that consists of a single vertex of arbitrary degree all of whose adjacent vertices have degree 1. (This notion of star is completely unrelated to the definition of a star graph as a graph whose vertices represent permutations and whose edges correspond to transpositions exchanging the first element of a permutation with any other element [AHK87].) The graphs obtained from stars by replacing edges by simple paths (i.e., by subdividing edges) are called *spiders*.

An undirected graph is called a *chain*, if it consists of a single path, and a *ring*, if it consists of a single cycle. Note that a chain is a special case of a tree. A *tree of rings* is an undirected graph that can be constructed from a single vertex by repeated application of the following operation: pick an arbitrary vertex v of the graph, add a disjoint ring that contains only new vertices, and identify v with an arbitrary vertex of the new ring. Observe that a tree of rings can be turned into a tree by removing an arbitrary edge from each ring used in its construction.

Any vertex r in a tree can be designated the *root* of the tree. In a rooted tree, the *level* of a vertex $v \in V$ is the length of the path from r to v. The root r has level 0. The maximum level among all vertices of the tree is the *depth* of the tree. For a vertex $v \neq r$ the unique neighbor of v whose level is one smaller than that of v is called the *parent* $p(v)$ of v; all neighbors of v whose level is one larger than that of v are called the *children* of v. Inductively, w is an *ancestor* of v either if $w = v$ or if w is an ancestor of $p(v)$. We denote by $lca(u, v)$ the *least common ancestor* (lca) of two vertices u and v, i.e., the vertex with smallest level on the path from u to v. The lca of a path is defined as the lca of its endpoints. The one or two edges on a path from u to w that are incident to $lca(u, w)$ are called the *top edges* of the path from u to w. A path *touches* a vertex v if it begins at v, ends at v, or contains v as an internal vertex.

In a rooted tree $T = (V, E)$, the *subtree* rooted at a vertex v consists of v and all vertices that have v as an ancestor as well as all edges between these vertices. Only such subtrees will be considered. We say that a subtree of T *contains* the path from u to w if $lca(u, w)$ is a vertex in that subtree.

An undirected graph $G = (V, E)$ is *bipartite* if V can be partitioned into disjoint sets V_1 and V_2 such that each edge in E joins a vertex in V_1 and a vertex in V_2. An undirected graph is *regular* if all its vertices have the same degree. If this degree is k, the graph is called *k-regular*. A 2-regular undirected graph is called a *cycle cover*, because its edges can be partitioned into a number of vertex-disjoint cycles such that each vertex is contained in exactly one cycle.

The *edge-expansion* of an undirected graph $G = (V, E)$ is the minimum ratio, taken over all non-empty subsets $S \subseteq V$ with $|S| \leq |V|/2$, of the number of edges leaving S to the cardinality of S.

In a *directed graph* $G = (V, E)$ every edge is directed from one vertex (called *tail*) to another vertex (called *head*) and represented by an ordered pair of vertices. (Note that directed edges are often called *arcs* in the literature; we reserve that term for arcs of a circle.) The edges with tail v are called *outedges* of v, and their number is the *outdegree* of v. The edges with head v are called *inedges* of v, and their number is the *indegree* of v. We say that a directed edge is *incident* to v if it is either an inedge or an outedge of v.

Paths and cycles in directed graphs are defined as in undirected graphs; consecutive vertices v_i and v_{i+1} on a path or cycle are joined by a directed edge (v_i, v_{i+1}). A directed path from u to w is sometimes written as (u, w), if no confusion with the directed edge (u, w) is possible.

Directed graphs are called *bidirected* if they can be obtained from an undirected graph by replacing each edge $\{u, v\}$ by two directed edges (u, v) and (v, u) with opposite directions. In particular, a *bidirected tree* is the graph obtained from an undirected tree in this way. The undirected graph from which a bidirected graph can be obtained is called its *underlying undirected graph*. In a bidirected graph, the outdegree of each vertex equals its indegree; hence, it is convenient to refer to the outdegree or indegree of a vertex simply as its degree. Given an undirected or bidirected graph, we use Δ to refer to the maximum degree of the graph. Note that bidirected graphs are sometimes called *symmetric directed graphs* in the literature.

In general we assume all graphs to be simple unless we speak of *multigraphs*. Multigraphs can have an arbitrary number of edges joining the same pair of vertices. On occasion we will also allow multigraphs that have self-loops (edges joining a vertex and itself). The *size* of a graph or multigraph $G = (V, E)$ is taken to be $|V| + |E|$.

An undirected graph $G = (V, E)$ is called a *line graph* if it can be obtained from a multigraph $G' = (V', E')$ (without self-loops) by setting $V = E'$ and inserting edges $\{e_1, e_2\}$ into E for all edges $e_1, e_2 \in E'$ that share an endpoint in G'.

2.1.2 Graph Searching

Often it is necessary for an algorithm to visit all vertices of a graph $G = (V, E)$ in a certain order. For connected, undirected graphs, a search procedure of the following general type can be used:

1. Pick a start vertex $s \in V$; mark s as *active* and all other vertices as *unvisited*.

2. While there is an active vertex v, pick such a vertex v, mark it as *visited* and mark all its unvisited neighbors as *active*.

Executing such a search procedure produces a rooted spanning tree of G: the start vertex is the root of the tree, and a vertex v is the parent of a vertex w if the procedure marked w *active* while visiting v.

Individual variants of this general search procedure differ with respect to the selection rule for active vertices in Step 2. *Depth-first search* (dfs) always chooses the vertex that has been marked *active* last among all presently *active* vertices, while *breadth-first search* (bfs) chooses the *active* vertex that has been marked *active* first. Seen from another viewpoint, dfs maintains the active vertices in a stack (LIFO), while bfs employs a queue (FIFO). The dfs procedure can also be implemented recursively, in which case the active vertices are implicitly maintained in the activation records on the execution stack of the program. In general, the stack-based implementation and the recursive implementation of dfs do not necessarily visit the vertices of the graph in exactly the same order unless the graph is a tree.

While the vertices of G are visited by a search procedure, numbers from 1 to $|V|$ can be assigned to them according to the order in which they are marked *visited*. If dfs is used, we refer to these numbers as *dfs-numbers*, and to the order in which the vertices are visited as *dfs-order*. Similarly, *bfs-numbers* and *bfs-order* can be defined. The spanning trees produced by dfs and bfs are called *dfs-trees* and *bfs-trees*, respectively. The running-time for dfs or bfs is linear in the size of the graph, i.e., $O(|V| + |E|)$.

2.1.3 Vertex Coloring

Given an undirected graph $G = (V, E)$, a *(vertex) coloring* of G is an assignment of colors to vertices such that vertices receive different colors if they are joined by an edge. Usually, colorings are represented by functions mapping V to $\mathbb{N}_0$. A coloring S must satisfy

$$\forall_{\{u,v\}\in E} S(u) \neq S(v) \ .$$

A coloring is called a *k-coloring* if it uses at most k different colors, i.e., if $|S(V)| \leq k$. For general graphs, it is $\mathcal{NP}$-hard to compute a coloring using the smallest possible number of different colors [GJ79].

2.1.4 Coloring of Paths

Given a set P of simple paths in a graph $G = (V, E)$, the *load* $L(e)$ of an edge $e \in E$ is the number of paths in P that contain edge e. The *maximum load* L of P in G is defined as $L = \max_{e\in E} L(e)$. The maximum load is also referred to as the *congestion* of P in G. A *coloring* of a set of paths is an assignment of colors to paths such that paths receive different colors if they share an edge. The *conflict graph* of a given set of paths is the graph with one vertex for each path and an edge between two vertices if the corresponding paths intersect. A coloring of a set of paths can be viewed as a vertex-coloring of the conflict graph. For a coloring $S : P \to \mathbb{N}_0$, we denote by $|S|$ the number of different colors used in the coloring. Again, a coloring that uses at most k colors is called a *k-coloring*. These definitions apply to undirected paths in undirected graphs and to directed paths in directed graphs.

2.1.5 Graph Matching and Edge Coloring

Given an undirected graph $G = (V, E)$, a subset $M \subseteq E$ is called a *matching* if no two edges in M share an endpoint. A *maximum matching* is a matching of maximum cardinality. Maximum matchings can be computed in time $O(\sqrt{|V|}|E|)$ in bipartite graphs [HK73] and in general graphs [MV80]. A matching M in G is called a *perfect matching* if $|M| = |V|/2$, i.e., if every vertex of G has an incident edge in M. In a regular bipartite graph $G = (V, E)$, there exists a perfect matching and it can be computed in $O(\Delta|E|)$ time [Sch98], where Δ is the degree of the vertices in G.

The notion of *b-matchings* [GLS88, pp. 257–259] is a generalization of the concept of matchings. Given an undirected graph $G = (V, E)$ and a function $b : V \to \mathbb{N}_0$, a b-matching is a function $x : E \to \mathbb{N}_0$ such that $\sum_{e \in \Gamma(v)} x(e) \le b(v)$ for every vertex $v \in V$. In other words, every edge may be included into the b-matching with arbitrary multiplicity, but the sum of the multiplicities of edges incident to a vertex v must not exceed $b(v)$. There is a polynomial-time algorithm for computing a b-matching of maximum weight, i.e., a b-matching for which $\sum_{e \in E} w(e)x(e)$ is maximum, where $w : E \to \mathbb{Q}$ is an arbitrary weight function. In particular, if $w(e) = 1$ for all $e \in E$, the algorithm produces a b-matching that maximizes the sum of the multiplicities of the edges.

If the multiplicities with which edges can be included into the b-matching are bounded by edge capacities $c(e)$, the resulting problem is called the *capacitated b-matching problem.* Given an undirected graph $G = (V, E)$, a function $b : V \to \mathbb{N}_0$, and a capacity function $c : E \to \mathbb{N}_0$, a *c-capacitated b-matching* is a function $x : E \to \mathbb{N}_0$ such that $x(e) \le c(e)$ for every edge $e \in E$ and $\sum_{e \in \Gamma(v)} x(e) \le b(v)$ for every vertex $v \in V$. Like for the standard b-matching problem, there is also a polynomial-time algorithm for computing a capacitated b-matching of maximum weight. In fact, the capacitated b-matching problem can be reduced to the standard b-matching problem [GLS88, p. 258].

An *edge coloring* of an undirected graph or multigraph $G = (V, E)$ is an assignment of colors to edges such that no two edges incident to the same vertex receive the same color. The *chromatic index* $\chi'(G)$ of G is the number of colors used in an optimal edge coloring, i.e., in an edge coloring using the least possible number of different colors. Note that the problem of edge coloring a multigraph G is equivalent to the problem of vertex coloring the line graph of G. It is $\mathcal{NP}$-complete to decide whether there is an edge coloring of a given 3-regular graph using three colors [Hol81]. The edges of a bipartite graph or multigraph, however, can always be colored using Δ colors, where Δ is the maximum degree of the graph or multigraph. Furthermore, such an edge coloring can be obtained in polynomial time. More precisely, an optimal edge coloring for a bipartite multigraph with maximum degree Δ and m edges can be obtained in time $O(m \log m)$ [CH82] or in time $O(\Delta m)$ [Sch98]. If a bipartite graph or multigraph $G = (V, E)$ is Δ-regular, an edge coloring of G corresponds to a partitioning of E into Δ perfect matchings. In Section 4.2 we will require a subroutine for edge coloring Δ-regular bipartite multigraphs. To simplify notation, we use $T_{\mathrm{ec}}(n, \Delta)$ to denote the time for edge coloring a Δ-regular bipartite multigraph with n vertices. Observe that $T_{\mathrm{ec}}(n, \Delta) \geq n\Delta/2$ (as the multigraph contains $n\Delta/2$ edges) and $T_{\mathrm{ec}}(n, \Delta) = O(\min\{\Delta^2 n, \Delta n \log(\Delta n)\})$ (using the algorithms mentioned above). Refer to the book [FW77] by Fiorini and Wilson for a comprehensive treatment of known results on edge coloring.

2.2 Problem Definitions

Now we are ready to formally define the optimization problems investigated in the following chapters. All the problems are related to resource allocation for connection requests or calls in a communication network. We model the given communication network by a graph $G = (V, E)$. Network nodes correspond to vertices of the graph, and physical links correspond to edges. In some cases, an undirected graph is a better model of the physical network, in other cases a bidirected graph.[1] All optimization problems introduced in the following can be investigated for both variants of graphs, and in most cases the relationship between the bidirected and the undirected case is less direct than one might expect.

As motivated in Chapter 1, we distinguish two basic types of communication networks: networks with bandwidth reservation and all-optical networks.

In a network with bandwidth reservation, every edge $e \in E$ has an associated capacity $c(e) \in \mathbb{Q}_+$. Every connection request specifies a certain bandwidth requirement, and this amount of bandwidth is reserved on all links along a path between the communication endpoints of the connection

[1] We do not consider networks whose topology is an arbitrary directed graph.

request at the time the connection is established. We refer to connection requests and connections in networks with bandwidth reservation as *calls*. A call $c = (u_c, v_c, b_c, d_c)$ is given by its communication endpoints $u_c \in V$ and $v_c \in V$, its bandwidth requirement $b_c \in \mathbb{Q}_+$, and its duration $d_c \in \mathbb{N}$. If the call c is established at time t_c, it will complete at time $t_c + d_c$.

The call-scheduling problem is to find a minimum makespan schedule for a given set of calls in a network with bandwidth reservation. Given a graph G, edge capacities $c(e)$ for all $e \in E$, and a set R of connection requests (calls), a *feasible schedule* S assigns to each call $c \in R$ a starting time $t_c \in \mathbb{N}_0$ and a path P_c from u_c to v_c in G such that $\forall_{e \in E, t \in \mathbb{N}_0} : \sum_{c \in A_t^e} b_c \leq c(e)$, where $A_t^e = \{c \in R \mid e \in P_c \wedge t_c \leq t < t_c + d_c\}$ is the set of calls using edge e that are active at time t. The makespan $|S|$ of a schedule S (also called the length of the schedule) is the latest completion time of a call, i.e., $\max_{c \in R} t_c + d_c$.

PROBLEM Call Scheduling

Input: graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{Q}_+$, set R of calls

Output: assignment of paths and starting times to calls such that the sum of the bandwidth requirements of simultaneously active calls using the same edge does not exceed the capacity of that edge

Goal: minimize the makespan of the schedule

Note that we assume that all calls are available at time 0 and do not have to be completed by a certain deadline. There is no precedence relation between the calls. We consider the off-line version of the call-scheduling problem, where call duration is known in advance, as well as the batch-style on-line version, where the duration of a call is unknown to the scheduling algorithm.

We will only deal with the case that all edges have the same capacity $c(e) = 1$. Furthermore, we will study restricted variants where we assume that either all calls have the same bandwidth requirement $b_c = 1$ or the same duration $d_r = 1$ or both. The most restricted variant, in which $c(e) = 1$ for all $e \in E$ and $b_c = d_c = 1$ for all calls c, is equivalent to the routing and wavelength assignment problem in all-optical networks (optical networks with wavelength-division multiplexing and reconfigurable wavelength-selective switches, as explained in Section 1.1.1) without wavelength conversion: time steps correspond to wavelengths.

In all-optical networks, a connection request between node u and node v is represented by the ordered pair (u, v). (If the network is undirected, the ordering of the nodes is irrelevant.) It is established by reserving one wave-

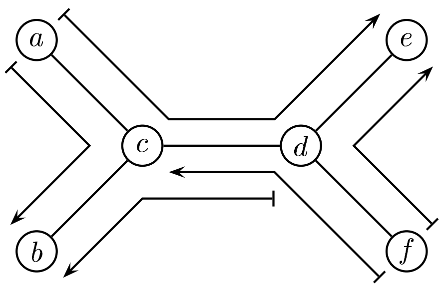


Figure 2.1: Directed paths in a bidirected tree

length on all links along a path from u to v. The paths corresponding to different connections must not use the same wavelength on the same link. If the switches do not support wavelength conversion, a connection must use the same wavelength on the whole path from transmitter to receiver. The routing and wavelength assignment problem in all-optical networks without wavelength conversion is commonly referred to as *path coloring*.

PROBLEM Path Coloring (PC)

Input:	graph $G = (V, E)$, set R of connection requests in G
Output:	assignment of paths and colors to requests such that paths using the same edge are assigned different colors
Goal:	minimize the number of colors used

If the graph is clear from the context, we denote by $OPT(R)$ the number of colors required for a set R of connection requests in an optimal solution.

An assignment of paths to the given connection requests is called a *routing*. Note that the routing has to be computed by the path coloring algorithm. If the routing is given as part of the input (instead of allowing the algorithm to choose the paths), the *fixed path coloring* problem is obtained. For tree networks, fixed path coloring and path coloring are equivalent, because the paths for connection requests are unique.

Figure 2.1 shows a simple example of an instance of the path coloring problem in a bidirected tree. The bidirected tree consists of six nodes; for the sake of simplicity, only its underlying undirected tree is drawn in the figure. A set of five paths is sketched: from a to e, from f to e, from f to c, from d to b, and from a to b. A possible valid coloring assigns these paths colors 1, 2, 1, 2, and 3, respectively. This coloring with three colors is optimal, because the conflict graph of the paths is a cycle of length 5.

In optical networks with support for full wavelength conversion, the requirement that a connection must use the same wavelength on the whole path from transmitter to receiver can be dropped. Hence, the number of wavelengths required to establish a set of connections is equal to the maximum load of the paths. We call the resulting optimization problem the *path packing* problem.

PROBLEM Path Packing (PP)
Input: graph $G = (V, E)$, set R of connection requests in G
Output: assignment of paths to requests
Goal: minimize the maximum load

The goal of the path packing problem is to compute a routing so that the maximum load is minimized. The maximum load achieved by an optimal routing for a given instance of path packing is denoted by L_{OPT}. Note that studying the path packing problem for tree networks is meaningless, because the paths and, therefore, the maximum load are completely determined by the connection requests.

Path packing applies to optical networks with space-division multiplexing as well. If a routing with maximum load L is found, all requests can be established if L parallel fiber links are available between adjacent nodes; hence, it is desirable to minimize the maximum load.

It is also interesting to study path coloring and path packing for sets of connection requests of special form. In particular, the set of connection requests for *all-to-all path coloring* or *all-to-all path packing* consists of connections from each node to every other node. (This communication pattern is also called *gossiping*.) Furthermore, *one-to-all* instances (also called *broadcast* instances) consist of connection requests from one node to all other nodes, k-*relations* are sets of connection requests in which every node is the sender and the receiver of at most k connection requests, and *permutations* are 1-relations. It is known that a k-relation can be split into $O(k)$ permutations. We will discuss some results regarding path coloring for special instances in the section on related work (Section 2.3).

Path coloring and path packing are relevant if a network provider designs a network so as to satisfy given demands or if an existing network has sufficient optical bandwidth to satisfy the given connection requests. The situation is different if an existing network has insufficient capacity to route all connections simultaneously; in this case, it is reasonable to select a subset of the given connection requests such that all requests in that subset can be established simultaneously. The goal can be to maximize the number of accepted requests. In all-optical networks without wavelength conversion this optimization problem is referred to as maximum path coloring

(MaxPC)[2], in networks with full wavelength conversion as maximum path packing (MaxPP).

PROBLEM Maximum Path Coloring (MaxPC)

Input:	graph $G = (V, E)$, set R of connection requests in G, positive integer W
Output:	subset $R' \subseteq R$ and assignment of paths and at most W colors to requests in R' such that paths using the same edge are assigned different colors
Goal:	maximize the cardinality of R'

Observe that the MaxPC problem can be viewed as a call-scheduling problem with deadlines: given a set of calls with unit bandwidth requirements and unit duration, a network with unit edge capacities, and a deadline $W \in \mathbb{N}$, the goal is to select a maximum cardinality subset of the given calls and schedule them before the given deadline.

PROBLEM Maximum Path Packing (MaxPP)

Input:	graph $G = (V, E)$, set R of connection requests in G, positive integer W
Output:	subset $R' \subseteq R$ and assignment of paths to requests in R' such that the maximum load is at most W
Goal:	maximize the cardinality of R'

Besides all-optical networks with wavelength converters, MaxPP applies also to optical networks with space-division multiplexing (W represents the number of parallel fiber links in this case) or to conventional networks with circuit switching (here, W represents the number of channels available on each link).

As with the fixed path coloring problem, one can also study the variants of MaxPC and MaxPP in which the paths for the connection requests are already given as part of the input. We refer to these variants as *MaxPC with fixed paths* and *MaxPP with fixed paths*, respectively. Note again that the variants with and without fixed paths are the same for tree networks.

If only a single wavelength is available, MaxPC and MaxPP are equivalent. This special case of MaxPC and MaxPP is usually called the *maximum edge-disjoint paths* problem.

[2]The name MaxPC was introduced by Nomikos and Zachos in [NZ97].

PROBLEM Maximum Edge-Disjoint Paths (MEDP)
Input: graph $G = (V, E)$, set R of connection requests in G
Output: subset $R' \subseteq R$ and assignment of paths to requests in R' such that the paths are edge-disjoint
Goal: maximize the cardinality of R'

Recall that all these problems can be studied for the undirected case and for the bidirected case. In networks with bandwidth reservation, the undirected case reflects symmetric communication requests like phone calls or video conferences, whereas the directed case models asymmetric communication patterns. In all-optical networks, the undirected case models an architecture with single bidirectional fiber links between adjacent nodes, while the bidirected case applies to the more common architecture with pairs of unidirectional fiber links.

Note that the definitions of the optimization problems above are given for the general case where the topology of the network is an arbitrary undirected or bidirected graph. We will mostly be dealing with the special case of tree networks. In tree networks, the path for a connection request is uniquely determined by its communication endpoints. We will refer to connection requests in tree networks as undirected or directed paths. Hence, we denote the set of connection requests given as input to one of the problems path coloring, path packing, MaxPC, or MaxPP in tree networks by P instead of R.

For a set R of calls with arbitrary duration and bandwidth requirements in a tree network $T = (V, E)$, the load of an edge $e \in E$ is defined as $L(e) = \sum_{r \in R : e \in P_r} b_r d_r$. Obviously, the load of an edge is a lower bound on the minimum schedule length for R.

Algorithms for tackling the above optimization problems in tree networks need not be concerned with the routing aspect of the problems; other interesting variants of the problems are obtained if the topology of the network is restricted to some other class of graphs, e.g., rings or meshes.

It should be remarked that the sets of connection requests or sets of calls that are part of the input for the optimization problems defined above can contain multiple connection requests or multiple calls with the same parameters, e.g., there can be several connection requests between the same two nodes. This means that every set of connection requests or set of calls can actually be a multiset; for the sake of simplicity, however, we will continue to call them sets.

2.3 Related Work

This section reviews a number of results that were obtained by different authors and that are relevant to our work in one way or another. In general, all work on network routing is closely related to the optimization problems defined in Section 2.2. However, we will mostly be concerned with tree networks, where the routing aspect of these optimization problems is trivial. Therefore, we don't discuss the extensive literature on routing here. Instead, we refer the interested reader to the book edited by Korte, Lovász, Prömel, and Schrijver [KLPS90], in particular the chapter written by Frank [Fra90], for a discussion of the complexity and polynomial-time solvable special cases of routing problems in the context of VLSI-layout; to Leighton's book [Lei92] for a comprehensive presentation of routing problems and algorithms for parallel computer architectures; to the book by Borodin and El-Yaniv [BEY98, Chapter 13] and to the survey by Leonardi [Leo98] for an overview of results for on-line routing problems.

2.3.1 Path Coloring and Path Packing

The path packing problem is closely related to the integral multicommodity flow problem. Every connection request corresponds to a demand for one unit of flow from its sender to its receiver, and the goal is to find the minimum capacity for the edges such that all demands can be routed through the network. It follows from [EIS76] that the path packing problem is $\mathcal{NP}$-hard for general networks in the bidirected and in the undirected case. A general technique to obtain (randomized) approximation algorithms for path packing under certain conditions is the randomized rounding technique due to Raghavan and Thompson; details of the method and applications to VLSI routing and multicommodity flow can be found in [RT87].

The path coloring problem, which models the wavelength allocation problem in all-optical networks with wavelength division multiplexing, has received considerable interest in the past few years.

Chlamtac, Ganz, and Karmi [CGK92] proved that the fixed path coloring problem is $\mathcal{NP}$-hard for arbitrary networks (this holds already for mesh networks, as shown by Harder, Lee, and Choi [HLC97]), but can be solved in polynomial time in directed acyclic graphs. For sets of paths with maximum load L in a directed acyclic graph, they showed that L colors are sufficient. Furthermore, they proposed shortest path routing and greedy wavelength assignment heuristics for path coloring and MaxPC in general networks. Extensive simulation results regarding the performance of these heuristics in irregular network topologies are reported in [CGK92] as well. They also studied the dynamic case in which lightpaths are established and terminated over time.

Banerjee and Mukherjee proposed to tackle the path coloring problem by using an LP relaxation and randomized rounding [RT87] to solve the routing problem, while employing a graph coloring heuristic to color the conflict graph of the resulting paths [BM96]. They reported that experiments with random inputs showed that this heuristic yields solutions whose number of colors is close to the lower bound obtained from the LP relaxation.

Some work on path coloring dealt with the question of how many colors are required for permutation routing, i.e., for sets of connection requests in which every node is the sender and the receiver of at most one connection. For networks of bounded degree with n nodes, Pankaj proved that $\Omega(\log n)$ wavelengths are required [Pan92]. Aggarwal et al. constructed a network for which $O(\log n)$ wavelengths suffice for permutation routing [ABNC+94, ABNC+96]. They also showed that $O(L \cdot \min\{d, \sqrt{m}\})$ colors always suffice to color paths with maximum load L and maximum path length (dilation) d in a network with m edges, and that there are networks and sets of connection requests where this many colors are in fact required. Other related work on permutation routing concentrated on bounds on the number of switches and the number of wavelengths in all-optical networks with nonreconfigurable routers or wavelength-independent switches [BH94, ABNC+94, ABNC+96, RU94].

Raghavan and Upfal studied path coloring for k-relations [RU94]. They showed that for every $\beta \leq 1$ and $k \in \mathbb{N}$, there is a bounded-degree graph with edge-expansion β and a k-relation such that $\Omega(k/\beta^2)$ colors are required. They also presented a randomized algorithm for arbitrary bounded-degree graphs that uses at most $O(k(-3\log(kn)/\log\lambda)^2)$ colors with high probability, where λ is the second largest eigenvalue (in absolute value) of the transition matrix of the standard random walk on G. For permutations ($k = 1$) and in terms of the edge-expansion β, this amounts to $O(\log^2 n/\beta^4)$ colors ($O(\log^2 n/\beta^2)$ for some graphs). Aumann and Rabani gave an improved algorithm that requires only $O(\log^2 n/\beta^2)$ colors for a permutation in any bounded-degree graph with edge-expansion β and n nodes [AR95].

Path coloring for one-to-all instances and all-to-all instances in bidirected graphs of several general classes has been investigated by Bermond et al. in [BGP+96]. A survey of these and other graph problems arising from wavelength-routing in all-optical networks was written by Beauquier et al. [BBG+97]. More recent references to results regarding path coloring for one-to-all instances, all-to-all instances, and permutation instances can be found in the survey by Klasing [Kla98].

Tree Networks

Since the topologies of existing wide-area networks are mainly trees and trees of rings, wavelength allocation in tree networks has been a focus of attention.

For chain networks (also called linear arrays), which are a special case of tree networks, the path coloring problem can be solved optimally in polynomial time. This follows because the conflict graph of paths in a chain is an interval graph. It is well-known that interval graphs can be colored optimally in polynomial time [Gav72]. This solves the path coloring problem for bidirected chains as well as for undirected chains.

Raghavan and Upfal gave a polynomial-time algorithm that requires at most $(3/2)L$ colors for coloring any set of paths with maximum load L in an undirected tree; hence, the algorithm is a $(3/2)$-approximation algorithm [RU94]. As a subroutine, their algorithm uses an edge-coloring algorithm that requires at most $(3/2)\Delta$ colors for a multigraph with maximum vertex degree Δ (this edge-coloring algorithm is due to Shannon [Sha49]; it is explained, e.g., in [FW77]).

Path coloring in undirected tree networks can be viewed as coloring edge intersection graphs of paths in a tree. These EPT graphs were already examined by Golumbic and Jamison in [GJ85a] and [GJ85b]. The former proved recognition of EPT graphs $\mathcal{NP}$-complete, the latter proved $\mathcal{NP}$-hardness of the coloring problem for EPT graphs (thus implying that path coloring for undirected trees is $\mathcal{NP}$-hard). Tarjan gave a $(3/2)$-approximation algorithm for coloring EPT graphs [Tar85], which was rediscovered in [RU94] as mentioned above. Note that EPT graphs are different from *path graphs*, which are the *vertex*-intersection graphs of paths in a tree and represent a special case of *chordal graphs* [Gol80].

It was observed in [MKR95] that the $(3/2)$-approximation algorithm for path coloring in undirected trees can be improved to give asymptotic approximation ratio $9/8$ using the edge-coloring algorithm by Goldberg [Gol84b, Gol84a]. In Section 4.1 we use the edge-coloring algorithm by Nishizeki and Kashiwagi [NK90] to obtain an algorithm with asymptotic approximation ratio $11/10$ and absolute approximation ratio $4/3$.

The directed version of path coloring in tree networks was first considered by Mihail, Kaklamanis, and Rao in [MKR95]. (Unlike EPT graphs, the conflict graphs of directed paths in a bidirected tree had not received attention previously.) They presented an approximation algorithm that uses at most $(15/8)L$ colors to color a set of paths with maximum load L. This was subsequently improved to $(7/4)L$ independently by Kaklamanis and Persiano in [KP96] and by Kumar and Schwabe in [KS97].

In Section 4.2 we will further improve this to $\lceil(5/3)L\rceil$. Jansen proved that every local greedy algorithm uses at least $\lfloor (5/3)L \rfloor$ colors in the worst case [Jan97], even in the case of binary trees. All known algorithms for path coloring in bidirected trees are local greedy algorithms, i.e., they consider the nodes of the tree in depth-first search order and extend an existing partial coloring by choosing colors for the uncolored paths touching the current node.

The algorithm from Section 4.2 for path coloring in arbitrary bidirected trees is quite technical; simpler algorithms requiring at most $\lfloor (5/3)L \rfloor$ colors for the special case of path coloring in binary trees were obtained by Jansen [Jan97] and by Caragiannis, Kaklamanis, and Persiano [CKP97].

The question of how many colors are required even in an optimal solution for sets of directed paths with maximum load L in a bidirected tree has also been studied. The example shown in Figure 2.1 on page 21 represents an instance with $L = 2$ for which an optimal coloring requires three colors. A much more complicated construction gives a set of directed paths in a binary tree such that the maximum load is three and an optimal solution requires five colors [Jan97]. Furthermore, there is a family of instances with arbitrarily large load L for which an optimal coloring uses $\lceil (5/4)L \rceil$ colors [KS97].

Many of the complexity results for path coloring in undirected and bidirected trees that we will present in Section 3.1 have been obtained independently and contemporaneously by Kumar, Panigrahy, Russel, and Sundaram [KPRS97]. In particular, they showed that path coloring is $\mathcal{NP}$-hard for undirected stars (previously proved in [GJ85b]), for bidirected trees of depth two, and for bidirected binary trees. Furthermore, they gave a polynomial-time optimal algorithm for path coloring in undirected trees of bounded degree, and they proved that path coloring in arbitrary undirected or bidirected graphs of constant size can be solved optimally in polynomial time; this result can be obtained by generalizing the proof of Theorem 3.1.7 in a straightforward way.

All-to-all path coloring in bidirected trees was studied by Gargano, Hell, and Perennes in [GHP97]. They proved that the directed paths in an all-to-all instance can always be colored with L colors, where L is the maximum load of the paths. They also observed that among all bidirected trees, only spiders (called *generalized stars* in [GHP97]) have the property that any set of paths with maximum load L can be colored with L colors.

Ring Networks and Trees of Rings

A 2-approximation algorithm for path coloring in ring networks can be obtained easily. It suffices to remove an arbitrary link from the ring and to route all requests so that they do not use the removed link. The conflict graph of the resulting paths is an interval graph, which can be colored optimally in polynomial time. The fact that this yields a 2-approximation was observed for undirected ring networks in [RU94] and for directed ring networks in [MKR95]; see also Section 3.2.

As observed by Raghavan and Upfal [RU94], this edge-removal technique can be generalized to trees of rings: If there is a path coloring algorithm A for trees that uses at most αL colors (for some $\alpha \geq 1$) for any set of paths with maximum load L, an algorithm using at most $2\alpha OPT$ colors for a set of connection requests in a tree of rings can be obtained, where OPT is the number of colors in an optimal solution for the tree of rings. It suffices to remove an arbitrary edge from each ring in the tree of rings and to apply algorithm A to the resulting tree; this way, a 3-approximation algorithm for path coloring in undirected trees of rings can be obtained from the $(3/2)L$-algorithm mentioned above, and a 10/3-approximation algorithm for path coloring in bidirected trees of rings can be derived from the $(5/3)L$-algorithm we present in Section 4.2.

A number of results for path coloring and path packing in bidirected ring networks were obtained by Wilfong and Winkler in [WW98]. Using a linear relaxation of an integer program as a subroutine, they devised an exact algorithm for path packing, i.e., a polynomial-time algorithm that produces a routing with load L_{OPT}. In addition, they showed that there are instances of path coloring for which $2L_{\text{OPT}} - 1$ colors are necessary, and that there is a polynomial-time approximation algorithm using at most $2L_{\text{OPT}} - 1$ colors, which gives a 2-approximation (just like the method that removes an arbitrary link of the ring). They also proved that the path coloring problem in bidirected rings is $\mathcal{NP}$-hard. Their proof for this was found independently from the proof we will present in Section 3.2 (Theorem 3.2.1); while the proof in [WW98] is somewhat simpler, our proof establishes $\mathcal{NP}$-hardness for path coloring in bidirected rings *and* undirected rings.

A weighted variant of the path packing problem for undirected ring networks was studied by Schrijver, Seymour, and Winkler in [SSW98]. Every connection request is associated with a nonnegative integral demand, and the goal is to find a routing that minimizes the maximum load, where the load of an edge is the sum of the demands routed through it. They gave a polynomial-time algorithm that achieves a maximum load that exceeds the optimal load by at most 3/2 times the maximum demand. If all demands are equal to 1, the problem is equivalent to the path packing problem and solvable optimally in polynomial time; this follows from a result by Okamura and Seymour [OS81], as shown by Frank et al. in [Fra85, FNS+92].

Fixed path coloring in undirected or bidirected ring networks is equivalent to the coloring problem for circular-arc graphs (ARC-COLORING). An approximation algorithm using at most $2L - 1$ colors was given by

Tucker in [Tuc75], and the problem was proved $\mathcal{NP}$-hard by Garey et al. in [GJMP80]. A (5/3)-approximation algorithm for arc-coloring was presented by Shih and Hsu in [SH90].

Kumar used randomized methods to devise improved approximation algorithms for arc-coloring and for path coloring in undirected rings [Kum98]. He obtained a randomized algorithm for arc-coloring that achieves asymptotic approximation ratio $1+1/e \approx 1.37$ with high probability if $\ln n = o(L)$, where n is the number of distinct arc endpoints and L is the width of the arc set. For path coloring in undirected rings, he obtained a randomized algorithm that achieves asymptotic approximation ratio $1.5 + 1/(2e) \approx 1.68$ with high probability if $\ln n = o(OPT)$, where n is the number of nodes of the ring and OPT is the optimal number of colors.

The problems of all-to-all path coloring and all-to-all path packing for bidirected ring networks were solved by Wilfong in [Wil96]. He showed that every routing has load at least k_n, where $k_n = (n^2-1)/8$ if n is odd and $k_n = \lceil n^2/8 \rceil$ if n is even, and he proved that a routing with load k_n can indeed be achieved and that k_n colors are sufficient for coloring the resulting paths.

Mesh Networks

Kramer and van Leeuwen gave an $\mathcal{NP}$-hardness proof for a wire routing problem that occurs in the context of VLSI theory [KvL84, Theorem 6]. Simple modifications of that proof show that the maximum edge-disjoint paths problem in undirected and bidirected (two-dimensional) mesh networks is $\mathcal{NP}$-complete. Therefore, path coloring and path packing are both $\mathcal{NP}$-hard for undirected and bidirected meshes, and there are no polynomial-time approximation algorithms with absolute approximation ratio smaller than 2 unless $\mathcal{P} = \mathcal{NP}$.

A randomized algorithm for path coloring in d-dimensional meshes that uses at most $O(kdn^{1/d})$ colors for k-relations with high probability was presented by Raghavan and Upfal in [RU94]. For path coloring in meshes of bounded dimension with n nodes, an approximation algorithm using at most $O(\log n \log |R| OPT(R))$ colors for an arbitrary set R of connection requests was presented by Aumann and Rabani in [AR95]. Kleinberg and Tardos improved this and obtained an algorithm with approximation ratio $O(\log n)$ for two-dimensional meshes with n nodes [KT95a]. The best known algorithm for two-dimensional meshes has approximation ratio polynomial in $\log\log n$; it was obtained by Rabani in [Rab96]. He also presented an algorithm that outputs an upper bound on the number of colors so that the upper bound is an $O(1)$-approximation to the optimal number. Some of these results for two-dimensional meshes actually extend to a slightly larger class of mesh-like planar graphs.

Wavelength Converters

The problem of bandwidth allocation in all-optical networks with wavelength converters has also been investigated. A (full) wavelength converter can receive an incoming signal on one wavelength and transmit it on an outgoing link on another wavelength. Therefore, a single path may be assigned different wavelengths on different subpaths if the subpaths are separated by a node with a wavelength converter.

Wilfong and Winkler observed in [WW98] that a single wavelength converter in a bidirected or undirected ring suffices to ensure that L wavelengths are enough to color any set of paths with maximum load L. Since there are exact algorithms for path packing in undirected and bidirected ring networks, this implies that an optimal assignment of wavelengths can be computed in polynomial time for ring networks with at least one wavelength converter. For arbitrary bidirected graphs $G = (V, E)$, they defined a *sufficient set* $S \subseteq V$ to be a set of nodes such that if exactly the nodes in S are equipped with wavelength converters, L wavelengths are enough to route any set of paths in G with maximum load L. They proved that a set S is sufficient for G if and only if the graph $G(S)$, which is obtained from G by "exploding" the nodes in S (see [WW98]), is a disjoint union of spiders. In addition, they proved that finding a sufficient set of minimum cardinality is $\mathcal{NP}$-hard even for planar graphs. Kleinberg and Kumar obtained a 2-approximation algorithm for the minimum sufficient set problem in arbitrary bidirected graphs by relating it to the vertex cover problem [KK99]. They also presented a polynomial-time primal-dual algorithm that yields a 2-approximation for arbitrary directed graphs. (See the chapter by Goemans and Williamson [GW97] in the book edited by Hochbaum for an introduction to primal-dual algorithms.)

Wavelength allocation in bidirected tree networks in the presence of (full) wavelength converters was studied by Auletta, Caragiannis, Kaklamanis, and Persiano in [ACKP97]. They proved that there is always a sufficient set consisting of at most $\lfloor (n-2)/4 \rfloor$ nodes in an arbitrary bidirected tree with n nodes, and that such a set can be found in polynomial time. They also constructed a binary tree with n nodes for which $\lfloor (n-2)/4 \rfloor$ wavelength converters are indeed necessary. In addition, they studied *limited wavelength converters*, i.e., converters that can convert wavelength i into wavelength j only for a subset of all possible pairs (i, j). Further results regarding limited wavelength converters in all-optical tree networks can be found in [ACKP98a, ACKP98b, Gar98].

On-line Path Coloring

In the on-line version of the path coloring problem the algorithm is given connection requests one by one and must assign paths and colors immediately without knowledge about future requests. For path coloring in tree networks or for fixed path coloring in arbitrary networks, this problem is a special case of on-line graph coloring: the goal is to color the conflict graph of the paths, and the vertices of this graph are revealed to the algorithm one by one. (The survey by Kierstead [Kie98] gives an overview of on-line graph coloring.)

A classic result for on-line coloring of interval graphs due to Kierstead and Trotter [KT81] provides on-line algorithms for path coloring in undirected and bidirected chain networks; these algorithms use at most $3OPT(P) - 2$ colors for a set P of paths in a chain network. Furthermore, no on-line algorithm can use fewer than $3OPT(P) - 2$ colors in the worst case [KT81]. The algorithm for on-line coloring of interval graphs can be generalized to an on-line algorithm for coloring circular-arc graphs that uses at most $4OPT - 2$ colors [MHIR96]. This gives an on-line algorithm for fixed path coloring in bidirected or undirected rings with competitive ratio 4.

Bartal and Leonardi [BL97] obtained deterministic on-line algorithms with competitive ratio $O(\log n)$ for path coloring in networks with n processors whose topology is that of a tree, a tree of rings, or a mesh. In addition, they presented a matching lower bound of $\Omega(\log n)$ for all on-line algorithms for path coloring in meshes, and a lower bound of $\Omega(\log n/ \log\log n)$ for path coloring in trees. Note that the on-line version of the path coloring problem corresponds to a call-scheduling problem where the algorithm must assign starting times to call requests one by one before the first call is established. Hence, the lower bounds in [BL97] do not apply to the batch-style on-line algorithms for the call-scheduling problem that we will present in Chapter 6.

The dynamic scenario in which the algorithm receives both connection requests and termination requests has been considered by Gerstel, Sasaki, Kutten and Ramaswami in [GSKR97]. They announced algorithms that use $O(L \log n)$ colors for paths with load L in rings or trees with n nodes. They also considered the case of limited wavelength conversion.

Recently, Leonardi and Vitaletti gave lower bounds on the competitive ratio of all on-line algorithms for path coloring that hold even for randomized algorithms [LV98]. They proved that the expected number of colors used by any randomized on-line algorithm for a set P of paths in a chain network is at least $3OPT(P) - 2 - o(1/OPT(P))$ in the worst case, thus extending the lower bound from [KT81] to randomized algorithms. Furthermore, they gave an $\Omega(\log \Delta)$ lower bound on the competitive ratio of randomized algorithms for on-line path coloring on trees of diameter $\Delta = O(\log n)$ with n nodes.

2.3.2 MaxPC, MaxPP, and Multicommodity Flow

In the following, we will review known results and some related work concerning MaxPC, MaxPP, and multicommodity flow. Since we study the off-line variant of MaxPC and MaxPP, we mention only a few references with work on on-line algorithms. More information about the rich field of on-line algorithms for call-admission and circuit routing can be found in [BEY98, Chapter 13] and [Leo98]; variants of path packing and MaxPP for connection requests with variable bandwidth requirements are also discussed there.

IP formulations for MaxPC and MaxPP and their LP relaxations were used by Ramaswami and Sivarajan in [RS95] to obtain upper bounds on the number of connections that can be established using only the available wavelengths. They generalized the upper bounds to the case of random arrivals of connection requests, and also showed that the upper bounds are asymptotically tight. Their work includes simulation results comparing the performance of heuristics with the upper bounds.

In (undirected or bidirected) chain networks, MaxPC and MaxPP are equivalent and can both be solved optimally in polynomial time by finding a maximum W-colorable subgraph in the conflict graph, which is an interval graph in this case [YG87].

In undirected tree networks, the maximum edge-disjoint paths problem is equivalent to the maximum independent set problem for EPT graphs; a polynomial-time optimal algorithm for the latter was given by Tarjan [Tar85]. Using the reduction from many wavelengths to one wavelength from Section 5.4, this gives an approximation algorithm whose approximation ratio is $e/(e-1) \approx 1.58$ for MaxPC in undirected trees. The approximation algorithm for integral multicommodity flow in trees (see below) from [GVY93] gives a 2-approximation algorithm for MaxPP in undirected trees (using the same adaptation that we employ for bidirected trees in Section 5.2).

Wan and Liu showed that the maximum edge-disjoint paths problem for bidirected or undirected ring networks can be solved optimally in polynomial time. With the reduction from Section 5.4, this gives a 1.58-approximation algorithm for MaxPC in bidirected or undirected ring networks. For MaxPC with fixed paths in undirected or bidirected ring networks, Nomikos and Zachos presented a (3/2)-approximation algorithm in [NZ97]. While the $\mathcal{NP}$-hardness of MaxPC in ring networks follows from the $\mathcal{NP}$-hardness of path coloring (in all four cases, i.e., with and without fixed paths and in bidirected or undirected rings), to our knowledge it is still open whether there are exact algorithms for (some of the variants of) MaxPP in ring networks.

The maximum edge-disjoint paths problem for undirected meshes, which is $\mathcal{NP}$-hard [KvL84], has been investigated by a number of researchers.

An $O(\log n)$-approximation algorithm for two-dimensional meshes with n nodes was presented by Aumann and Rabani in [AR95] and, independently, by Kleinberg and Tardos in [KT95a]. Subsequently, Kleinberg and Tardos gave an $O(1)$-approximation algorithm for this problem [KT95b]. Using the technique from Section 5.4, this gives an $O(1)$-approximation algorithm for MaxPC in two-dimensional meshes (in fact, for a slightly more general class of planar graphs). Note that the constant hidden by the $O(1)$ is very large for these algorithms. A randomized on-line algorithm with competitive ratio $O(\log n)$ for the maximum edge-disjoint paths problem in meshes was also given in [KT95b]. This implies a randomized on-line algorithm with competitive ratio $O(\log n)$ for the MaxPC problem in meshes as well.

For MaxPC with fixed paths in two-dimensional meshes, Nomikos and Zachos showed in [NZ97] that no approximation algorithm with constant approximation ratio can exist unless $\mathcal{P} = \mathcal{NP}$.

A different version of the MaxPC problem was considered by Awerbuch, Azar, Fiat, Leonardi, and Rosén in [AAF$^+$96]. They studied the case where each wavelength is associated with a different network topology, and they gave a general technique to obtain a $(\rho + 1)$-approximation for arbitrary number of wavelengths by repeated application of a ρ-approximation for one wavelength. This technique will be used in Section 5.4 to obtain algorithms for MaxPC from algorithms for the maximum edge-disjoint paths problem. It can be applied to randomized algorithms and to on-line algorithms with the same increase in competitive ratio. Awerbuch et al. obtained randomized on-line algorithms with polylogarithmic competitive ratio for their variant of MaxPC in switchless networks shaped as rooted directed forests.

MaxPP is strongly related to the integral multicommodity flow problem. For a given set of source-sink pairs (commodities) in an undirected graph with edge capacities, the multicommodity flow problem is to maximize the sum of the flows of the commodities constrained by the given edge capacities.[3] For the *integral* multicommodity flow problem, the flow through an edge is required to be integral for every commodity. In the case of unit edge capacities, the integral multicommodity flow problem is equivalent to the maximum edge-disjoint paths problem. Integral multicommodity flow and multicut have been studied for undirected trees by Garg, Vazirani, and Yannakakis in [GVY93]. They obtained exact algorithms for integral multicommodity flow in undirected trees of depth one with arbitrary edge capacities and for arbitrary undirected trees with unit edge capacities (the polynomial-time solvability for the latter case was already shown by Tarjan [Tar85]). For trees with edge capacities 1 or 2, they proved the problem $\mathcal{NP}$-hard

[3] In fact, this is only one out of several variants of the multicommodity flow problem.

and MAX SNP-hard. For trees with arbitrary edge capacities, they gave a 2-approximation algorithm. The algorithm can be adapted to give a 2-approximation for MaxPP in undirected trees, as mentioned above, and we will use the algorithm to obtain a 2-approximation for MaxPP in bidirected trees in Section 5.2.

An interesting combination of path coloring and MaxPC, called *path multicoloring*, was studied by Nomikos, Pagourtzis, and Zachos in [NPZ97]. A multicoloring of a set of paths is an assignment of colors such that several paths through an edge may be assigned the same color. Given a set P of paths in a graph and a number W of available colors, the goal of path multicoloring is to color all paths in P using W colors such that the number of color collisions in minimized. More formally, the number of color collisions $cc(e)$ on an edge e is defined as the maximum number of paths through e that are assigned the same color, and the goal is to find a multicoloring that minimizes $\sum_{e \in E} cc(e)$. Nomikos et al. obtained an exact algorithm for path multicoloring in chain networks and a 2-approximation algorithm for rings. Path multicoloring is relevant for optical networks that combine space-division multiplexing and wavelength-division multiplexing.

2.3.3 Call Scheduling

Feldmann et al. initiated research on on-line call scheduling in [FMS+95]. They motivated the investigation of call-scheduling problems by observing that the utilization of a network is likely to improve if calls are not rejected in times of overload, but deferred and established at a later time. First, they considered the case that the call-admission algorithm is allowed to defer a call at most for a time interval that is bounded by a constant times the (known) duration of the call. In this scenario, they proved that no deterministic or randomized on-line algorithm can achieve a competitive ratio smaller than $\Omega(\log n)$ with respect to the data-admission ratio (see [FMS+95]). Consequently, they considered the call-scheduling problem as defined in Section 2.2. They also focused on batch-style on-line algorithms, because they observed that conversion of a batch-style on-line algorithm into a fully on-line algorithm can be done losing at most a factor of 2 in the competitive ratio (for certain algorithms, only an additive term of 1 is lost).

They analyzed the competitive ratio of the List-Scheduling (LS) algorithm (see Section 6.1; the algorithm is called GREEDY in [FMS+95]) for call scheduling in a number of different network topologies. They also considered only the case that all edges of the network have the same capacity. For the case of arbitrary bandwidth requirements, they showed that the schedule produced by LS can be longer than an optimal schedule by a factor of $\Omega(n)$

even for calls with unit duration in a chain network with n nodes. Furthermore, they gave examples of calls with unit duration and unit bandwidth requirements in a binary tree with n nodes such that the schedule produced by LS is longer than the optimal schedule by a factor of $\Omega(\log n)$.

Regarding upper bounds, Feldmann et al. proved that LS has competitive ratio at most $2(c+1)\log n$ in a binary tree with n nodes, if all bandwidth requirements are either bounded from above by $1 - 1/c$ or bounded from below by $1/c$ for an arbitrary constant $c > 1$. By running LS once on the calls with bandwidth requirements at most $1/2$ and once on the calls with bandwidth requirements greater than $1/2$, they obtained a batch-style on-line algorithm for call scheduling in binary trees with competitive ratio $12 \log n$. They also proved a lower bound of $\Omega(\log\log n/\log\log\log n)$ on the competitive ratio of any deterministic on-line algorithm for call scheduling in binary trees. Furthermore, they generalized the LS algorithm to arbitrary networks with small edge separators by restricting the routing to *reasonable* paths with respect to the separators (see [FMS+95]). Their result implies a call-scheduling algorithm with competitive ratio $O(\sqrt{n})$ for arbitrary planar graphs, for example.

For chain networks, Feldmann et al. showed that the competitive ratio of LS is between 4.4 and 25.8 in the case of unit bandwidth requirements. For the case of arbitrary bandwidth requirements between $1/c$ and $1/2$, where $c \geq 2$ is an arbitrary constant, they gave a different algorithm that achieves competitive ratio c.

In [Fel95], Feldmann presented, in addition to the results from [FMS+95], a variant of LS with competitive ratio $O(\log^2 n)$ for call scheduling in meshes with n nodes. Furthermore, she obtained a variant of LS that uses only two-hop paths for call scheduling in complete graphs with n nodes. She proved that this variant has constant competitive ratio. (The exact bound given in [Fel95] is 78.)

2.3.4 Scheduling of File-Transfers

Coffman et al. studied a file-transfer scheduling problem that is closely related to call scheduling in stars [CGJL85]. An instance of the file-transfer scheduling problem is given by a file-transfer multigraph $G = (V, E)$ with integer node labels $p(v)$ (representing port constraints) and edge labels $d(e)$. An edge $e = \{u, v\}$ corresponds to a requested file-transfer connection between node u and node v with duration $d(e)$. The goal is to find a minimum makespan schedule such that at any instant no node v is involved in more than $p(v)$ active file-transfer connections. This model corresponds to call scheduling in a star with varying edge capacities (representing port constraints) and calls with unit bandwidth requirements and arbitrary duration (representing edges of G). [CGJL85] contains complexity results for various restricted versions of the problem (file-transfer graph structure, port constraints, edge multiplicity), approximation results (approximation ratio of List-Scheduling and variants of it), and distributed implementations of file-transfer scheduling algorithms. However, most of their results do not apply to the call-scheduling problem with unit edge capacities and arbitrary bandwidth requirements, which we study in Chapter 6.

2.3.5 Multiprocessor Scheduling

Scheduling calls with unit bandwidth requirements in stars with unit edge capacities is equivalent to scheduling multiprocessor tasks with prespecified processor allocations if each task requests one or two processors. (Processors correspond to edges of the star, tasks correspond to calls, and task execution time corresponds to call duration.) The computational complexity of this multiprocessor scheduling problem was investigated by Hoogeveen, van de Velde, and Veltman in [HvdVV94]. They proved the problem strongly $\mathcal{NP}$-hard for three processors, which implies that call scheduling with unit edge capacities, unit bandwidth requirements, and arbitrary call duration is strongly $\mathcal{NP}$-hard in a star of degree three. Furthermore, they gave an exact algorithm for their multiprocessor scheduling problem in the case of constant number of processors and unit execution time; their proof uses ideas by Blazewicz et al. [BDW86], and we will use the same ideas to prove Theorem 3.1.7 in Section 3.1.1. Hoogeveen et al. also proved $\mathcal{NP}$-hardness results for variants of their scheduling problem with release dates, with precedence constraints, and with a different objective function (minimizing the sum of the completion times instead of the makespan).

Note that the call-scheduling problem in tree networks can also be seen as a special case of multiprocessor scheduling with resource constraints: edges correspond to resources, calls correspond to tasks, and there are more processors than tasks. Garey and Graham showed that the approximation ratio of List-Scheduling (LS) for multiprocessor scheduling with resource constraints is $s+1$, where s is the number of resources [GG75]. It follows that LS has approximation ratio at most $m+1$ for call scheduling in a tree network with m edges.

Chapter 3

Complexity of Path Coloring and Call Scheduling

This chapter studies the complexity of path coloring and call scheduling for various special cases. Each variant is either proved $\mathcal{NP}$-hard, or a polynomial-time exact algorithm is given. In Sections 3.1 and 3.2, we consider the path coloring problem, which is equivalent to call scheduling in the setting where calls have unit bandwidth requirements and unit duration.

First, in Section 3.1, we examine path coloring in tree networks. For the special case of undirected binary trees, a simple greedy algorithm is shown to produce the optimal solution. Then we give a polynomial-time exact algorithm to solve the undirected path coloring problem in any undirected tree network of bounded degree. The main ingredient of this result is a proof that an optimal edge coloring in multigraphs with a bounded number of nodes (but unbounded number of edges) can be computed in polynomial time, either by a dynamic programming approach or by integer linear programming with a fixed number of variables. If the degree of the nodes of the tree is unbounded, however, path coloring in undirected trees (even in stars) is proved $\mathcal{NP}$-hard, and approximating it with absolute approximation ratio smaller than 4/3 is also proved $\mathcal{NP}$-hard. It is shown that path coloring in undirected trees is completely equivalent to edge coloring of multigraphs and that approximation algorithms for the two problems are interchangeable.

For path coloring in bidirected trees, we prove that restricting the degree does not help: the problem is already $\mathcal{NP}$-hard for binary trees. This resolves an open question in [MKR95]. For bidirected trees of arbitrary degree, we prove again that approximation with absolute approximation ratio smaller than 4/3 is $\mathcal{NP}$-hard. For comparison, the best known approximation algorithm (presented in Section 4.2) uses at most $\lceil (5/3)L \rceil$ colors, where L is the maximum load and a lower bound on the optimal solution.

Note that many of the complexity results mentioned so far for path coloring in bidirected and undirected trees have been obtained contemporaneously and independently by Kumar, Panigrahy, Russel, and Sundaram [KPRS97].

In addition, we show that path coloring is solvable optimally in polynomial time if the number of paths touching a single node of the tree network is bounded by a constant, for undirected trees as well as for bidirected trees. Such an assumption may be valid in real telecommunication applications if each customer submits only a limited number of call requests to the network and if most calls are local calls. The corresponding algorithm works by trying out all possible colorings for paths touching a single node and combining for each node the information obtained from its children.

Section 3.2 proves $\mathcal{NP}$-hardness for path coloring in undirected and bidirected ring networks. A very simple approximation algorithm gives an approximation ratio of 2, but devising an algorithm with a substantially better approximation ratio seems difficult.

Regarding call scheduling in the general setting with arbitrary bandwidth requirements and duration, we give some $\mathcal{NP}$-hardness results in Section 3.3. In particular, scheduling calls with arbitrary bandwidth requirements on a single link, scheduling calls with arbitrary duration in chain networks, and scheduling calls with either arbitrary bandwidth requirements or arbitrary duration in networks that contain at least two edge-disjoint paths between some pair of nodes are all $\mathcal{NP}$-hard.

3.1 Path Coloring in Tree Networks

Recall that an undirected graph $G = (V, E)$ is a tree if it is connected and does not contain a cycle, and that a bidirected tree is the graph obtained from an undirected tree by replacing each undirected edge by two directed edges with opposite direction.

Note that the path coloring problem is solvable optimally in polynomial time for undirected and bidirected chain networks, because it is equivalent to the vertex coloring problem for interval graphs [Gav72].

Obviously, the maximum load L of the given set of paths in the tree is a lower bound on the optimal number of colors necessary to color the paths. On the other hand, it is not difficult to see that $2L$ colors always suffice, both in the undirected and the bidirected case. This upper bound can be tightened to $(3/2)L$ for undirected trees (see [Tar85]) and to $\lceil (5/3)L \rceil$ for bidirected trees (see Section 4.2).

Algorithm: Level Algorithm

Input: undirected binary tree $G = (V, E)$, set P of paths in G

begin

(1) compute for each node $v \in V$ the set $P(v)$

(2) **for** color $c = 0, 1, \ldots$ **do**

(3) **for** all nodes $v \in V$, processed in order of non-decreasing levels **do**

begin

(4) choose ≤ 2 paths in $P(v)$ to be assigned color c:

(i) if $P(v)$ contains a two-sided path p and both downward links of v are still available for this color, choose that path p

(ii) otherwise, choose one left path p_1, if possible, and one right path p_2, if possible

(5) remove the chosen paths from $P(v)$ and assign them color c

end

(6) terminate when all paths are colored

end

Figure 3.1: The Level Algorithm

3.1.1 Undirected Trees

Given a set P of undirected paths in an undirected tree, colors must be assigned to all paths in P such that paths receive different colors if they share an edge. The goal is to minimize the number of colors used.

A Simple Algorithm for Binary Trees

First, we examine the case that $G = (V, E)$ is an undirected binary tree with n nodes. In [FMS$^+$95] it was shown that in this setting the greedy algorithm, i.e., the algorithm that considers the paths one by one and assigns each the smallest available color, can require a number of colors that is greater than the optimal number by a factor of $\Omega(\log n)$. We present a simple polynomial-time algorithm that always produces an optimal coloring.

Let w be the root of the binary tree G (any node of G with degree at most 2 can be used as the root). Recall that the level of a node is its distance from the root. For a node v of the tree, we refer to the (undirected) edges joining v and its children as the *downward links* of v.

For a path $p \in P$ from u to v, we let z_p denote the node $lca(u, v)$. For every node $v \in V$, the set $P(v) \subseteq P$ is the set of all paths $p \in P$ with $z_p = v$. With respect to a node v, a path $p \in P(v)$ is called a *two-sided* path if it uses both downward links of v, otherwise a *one-sided* path. Note that every one-sided path starts at v. More specifically, one-sided paths are called *left* paths or *right* paths depending on which of the two downward links they use (assuming an arbitrary ordering of the children of each node).

The path coloring algorithm for undirected binary trees is given in Figure 3.1. Note that whenever the algorithm reaches step (4), at most one of the two downward links is reserved by a path that has been assigned the current color at a previous node. We call this algorithm the *Level Algorithm*. Obviously, it can be implemented to run in time polynomial in the size of G and P.

Theorem 3.1.1 *The Level Algorithm computes an optimal coloring for paths in undirected binary trees.*

Proof: We describe how to transform any optimal coloring into the coloring produced by the Level Algorithm without increasing the number of colors used, thereby showing that the Level Algorithm is optimal. Let P be the set of paths given to the algorithm. Denote by C^* and C the number of colors used in the optimal solution S^* and in the solution S produced by the Level Algorithm, respectively. The transformation proceeds color by color.

Assume that S and S^* are identical for all colors from 0 up to $c-1$. (This assumption holds trivially for $c = 0$.) We explain how to modify S^* such that S and S^* become identical for all colors from 0 up to c. This modification does not increase the number of colors used in S^*. Hence, S^* can be transformed into S by repeated application of such modifications, and the theorem follows.

For the modification of S^* for color c, the nodes of G are processed in order of non-decreasing levels. When processing a node v at level ℓ, we can therefore assume that S and S^* are already identical with respect to paths that touch nodes with level smaller than ℓ. We call these paths *fixed paths* (with respect to v). Note that there can be at most one fixed path (w.r.t. v) touching v. Hence, color c can be in use by such a fixed path on at most one of the downward links of v, and it is the same downward link both in S and in S^*. Color c has not been assigned to fixed paths using the remaining one or two downward links (*free* links), but may have been assigned to paths from $P(v)$ in S or S^*. We distinguish three cases in our presentation of the modifications that are necessary to make S and S^* identical w.r.t. node v for color c as well (we call S and S^* *identical w.r.t. node v for color c* if the paths on all edges incident to v assigned color c are the same in S and S^*):

1. If no paths on the free downward links are assigned color c in coloring S, all paths of $P(v)$ that could have been assigned color c have

already been assigned a smaller color. Therefore, no such path has been assigned color c in coloring S^* either. Hence, S and S^* are already identical w.r.t. node v for color c, and we can proceed to the next node.

2. If a two-sided path p of $P(v)$ is assigned color c in S, there must exist a color $c' \geq c$ such that p is assigned color c' in S^*. If $c \neq c'$, we exchange the paths in the subtree rooted at v assigned color c in S^* and those assigned color c'. Obviously, S^* remains a feasible solution with the same number of colors. S and S^* are now identical w.r.t. node v for color c, and we can proceed to the next node.

3. If one or two one-sided paths p_1 and (possibly) p_2 are assigned color c in S, there must exist colors $c'_1 \geq c$ and $c'_2 \geq c$ such that p_1 and p_2 are assigned color c'_1 and c'_2 in S^*, respectively. If $c \neq c'_1$, we exchange the paths in the left subtree of v assigned color c in S^* and those assigned color c'_1. Analogously, if $c \neq c'_2$, we exchange the paths in the right subtree of v assigned color c in S^* and those assigned color c'_2. This is always possible because it cannot occur that a two-sided path of $P(v)$ is assigned color c in S^* while two one-sided paths are assigned color c in S. Again, the exchanges preserve the feasibility of S^* and do not increase the number of colors used. S and S^* are now identical w.r.t. node v for color c, and we can proceed to the next node.

The exchanges performed at a node v do not affect the work we have done at nodes processed previously for this color or for previous colors. Therefore, after processing at most n nodes, colorings S and S^* are identical for all colors up to c. □

Note that the existence of an exact algorithm for path coloring in undirected binary trees follows also from the more general result we will present later as Corollary 3.1.8. However, the Level Algorithm is of independent interest because it is considerably simpler and faster than the algorithm we will use to obtain Theorem 3.1.7 and Corollary 3.1.8.

Combining Local Colorings is Easy

Given a tree $G = (V, E)$ (of arbitrary degree) and a set P of paths in G, denote by P_w (for any $w \in V$) the subset of P that contains all paths touching w. We refer to the problem of coloring paths in a set P_w as *computing a local coloring*. Next, we investigate how individual local colorings can be combined to a global coloring without increasing the number of colors.

Theorem 3.1.2 *Given colorings S_w for the sets of paths P_w for all $w \in V$, there is a polynomial-time algorithm that computes a coloring S for P satisfying $|S| = \max_{w\in V} |S_w|$.*

Proof: The merging process starts at an arbitrary node v. Initially, we set $S = S_v$ and mark v as the only node that has already been processed. During the whole construction $|S|$ will remain less than or equal to $\max_{w\in V} |S_w|$. We repeatedly merge S and a coloring S_w, where w is a node that has not yet been processed but that is adjacent to a previously processed node (for example, visit the nodes using a depth-first search). Note that the node w is adjacent to exactly one previously processed node u, and that all paths that are contained in both S and S_w use the edge $\{u, w\}$. Call these paths the *intersecting* paths. Let T denote the subtree of G that contains w and all nodes reachable from w without using edge $\{u, w\}$. The intersecting paths are the only paths in S that touch T. Furthermore, the intersecting paths are the only paths in S_w that touch $G \setminus T$. Hence, the intersecting paths are the only paths that have to be taken care of when merging S and S_w.

We permute the coloring S_w (i.e., we rename the colors) in such a way that the resulting coloring is a feasible coloring for P_w with the intersecting paths being assigned the same colors as in S. Since the permuted coloring for P_w and the coloring S are "compatible" (the intersecting paths are assigned the same colors), we can merge them in the obvious way and obtain a coloring S'. No new color is necessary to obtain S' from S and S_w in this way. Therefore, the number of colors used in S' is equal to $\max\{|S|, |S_w|\}$, and we can use S' as the new S. We mark w as processed and continue with the next node. This process terminates after finitely many steps. The coloring S we obtain in the end uses $\max_{w\in V} |S_w|$ colors. Hence, we obtain a coloring S using the required number of colors for the whole set of paths P. $\Box$

As a consequence, optimal colorings for the sets P_w can be merged into an optimal coloring for P in polynomial time.

Corollary 3.1.3 *For any set P of undirected paths in an undirected tree, the number of colors used in an optimal coloring, denoted $OPT(P)$, is given by $OPT(P) = \max_{w\in V} OPT(P_w)$.*

Computing Local Colorings is Difficult

Now we will show that the approach of computing optimal local colorings for all P_w and combining them into an optimal coloring for P is not feasible, because it is already $\mathcal{NP}$-hard to compute an optimal coloring for a set P_w. This is a consequence of the following theorem, which was proved by Golumbic and Jamison in [GJ85b, Theorem 5]; we include a short proof here for the sake of completeness.

Note that paths in a set P_w intersect if and only if they share an edge incident to w. Therefore, computing a local coloring is equivalent to path coloring in a star.

Theorem 3.1.4 (Golumbic and Jamison, 1985) *Given a multigraph G with maximum degree Δ, it is possible to compute in polynomial time a set P of paths with maximum load Δ in a star G' such that the conflict graph of P is (isomorphic to) the line graph of G, and vice versa.*

Proof: Let $G = (V, E)$ be the given multigraph. Construct the star $G' = (V', E')$ by setting $V' = V \cup \{x\}$, where $x \notin V$, and $E' = \{ \{v, x\} \mid v \in V \}$. Build P by including a path from v to w (in G') for every edge $\{v, w\} \in E$. Observe that two paths in P intersect if and only if the corresponding edges in G share a vertex. Thus, the conflict graph of P is (isomorphic to) the line graph of G, as required. Furthermore, note that the maximum load of P in G' is equal to the maximum degree of G.

For the other direction, let $G' = (V', E')$ be the given star (with central node x) and let P be the given set of paths in G'. Let k be the number of paths in P that use only one edge of G', i.e., paths that start at x. Construct the multigraph $G = (V, E)$ as follows. Set $V = (V' \setminus \{x\}) \cup \{u_1, \ldots, u_k\}$, where $u_i \notin V'$ for $1 \leq i \leq k$. For every path from $v \neq x$ to $w \neq x$ in P, include an edge $\{v, w\}$ into E; for the i-th path from x to $v \neq x$ in P, include an edge $\{v, u_i\}$ into E. Observe that two edges of G share an endpoint if and only if the corresponding paths in G' share an edge. Hence, the conflict graph of P is (isomorphic to) the line graph of G. Furthermore, note that the maximum degree of G is equal to the maximum load of P in G'. □

A consequence of the theorem is that the class of graphs that can be obtained as conflict graphs of paths in a star is exactly the class of line graphs of multigraphs. Since edge coloring is equivalent to vertex coloring of line graphs and was shown $\mathcal{NP}$-hard by Holyer [Hol81], the theorem implies that path coloring is $\mathcal{NP}$-hard for undirected paths in stars. Therefore, coloring the paths in a set P_w is also $\mathcal{NP}$-hard.

On the positive side, the construction from the proof of Theorem 3.1.4 shows that approximation algorithms for edge coloring of multigraphs and for path coloring in trees are interchangeable.

Corollary 3.1.5 *Let $f : \mathbb{N}_0 \to \mathbb{N}_0$ be an arbitrary non-decreasing function, i.e., a function satisfying $f(a) \leq f(b)$ for $a \leq b$. If there is an approximation algorithm A for edge coloring of multigraphs that uses at most $f(\chi'(G))$ colors* *for any multigraph G, it is possible to derive an approximation algorithm B for path coloring in undirected trees such that B uses at most $f(OPT(P))$ colors for any given set P of paths, and vice versa.*

Proof: Given an approximation algorithm A for edge coloring that uses at most $f(\chi'(G))$ colors for a multigraph G, construct algorithm B as follows. Let an instance of the path coloring problem be given by a tree $T = (V, E)$ and a set P of paths in T. For every $w \in V$, construct a multigraph G_w from the set P_w of paths touching w as in the proof of Theorem 3.1.4, use A to compute an edge coloring for G_w, and convert it to a coloring for the paths in P_w. This coloring uses at most $f(\chi'(G_w)) = f(OPT(P_w))$ colors. Use the merging algorithm from Theorem 3.1.2 to obtain a coloring for P using at most

$$\max_{w \in V} f(OPT(P_w)) = f(\max_{w \in V} OPT(P_w)) = f(OPT(P))$$

colors. Algorithm B calls algorithm A as a subroutine $|V|$ times, and every such call takes time polynomial in the size of the input; hence, the total running-time of algorithm B is polynomial in the size of the input.

Given an approximation algorithm B for path coloring that uses at most $f(OPT(P))$ colors for any set P of paths in an undirected tree, construct algorithm A as follows. For a given multigraph G, construct a set P of paths in a star like in the proof of Theorem 3.1.4. Now the paths in P can be colored using $f(OPT(P)) = f(\chi'(G))$ colors using algorithm B, and the coloring can be converted to an edge coloring of G using the same number of colors. □

Note that an approximation algorithm for path coloring in undirected trees with approximation ratio smaller than 4/3 could be used to decide whether the edges of a given multigraph can be colored with three colors or not. Furthermore, it is known that the edges of a 3-regular simple graph can always be colored with either three or four colors, but it is $\mathcal{NP}$-hard to decide whether three colors suffice [Hol81]. Thus, we obtain the following corollary.

Corollary 3.1.6 *For undirected paths in stars (and, therefore, also in arbitrary undirected tree networks), it is $\mathcal{NP}$-hard to approximate the optimal coloring with absolute approximation ratio smaller than $4/3$.*

Undirected Trees of Bounded Degree

Next, we examine the path coloring problem in undirected trees whose degree is bounded by an arbitrary constant c. The following theorem on edge coloring of multigraphs with bounded number of nodes implies that it is solvable in polynomial time. The basic idea is similar to the one employed by Blazewicz et al. in [BDW86] to schedule multiprocessor tasks with bounded parallelism and unit execution time.

Theorem 3.1.7 *Let $G = (V, E)$ be a multigraph with $|V| = n$ and $|E| = m$, possibly with self-loops. If n is bounded by a constant c, an optimal edge coloring for G can be computed in polynomial time.*

Proof: We give two alternative polynomial-time algorithms for computing an optimal edge coloring for G, a dynamic programming approach and an integer linear programming approach. Both make use of the fact that if n is bounded by a constant, the number of different possible types of edges of G and the number of different types of matchings of G, i.e., sets of pairwise disjoint edges, are both bounded by a constant as well. The former is bounded by $k = \binom{c}{2} + c$, because there are at most $\binom{c}{2}$ different types of edges between different nodes and at most c different types of self-loops. The latter is trivially bounded by 2^k, because this is the number of different subsets of the set of types of edges. Each set of disjoint edges constitutes a potential *color class*; its edges can be assigned the same color in an edge coloring.

The multigraph G can be represented by a vector $\vec{g} \in \mathbb{N}^k$, where the i-th component g_i gives the number of edges of type i in G. Let $l \leq 2^k$ be the number of different possible sets of pairwise disjoint edges in a multigraph with at most c nodes. Then the possible sets of pairwise disjoint edges can be represented as vectors $\vec{b}_1, \ldots, \vec{b}_l \in \{0,1\}^k$, where the i-th component of $\vec{b}_j$ is 1 if and only if an edge of type i belongs to the j-th set.

The dynamic programming approach computes the minimum number of colors for an edge coloring of every subgraph of G, starting with the smallest subgraphs. (Here, a subgraph of G is a multigraph with the same set of vertices as G, but whose edge set is an arbitrary subset of the edge set of G.) For a multigraph $\vec{a} \in \mathbb{N}^k$, the minimum number of colors $OPT(\vec{a})$ is given by $\min_{i=1,\ldots,l}\left\{OPT(\vec{a} - \vec{b}_i) + 1\right\}$ and can be computed in constant time if $OPT(\vec{h})$ has already been computed for all multigraphs $\vec{h}$ with $\vec{h} < \vec{a}$. For a given multigraph $\vec{g}$, the results of these computations are stored in a table with $(g_1 + 1) \cdot (g_2 + 1) \cdot \ldots \cdot (g_k + 1) - 1 = O(m^k)$ entries before $OPT(\vec{g})$ can be computed. The running-time for the algorithm is $O(m^k)$, where k is a constant depending on c as shown above. It is easy to see that this approach cannot only compute the minimum number of colors, but also a minimum edge coloring for G.

The integer linear programming approach is described next. Obviously, our edge-coloring problem is equivalent to the following ILP:

Minimize
$$\sum_{j=1}^{l} x_j$$
subject to
$$\sum_{j=1}^{l} x_j \cdot \vec{b_j} = \vec{g}$$
$$x_j \geq 0$$
$$x_j \text{ integer}$$

Intuitively, each x_j gives the number of color classes of type j appearing in the minimum edge coloring. The number of variables in this ILP is l, which is a constant depending on c. Due to a result by Lenstra [Len81], integer linear programming with a fixed number of variables is solvable in polynomial time. □

If we have an undirected tree $G = (V, E)$ whose degree is bounded by a constant c, the path coloring problem for each individual set P_w corresponds to an edge-coloring problem in a multigraph whose number of nodes is bounded by c as well (employ the construction from the proof of Theorem 3.1.4, but insert self-loops in the multigraph for paths starting at w). Hence, we can use one of the algorithms given in the proof of Theorem 3.1.7 to compute an optimal coloring S_w for each P_w and combine these individual colorings according to Theorem 3.1.2.

Corollary 3.1.8 *The path coloring problem in undirected trees whose degree is bounded by a constant can be solved optimally in polynomial time.*

3.1.2 Bidirected Trees

Given a set P of directed paths in a bidirected tree, colors must be assigned to all paths in P such that paths receive different colors if they share a directed edge. The goal is to minimize the number of colors used.

Hardness for Binary Trees

Unlike in the undirected case, restricting the degree of the tree does not seem to make path coloring in bidirected trees easier. For the proof of the following theorem, we use a reduction from the $\mathcal{NP}$-complete problem ARC-COLORING. A graph $G = (V, E)$ is a circular-arc graph if its vertices can be represented by arcs of a circle such that there is an edge between two vertices in G if and only if the corresponding arcs intersect. The vertex-coloring problem for circular-arc graphs is referred to as ARC-COLORING, and its decision version is known to be $\mathcal{NP}$-complete [GJMP80].

Theorem 3.1.9 *Path coloring in bidirected binary trees is $\mathcal{NP}$-hard.*

Proof: (by reduction from ARC-COLORING)
An instance I of ARC-COLORING is given by a family $F = \{A_1, A_2, \ldots, A_n\}$ of circular arcs and a positive integer K. Each arc $A_i \in F$ is given by a pair (a_i, b_i) with $a_i \neq b_i$ and $a_i, b_i \in \{1, \ldots, m\}$. We call a_i and b_i the *endpoints* of arc A_i. Intuitively, the set $\{1, \ldots, m\}$ represents points that are located consecutively around a circle.

The *span* $sp(A_i)$ of arc A_i is the set $\{a_i + 1, a_i + 2, \ldots, b_i\}$ if $a_i < b_i$ and $\{a_i + 1, a_i + 2, \ldots, m, 1, \ldots, b_i\}$ if $a_i > b_i$. Two arcs A_i and A_j intersect if $sp(A_i) \cap sp(A_j) \neq \emptyset$. It is an $\mathcal{NP}$-complete problem to decide whether the arcs in F can be colored with K colors such that arcs with the same color do not intersect [GJMP80].

Note that we can assume without loss of generality that $m = O(n)$ (n arcs can have at most $2n$ different endpoints), that $K \leq n$ (otherwise, the answer is *yes*, because n arcs can always be colored with n colors), and that each number i, $i \in \{1, \ldots, m\}$, is contained in the span of exactly K arcs (otherwise, new arcs of the form $(i, i+1)$ or $(m, 1)$ can be introduced without changing the K-colorability). Furthermore, we can assume that m is even (otherwise, choose the next larger multiple of 2 as m).

Intuitively, the idea behind the transformation is to view the arcs as directed paths (with, say, counter-clockwise orientation) in a ring network, and to "squash" the ring so as to obtain a bidirected chain network. The obtained paths in the chain network, however, are not necessarily simple. Therefore we split each problematic path into two (or even three) simple paths. Additional care will be taken to ensure that the split paths resulting from the same original path are assigned the same color in any K-coloring. For this purpose, we need to attach some additional nodes to the chain, turning it into a binary tree.

A formal presentation of the reduction follows. We transform instance I of the ARC-COLORING problem into an instance I' of the path coloring problem in a binary tree $T = (V, E)$. The set of paths for I' is denoted by P. T consists of a chain with $\frac{m}{2} + 2K$ nodes, where each of the leftmost K nodes and each of the rightmost K nodes is connected to the root of a distinct binary tree of depth 1. The structure of the binary tree T is shown in Figure 3.2; note that the underlying undirected tree is drawn in the figure. The nodes of the chain are, from left to right, l_K, $l_{K-1}, \ldots$, l_2, l_1, v_1, $v_2, \ldots$, $v_{\frac{m}{2}}$, r_1, $r_2, \ldots$, r_{K-1}, r_K. For each node l_i, $1 \leq i \leq K$, there are three additional nodes l_i^1, l_i^2, and l_i^3, with incident edges as indicated in Figure 3.2. Similarly, there are three such nodes r_i^1, r_i^2, and r_i^3 for each r_i, $1 \leq i \leq K$. Each node v_i corresponds to points i and $m + 1 - i$ on the circle in instance I, the other nodes are used for the turn-around of problematic paths. Regarding the span of arcs, each edge $\{v_i, v_{i+1}\}$ corresponds to the

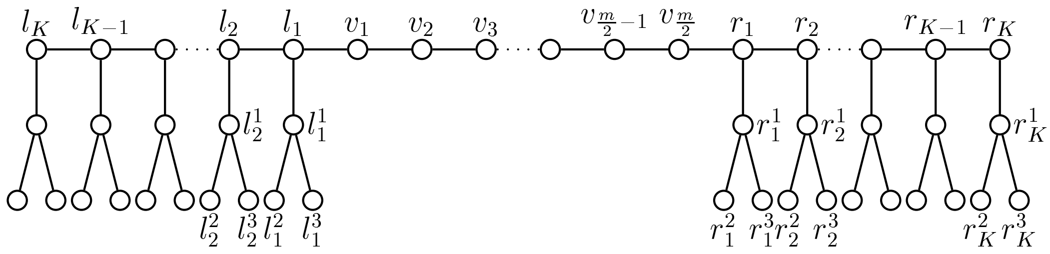


Figure 3.2: Binary tree network T

point $i+1$, if used in direction from left to right, and to the point $m+1-i$, if used in direction from right to left. The right part of T (the part to the right of $v_{\frac{m}{2}}$) corresponds to the point $\frac{m}{2}+1$, the left part (the part to the left of v_1) to point 1.

Each arc (a_i, b_i) in F can be classified into one of six distinct groups, depending on which of the following conditions it satisfies:

(1) $a_i < b_i \le \frac{m}{2}$	(3) $b_i < a_i \le \frac{m}{2}$	(5) $a_i > \frac{m}{2}, \quad b_i \le \frac{m}{2}$
(2) $a_i \le \frac{m}{2}, \quad b_i > \frac{m}{2}$	(4) $b_i > a_i > \frac{m}{2}$	(6) $a_i > b_i > \frac{m}{2}$

Figure 3.3 sketches the corresponding paths we put into P for each arc in group (1), (2) and (3). Groups (4), (5) and (6) are treated analogously.

Arcs in groups (1) and (4) are not problematic because they span neither point 1 nor point $\frac{m}{2}+1$. For each such arc, we have exactly one corresponding path in P. For an arc in group (1) the corresponding path is (v_{a_i}, v_{b_i}), for an arc in group (4) it is $(v_{m+1-a_i}, v_{m+1-b_i})$. Arcs in the remaining groups are problematic, because they span point 1 or point $\frac{m}{2}+1$ (or both) and the corresponding paths have to "turn around" at one or both ends of T. Arcs in group (2) have to turn around at the right end of T, arcs in group (5) at the left end, and arcs in groups (3) and (6) at both ends. According to our assumption, exactly K arcs have to turn around at the right end, and exactly K arcs at the left end. For each problematic arc c, we can therefore fix a value R_c, if it has to turn around at the right end, and a value L_c, if it has to turn around at the left end, such that two distinct arcs turning around at the same end of T have different values for R_c (or L_c), and such that $R_c, L_c \in \{1, \ldots, K\}$. Intuitively, R_c [L_c] determines which of the K attached binary trees at the right [left] end of T the paths corresponding to c will use for the turn-around. The following table shows which corresponding paths we introduce for each problematic arc:

arc c in	condition	paths
group (2)	$a_i \leq \frac{m}{2}, \quad b_i > \frac{m}{2}$	$(v_{a_i}, r^2_{R_c}), \quad (r^3_{R_c}, v_{m+1-b_i})$
group (3)	$b_i < a_i \leq \frac{m}{2}$	$(v_{a_i}, r^2_{R_c}), \quad (r^3_{R_c}, l^2_{L_c}), \quad (l^3_{L_c}, v_{b_i})$
group (5)	$a_i > \frac{m}{2}, \quad b_i \leq \frac{m}{2}$	$(v_{m+1-a_i}, l^2_{L_c}), \quad (l^3_{L_c}, v_{b_i})$
group (6)	$a_i > b_i > \frac{m}{2}$	$(v_{m+1-a_i}, l^2_{L_c}), \quad (l^3_{L_c}, r^2_{R_c}), \quad (r^3_{R_c}, v_{m+1-b_i})$

In addition, for each $i, 1 \leq i \leq K$, we add $K-1$ paths (l_i^3, l_i^2) and $K-1$ paths (r_i^3, r_i^2). These paths act as blockers. They make sure that all paths corresponding to a problematic arc are assigned the same color in any K-coloring. In Figure 3.3, only one of the $K-1$ blockers for a pair of paths corresponding to the same arc is sketched.

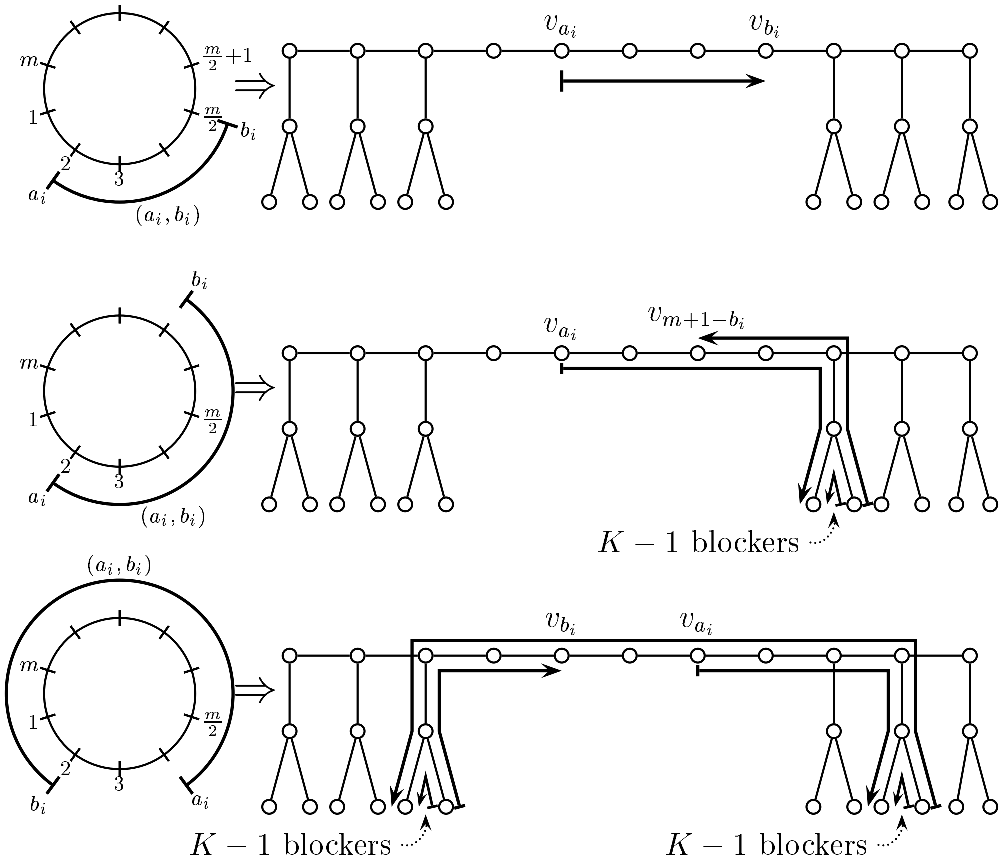


Figure 3.3: Arcs and corresponding paths

Summing up, we have one, two or three corresponding paths for each arc, and $2K(K-1)$ additional blockers. These paths make up the set P of paths for instance I' of the path coloring problem. Obviously, I' can be constructed in polynomial time. We will show that the paths in I' can be colored with K colors if and only if the arcs in I can be colored with K colors. This is a consequence of the following facts:

(a) any coloring with at most K colors for I' uses exactly K colors, and all paths corresponding to the same arc are assigned the same color

(b) if two arcs c_1 and c_2 do not intersect, their corresponding paths do not intersect either (and can be assigned the same color)

(c) if two arcs c_1 and c_2 intersect, there is a conflict between at least one path corresponding to c_1 and one path corresponding to c_2

(d) if the two paths meeting at an attached binary tree are assigned the same color, the $K-1$ blockers of that binary tree can be assigned the $K-1$ other colors arbitrarily

Each of the edges (r_i^1, r_i^2) is used by K paths in the same direction, therefore any coloring for the paths in I' must use at least K colors. Since $K-1$ colors are required for the $K-1$ blockers (r_i^3, r_i^2), there is only one color left, in any coloring with K colors, for the two remaining paths using edges (r_i^1, r_i^2) and (r_i^3, r_i^1). Hence, these two paths must be assigned the same color in such a coloring. Analogous reasoning can be applied for the paths meeting in a binary tree on the left side of T. This shows that fact (a) holds. Fact (d) is obviously true, because each attached binary tree "sees" only the $K-1$ blockers and the two paths meeting in that binary tree.

Facts (b) and (c) are also easy to see. Just observe the following: An arc has a corresponding path that uses the edge (v_i, v_{i+1}) if and only if the arc spans point $i+1$. An arc has a corresponding path that uses the edge (v_{i+1}, v_i) if and only if the arc spans point $m+1-i$. A path (except for a blocker) that uses any edge to the right of $v_{\frac{m}{2}}$ corresponds to an arc that spans point $\frac{m}{2}+1$. A path (except for a blocker) that uses any edge to the left of v_1 corresponds to an arc that spans point 1. In addition, if an arc spans 1 or $\frac{m}{2}+1$, it has corresponding paths using edge (l_1, v_1) and (v_1, l_1) or edge $(v_{\frac{m}{2}}, r_1)$ and $(r_1, v_{\frac{m}{2}})$, respectively. Therefore, if two arcs c_1 and c_2 have corresponding paths touching the left [right] part of T, at least one of the corresponding paths of c_1 is in conflict with a corresponding path of c_2. Furthermore, it is clear that all paths corresponding to the same arc can be assigned the same color.

Facts (a) to (d) show that there is a one-to-one correspondence between K-colorings of the arcs in I and K-colorings of the paths in I'. Hence, deciding whether a K-coloring for the paths in T exists is $\mathcal{NP}$-complete. $\square$

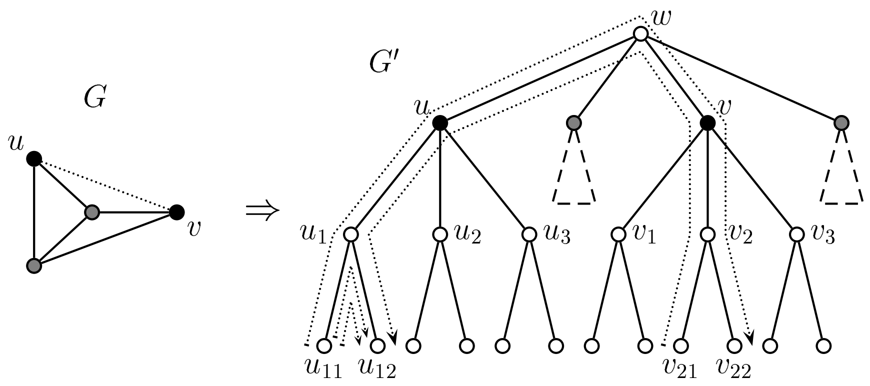


Figure 3.4: Reduction from EDGE-COLORING

Consequently, we cannot hope for an optimal polynomial-time algorithm for path coloring in bidirected trees even in the binary case unless $\mathcal{P} = \mathcal{NP}$.

A Lower Bound on the Achievable Approximation Ratio

The following theorem gives a lower bound on the best absolute approximation ratio that can be achieved by a polynomial-time algorithm if $\mathcal{P} \neq \mathcal{NP}$.

Theorem 3.1.10 *Deciding whether a set of paths in a bidirected tree network of arbitrary degree can be colored with* 3 *colors is* $\mathcal{NP}$*-complete. Therefore, no polynomial-time approximation algorithm with approximation ratio smaller than* 4/3 *exists unless* $\mathcal{P} = \mathcal{NP}$.

Proof: (by reduction from EDGE-COLORING)
It is $\mathcal{NP}$-complete to decide whether a given regular graph $G = (V, E)$ with degree three can be properly edge-colored with three colors [Hol81]. We show how to transform any 3-regular graph G into an instance of the path coloring problem in a bidirected tree network $G' = (V', E')$ such that the paths can be colored with 3 colors if and only if G can be properly edge-colored with 3 colors. Here, V' contains all nodes $v \in V$ and nine additional nodes $v_1, v_2, v_3, v_{11}, v_{12}, v_{21}, v_{22}, v_{31}, v_{32}$ for each $v \in V$, as well as a new node w. The edges in E' are such that w is the root of G', $v \in V$ are the children of w, v_1, v_2, v_3 are the children of v, and v_{i1}, v_{i2} are the children of v_i:

$$\begin{aligned} V' &= \{w\} \cup \{\, v, v_1, v_2, v_3, v_{11}, v_{12}, v_{21}, v_{22}, v_{31}, v_{32} \mid v \in V \,\} \\ E' &= \{\, \{w, v\}, \{v, v_i\}, \{v_i, v_{ij}\} \mid v \in V, i \in \{1, 2, 3\}, j \in \{1, 2\} \,\} \end{aligned}$$

The set P of paths contains four paths for each edge $e = \{u, v\} \in E$, namely $p_1^e = (u_{i1}, v_{j2})$, $p_2^e = (v_{j1}, u_{i2})$, and $p_3^e = p_4^e = (u_{i1}, u_{i2})$. (The node u

can be chosen arbitrarily among the two nodes joined by the undirected edge e.) The values of i and j are selected from $\{1, 2, 3\}$ such that a different value of i [of j] is chosen for each edge incident to u [incident to v]. Intuitively, p_1^e and p_2^e play the same role as a single undirected path in the proof of Theorem 3.1.4, and the other two paths p_3^e and p_4^e are merely blockers that make sure that p_1^e and p_2^e are assigned the same color in any 3-coloring of P. A 3-regular graph and part of the resulting instance of the path coloring problem are sketched in Figure 3.4. The nodes of the graph G on the left-hand side correspond to the children of the root of the bidirected tree G' on the right-hand side (only the underlying undirected tree of G' is drawn). The black nodes in G correspond to the black children of the root in G'. The dotted edge between the black nodes in G corresponds to the four dotted paths indicated in G'. The subtrees rooted at the children of the root of G' are shown only for the two relevant nodes.

Now we show that there is a 3-coloring of P in G' if and only if a proper edge coloring with 3 colors exists for G. Since the decision version of path coloring in bidirected trees is obviously in $\mathcal{NP}$, this implies that it is $\mathcal{NP}$-complete.

Assume that we have a 3-coloring of G. For each edge $e \in E$, we assign to the paths p_1^e and p_2^e the color of edge e. The paths p_3^e and p_4^e are assigned the two remaining colors that are still available. Obviously, this yields a valid 3-coloring for P in G'.

In the other direction, assume that there is a 3-coloring for P in G'. The blockers make sure that in such a coloring p_1^e and p_2^e (for each $e \in E$) are assigned the same color. No path p_1^f with $e \neq f \in E$ and $f \cap e \neq \emptyset$ can be assigned this color. As a consequence, if we assign each edge $e \in E$ the color of the path p_1^e, we obtain a proper 3-coloring for the edges of G.

An approximation algorithm with absolute approximation ratio smaller than $4/3$ could be used to decide whether a given set of paths can be colored with 3 colors or not. Using the reduction above, this algorithm could be used to decide whether a given 3-regular graph can be properly edge-colored with 3 colors, which is $\mathcal{NP}$-complete according to [Hol81]. Since the construction of G' and P can be done in polynomial time, the theorem follows. $\square$

Bounding the Number of Paths Touching a Single Node

In the following, we examine the case when the number of paths touching a single node of the tree is bounded by a constant. Recall that L denotes the maximum load of the given paths. Observe that L is also bounded by a constant (the same constant) if the number of paths touching a single node is bounded by a constant.

We will show that the path coloring problem in bidirected trees can be solved optimally in polynomial time if the number of paths touching a single node of the tree is bounded by a constant. A similar proof can establish this fact for path coloring in undirected trees as well, but there this result already follows from Theorems 3.1.2, 3.1.4, and 3.1.7: If the number of paths touching a node is bounded, then also the number of non-isolated nodes in the multigraph whose edge coloring corresponds to a coloring of the paths touching the node (Theorem 3.1.4) is bounded by a constant; thus the edge coloring can be computed in polynomial time (Theorem 3.1.7), giving an optimal local coloring, and the local colorings can be merged into a global optimal coloring (Theorem 3.1.2).

Theorem 3.1.11 *If $G = (V, E)$ is a bidirected tree network and P is a set of directed paths such that the number of paths touching a single node of G is bounded by a constant, an optimal coloring of P can be computed in polynomial time.*

Proof: We give a polynomial-time algorithm that decides, for any given $C \in \{L, L+1, \ldots, 2L\}$, whether P can be colored with C colors. If P can be colored with C colors, the algorithm computes such a coloring. Since P can always be colored using less than $2L$ colors [MKR95], we can then use linear or binary search to find the optimal coloring. This yields a polynomial-time algorithm.

Given a number $C \leq 2L$ of available colors, the algorithm to compute a C-coloring, if it exists, is as follows. First, it computes for every leaf v of G the set of all possible C-colorings of the paths touching v. Since both C and the number of paths touching v are bounded by a constant, this set contains only a constant number of colorings. Now the algorithm works its way to the (arbitrarily chosen) root of the tree (bottom-up). Each node v passes to its parent p the set S_v^p of all valid C-colorings for the paths touching both v and p. (Such a coloring is called valid if it can be extended to a C-coloring for all paths touching nodes in the subtree of G rooted at v.)

When the algorithm processes an internal node v of G, it computes the set S_v of all possible C-colorings of paths touching v. Again, there are only a constant number of such colorings. Then it checks, for every coloring $s \in S_v$, whether there is a compatible coloring $s' \in S_w^v$ for every child w of v; here, a coloring $s' \in S_w^v$ is considered compatible if the paths touching v and w are assigned the same color in s' and in s. If there is no such compatible coloring in at least one S_w^v, s is discarded from S_v. Finally, the paths that touch v but not the parent p of v are removed from all colorings remaining in S_v. The resulting set of colorings is the set S_v^p of valid C-colorings that is passed to the parent p of v.

When the algorithm processes the root node r, there are two possibilities. Either, all colorings in the set S_r must be discarded, in which case no C-coloring of the given paths exists, or at least one coloring remains in S_r, in which case the given set P of paths can be colored with at most C colors.

If we store at each node v of the tree which of the valid colorings passed from its children were found to be compatible with each valid coloring for the paths touching v, the same algorithm can be used to obtain a C-coloring for all paths. □

If the maximum load L and the degree of the nodes in G are both bounded by a constant, it follows that the number of paths touching a node of G is also bounded by a constant; hence, we get the following corollary.

Corollary 3.1.12 *Path coloring in a bidirected tree G can be solved optimally in polynomial time if both L and the degree of the nodes in G are bounded by a constant.*

3.2 Path Coloring in Ring Networks

The path coloring problem in ring networks is closely related to the problem of coloring circular-arc graphs (this latter problem was discussed at the beginning of Section 3.1.2). If each connection request has to specify which of the two alternative routes in the ring it takes, the resulting fixed path coloring problem is equivalent to ARC-COLORING. Since we have the additional freedom of choosing one of two possible routings for each connection request, however, it is not immediately clear whether the path coloring problem for ring networks is $\mathcal{NP}$-hard. Nevertheless, this can be shown by a reduction from ARC-COLORING.

Theorem 3.2.1 *Path coloring is $\mathcal{NP}$-hard for undirected and for bidirected ring networks.*

Proof: (by reduction from ARC-COLORING)
Like in the proof of Theorem 3.1.9, let an instance I of the ARC-COLORING problem be given by a family $F = \{A_1, \ldots, A_n\}$ of circular arcs and a positive integer K, where each arc $A_i \in F$ is a pair (a_i, b_i) with $a_i, b_i \in \{1, \ldots, m\}$. Assume again that $m \leq 2n$ and that each point i, $1 \leq i \leq m$, is contained in the span of exactly K arcs.

First, we transform I into an instance I' of the ARC-COLORING problem that can be colored with K colors if and only if I can be colored with K colors. Whereas I can contain arcs that span more than half of the circle,

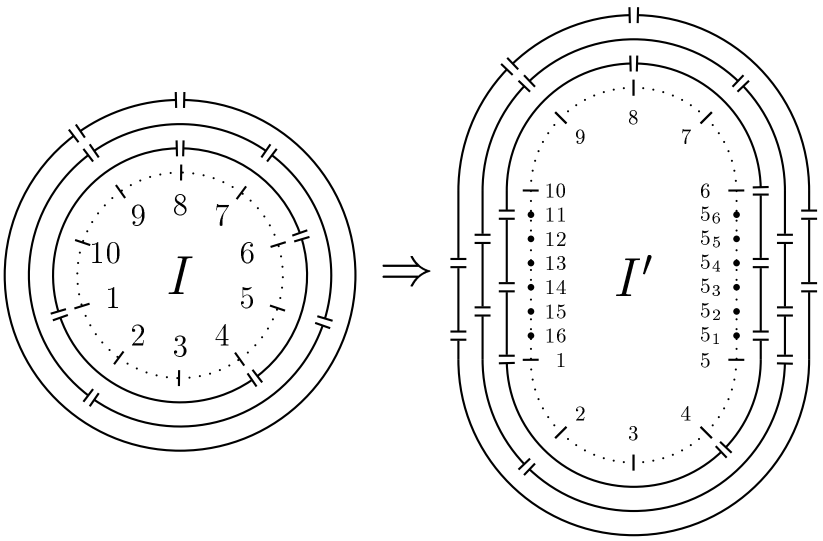


Figure 3.5: Instances I and I' of the ARC-COLORING problem

the length of all arcs in I' will be strictly smaller than half the circumference of the (slightly larger) circle. Afterwards, we will prove that the instance I', if viewed as an instance I'' of the path coloring problem, can be colored with K colors if and only if I can be colored with K colors.

In order to transform I into I', we introduce $4K$ new points on the circle, $2K$ points between m and 1 and $2K$ points between $\lfloor m/2 \rfloor$ and $\lfloor m/2 \rfloor + 1$. We number the new points between m and 1 by $m+1, m+2, \ldots, m+2K$. By our assumption, there are exactly K arcs $B_1, \ldots, B_K$ spanning point 1. We replace each such arc $B_i = (u_i, v_i)$ by three new arcs $B_i^1 = (u_i, m+i)$, $B_i^2 = (m+i, m+K+i)$ and $B_i^3 = (m+K+i, v_i)$. Each of the points $m+1, \ldots, m+2K$ is an endpoint of exactly two arcs. Every two arcs that share such an endpoint must necessarily be assigned the same color in any proper K-coloring for the new family of arcs because they intersect the same $K-1$ other arcs. Hence, B_i^1, B_i^2 and B_i^3 must be assigned the same color in such a coloring, and any K-coloring for the modified instance corresponds to a K-coloring for I and vice versa.

The insertion of $2K$ new points between $\lfloor m/2 \rfloor$ and $\lfloor m/2 \rfloor + 1$ works the same way as between m and 1. Here, we replace each arc C_i from a set $\{C_1, \ldots, C_K\}$ by three new arcs C_i^1, C_i^2 and C_i^3. Having inserted $4K$ new points altogether, we obtain the instance I' of the ARC-COLORING problem. It is clear that the transformation can be done in polynomial time and that there is a K-coloring for I' if and only if there is a K-coloring for I.

Figure 3.5 shows an instance I of the ARC-COLORING problem with $K = 3$ and the instance I' obtained from I by inserting $4K = 12$ new points.

Now we interpret I' as an instance I'' of the path coloring problem in an undirected ring network. (We will show later how to obtain an instance of the path coloring problem in a bidirected ring.) Each arc (a_i, b_i) is viewed as a connection request between nodes a_i and b_i on a ring network with $m+4K$ nodes. We claim that there is a routing and K-coloring for I'' if and only if there is a proper K-coloring for I'. First, it is possible to generate a routing and K-coloring for I'' from a K-coloring for I' by routing each connection request in the direction indicated by the corresponding arc and assigning it the color of that arc. Second, a routing and K-coloring for I'' also gives a K-coloring for I'. This follows because all connection requests in I'' must be routed the short way in order to allow a K-coloring for the resulting paths, and because the corresponding arc of a connection request corresponds to the short route for this connection request.

Note that every connection request in I'' uses strictly more edges if routed the long way ($\geq m/2+2K+1$ edges, if m is even, and $\geq (m+1)/2+2K$ edges, if m is odd) than it uses if routed the short way ($\leq m/2+2K-1$ edges, if m is even, and $\leq (m-1)/2+2K$ edges, if m is odd). If each connection request is routed the short way and the resulting paths are colored with K colors, every edge of the ring is used by K paths with different colors. Therefore, if at least one connection request is routed the long way, the coloring of the paths must use strictly more than K colors.

To obtain a K-coloring for I from a routing and K-coloring for I'', we simply take the K-coloring for I' which we get immediately and derive a K-coloring for I by coloring each arc of I with the color of all corresponding arcs in I'. (We already showed that these arcs are colored with the same color in any K-coloring for I'.) Given a particular arc in I, note that we have up to five corresponding arcs in I', which are viewed as connection requests in I''. These connection requests must be assigned the same color, if all connection requests in I'' are routed and colored with K colors.

Finally, we explain how an instance I'' of the path coloring problem in a bidirected ring can also be obtained from I'. For this purpose, it suffices to substitute two connection requests (u, v) and (v, u) for each arc (u, v) of I'. In any routing and K-coloring for I'', all connection requests must again be routed the short way. (This follows from the same argument as above.) Such a routing and K-coloring actually consists of two independent colorings, one for the connection requests routed in clockwise direction and one for those routed in counterclockwise direction. Each of these two colorings contains exactly one of the two connection requests corresponding to an arc of I', and can therefore be used to obtain a K-coloring for I'.

Similarly, any K-coloring for I' can be used to obtain a routing and K-coloring for I'': assign both connection requests corresponding to an arc the color assigned to that arc in the K-coloring of I', and route all connection requests the short way. □

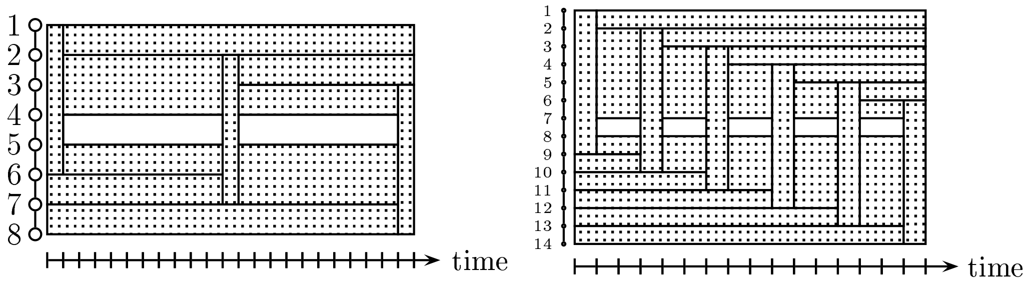
Interestingly, a 2-approximation algorithm for path coloring in undirected ring networks is obtained by simply ignoring some arbitrary edge e of the ring [RU94]. The connection requests on the resulting chain graph can be colored optimally by coloring the conflict graph, which is an interval graph. An optimal coloring for the paths on the chain graph uses at most twice as many colors as an optimal coloring on the ring. This follows from the fact that given an optimal routing and coloring with C^* colors on the ring, one can route the connection requests using e the other way and assign them colors in the range from $C^* + 1$ to $2C^*$, thereby obtaining a coloring of the paths in the chain graph using at most $2C^*$ colors. The optimal coloring of the paths in the chain graph uses no more colors than this.

A 2-approximation algorithm for path coloring in bidirected ring networks can be obtained the same way as in undirected ring networks [MKR95]. Removing an arbitrary edge from the given ring leads to two independent path coloring problems in separate chain networks. These can be solved optimally in polynomial time using an interval graph coloring algorithm. The resulting routing and coloring of the connection requests in the ring network uses at most twice as many colors as an optimal routing and coloring.

It seems surprisingly difficult to devise approximation algorithms with approximation ratio better than 2 for path coloring in rings. Recently, however, a randomized approximation algorithm for path coloring in undirected rings achieving approximation ratio $1.5 + 1/2e + o(1)$ with high probability under certain conditions was presented [Kum98]. For the ARC-COLORING problem, a (5/3)-approximation algorithm was given in [SH90]. A randomized approximation algorithm for ARC-COLORING achieving approximation ratio $1 + 1/e + o(1)$ with high probability can be found in [Kum98]. Apart from that, better deterministic approximation algorithms for ARC-COLORING are known only for special classes of circular-arc graphs [Tuc75, OBB81].

3.3 Arbitrary Duration and Bandwidth

The previous sections in this chapter have dealt with the complexity of the path coloring problem in tree and ring networks. Recall that the path coloring problem is equivalent to the call-scheduling problem with unit call duration, unit bandwidth requirements, and unit edge capacities. In this section we investigate the more general call-scheduling problem with either arbitrary call



(a) Reduction from PARTITION (b) Reduction from 3-PARTITION

Figure 3.6: Proving $\mathcal{NP}$-hardness for call scheduling with arbitrary duration

duration or arbitrary bandwidth requirements. We assume that the capacity of all edges in the network is the same and that this capacity is normalized to 1.

Once we allow either arbitrary call duration or arbitrary bandwidth requirements, call scheduling becomes $\mathcal{NP}$-hard for virtually every network topology. If arbitrary bandwidth requirements are allowed, a call can request any bandwidth b, $0 < b \leq 1$. Similarly, arbitrary call duration means that the duration of a call can be any natural number (we assume without loss of generality that the duration is given as a multiple of a fixed time quantum). As the $\mathcal{NP}$-hardness proofs are not difficult or follow directly from known results, we sketch only the basic ideas here. All results hold for the call-scheduling problem both in undirected and in bidirected graphs.

An instance of the $\mathcal{NP}$-complete problem PARTITION [GJ79] is given by a finite set A with sizes $s(a) \in \mathbb{N}$ for each $a \in A$ such that $\sum_{a\in A} s(a) = 2B$ is even. The question is whether there is a subset $A' \subseteq A$ such that $\sum_{a\in A'} s(a) = \sum_{a\in A\setminus A'} s(a) = B$. In other words, the question is whether A can be partitioned into two equal-size subsets.

An instance of the strongly $\mathcal{NP}$-complete problem 3-PARTITION [GJ79] is given by a finite set A of $3m$ elements, a positive integer B, and a size $s(a) \in \mathbb{N}$ satisfying $B/4 < s(a) < B/2$ for each $a \in A$ such that $\sum_{a\in A} s(a) = mB$. Here, the question is whether A can be partitioned into m disjoint sets such that the sum of the item sizes in each set is equal to B.

Once arbitrary call duration is allowed, call scheduling in chain networks becomes equivalent to the one-dimensional layout compaction problem, whose decision version is known to be strongly $\mathcal{NP}$-complete [DL87]. The proof of $\mathcal{NP}$-completeness can be adapted to call scheduling immediately. Given an instance of the PARTITION or 3-PARTITION problem, a set of frame-calls can be specified (shown dotted in Figure 3.6) that can be scheduled optimally only such that one particular edge of the chain network remains idle during separate time intervals of length B. In addition, it is possible to generate a call of duration $s(a)$ for each $a \in A$ that uses only that idle edge. These calls can be scheduled within the gaps left by the frame-calls if and only if A can be partitioned into two (in the case of PARTITION, see Figure 3.6(a)) or m (in the case of 3-PARTITION, see Figure 3.6(b)) equal-size subsets. Therefore, we have $\mathcal{NP}$-hardness for chain networks with at least 8 nodes and strong $\mathcal{NP}$-hardness for arbitrary chain networks.

Another reduction from 3-PARTITION shows that call scheduling with unit bandwidth requirements and arbitrary duration is strongly $\mathcal{NP}$-hard in a star with degree 3; this follows from the proof of strong $\mathcal{NP}$-hardness for scheduling of multiprocessor tasks with prespecified processor allocations on three processors in [HvdVV94, Theorem 2.3]. Therefore, call scheduling with unit bandwidth requirements and arbitrary duration is strongly $\mathcal{NP}$-hard in any tree containing a node of degree at least 3.

If we have unit call duration but allow arbitrary bandwidth requirements, call scheduling is already $\mathcal{NP}$-hard for a chain network consisting of a single link, because it is equivalent to the bin-packing problem. In particular, PARTITION and 3-PARTITION can be reduced to the call-scheduling problem with arbitrary bandwidth requirements on a single link, implying also that the decision problem of the latter is strongly $\mathcal{NP}$-complete.

Finally, we consider networks that contain two nodes u and v with at least two edge-disjoint paths connecting u and v. Note that the maximum number k of edge-disjoint paths between two nodes u and v can be computed in polynomial time with a maximum flow algorithm (see [AMO93]). Furthermore, this computation gives also a minimum u-v-cut, i.e., a set of k edges such that every path from u to v uses at least one of these k edges.

In a network G with k edge-disjoint paths between u and v, call scheduling is $\mathcal{NP}$-hard for calls with unit duration and arbitrary bandwidth requirements and for calls with arbitrary duration and unit bandwidth requirements. Given an instance of PARTITION, a call from u to v with bandwidth requirement $s(a)/B$ or duration $s(a)$ is generated for each $a \in A$, and $k-2$ additional calls that ensure that the former calls use exactly two of the edges in the minimum u-v-cut. Hence, these calls must be partitioned into two equal-size subsets in order to obtain a schedule with makespan 1 or with makespan B, respectively.

Chapter 4

Path Coloring in Trees

Given a set of paths in a tree, the path coloring problem is to assign colors to the paths such that paths receive different colors if they share an edge. The goal is to use as few different colors as possible. This problem models wavelength assignment in all-optical communication networks without wavelength converters.

In Chapter 3 it has been shown that path coloring is $\mathcal{NP}$-hard for undirected paths in undirected trees if the degree of the tree is arbitrary and for directed paths in bidirected trees even if the degree is bounded by three. In this chapter we present polynomial-time approximation algorithms for path coloring in undirected and bidirected trees of arbitrary degree.

In Section 4.1 we use a known approximation algorithm for edge coloring of multigraphs to obtain an approximation algorithm for path coloring in undirected trees with asymptotic approximation ratio 1.1. In Section 4.2 we present an approximation algorithm for path coloring in bidirected trees that uses at most $\lceil (5/3)L \rceil$ colors to color a given set of paths with maximum load L. The algorithm belongs to a class of local greedy algorithms, and it is known that every algorithm in this class uses at least $\lfloor (5/3)L \rfloor$ colors in the worst case, even for instances where an optimal coloring uses only L colors [Jan97]. Hence, the approximation algorithm from Section 4.2 is optimal in the class of local greedy algorithms with respect to the number of colors used in the worst case.

4.1 Path Coloring in Undirected Trees

Using a known result for edge coloring of multigraphs, we improve on a (3/2)-approximation from [RU94] and derive a polynomial-time approximation algorithm with absolute approximation ratio 4/3 and asymptotic approximation ratio 1.1 for path coloring in undirected trees.

Corollary 3.1.5 in Section 3.1.1 implies that approximation algorithms for edge coloring of multigraphs can be converted to approximation algorithms for path coloring in undirected trees with the same absolute and asymptotic approximation ratio.

In [NK90], Nishizeki and Kashiwagi presented an approximation algorithm for edge coloring that uses at most $\lfloor 1.1 \cdot \chi'(G) + 0.8 \rfloor$ colors for any multigraph G. Furthermore, if the edges of a multigraph G can be colored with one or two colors, an optimal coloring can be obtained in polynomial time (by coloring the line graph of G, which is a bipartite graph in this case). If $\chi'(G) \geq 3$, note that $\lfloor 1.1 \cdot \chi'(G) + 0.8 \rfloor \leq (4/3)\chi'(G)$. Therefore, there is a polynomial-time algorithm for edge coloring of multigraphs with absolute approximation ratio 4/3 and asymptotic approximation ratio 1.1. Using Corollary 3.1.5, this implies the following theorem.

Theorem 4.1.1 *There is a polynomial-time approximation algorithm that colors a given set P of undirected paths in an undirected tree using at most $OPT(P)$ colors if $OPT(P) \leq 2$ and at most $\lfloor 1.1 \cdot OPT(P) + 0.8 \rfloor$ colors otherwise. The algorithm has absolute approximation ratio* 4/3 *and asymptotic approximation ratio* 1.1.

Note that it is conjectured in [Hoc97, p. 394] that ultimately an approximation algorithm for edge coloring any multigraph G using at most $\chi'(G)+1$ colors will be found; such an algorithm would immediately give an approximation algorithm that colors a set P of paths in an undirected tree using at most $OPT(P) + 1$ colors.

4.2 Path Coloring in Bidirected Trees

For path coloring in undirected trees it is difficult to color the paths touching a single node, but it is easy to combine individual local colorings to obtain a global coloring without increasing the number of colors used. For path coloring in bidirected trees, it is easy to color the paths touching a single node, as will be seen later, but it is $\mathcal{NP}$-hard to combine these optimal local colorings into an optimal coloring of all paths. Hence, a different approach is required to obtain an approximation algorithm for path coloring in bidirected trees.

4.2.1 Outline of Algorithm

The input to the algorithm is a bidirected tree $T = (V, E)$ with N nodes and a set P of directed paths in T. L denotes the maximum load among all edges of T. We analyze the running-time of our algorithm in terms of the parameters N and L.

We will present a polynomial-time algorithm that produces a coloring of P using at most $\lceil (5/3)L \rceil$ colors. For now we assume that $L = 3\ell$ is a multiple of 3. As in the previous approaches to path coloring in bidirected trees [MKR95, KP96, KS97], our algorithm is a greedy algorithm. It processes the nodes of the given tree in depth-first search order (or any other order corresponding to a graph search procedure derived from the general procedure presented in Section 2.1.2), starting at an arbitrary leaf node. Initially, the paths touching the leaf node can easily be colored using L colors. When a node v is processed, all paths touching its parent (in the depth-first search tree) or any other node with smaller dfs-number have already been colored, and this coloring is now extended to include all paths touching v.

Note that this coloring-extension step at a node v could be carried out by considering the uncolored paths touching v one by one and simply assigning each path the smallest color still available for it. In fact, this *simple greedy algorithm* uses at most $2L - 1$ colors, as observed in [MKR95]. The number of colors used in the worst case can be improved by performing the coloring-extension step in a more careful manner.

During the whole path coloring process, our algorithm maintains the following two invariants:

Invariant 1: The number of colors used in total is at most $(5/3)L = 5\ell$.

Invariant 2: The number of colors used on a pair of directed edges with opposite directions is at most $(4/3)L = 4\ell$.

Initially, when only the paths touching one leaf node are colored, the invariants hold. We will show how a given coloring can be extended to include the paths touching an additional node without violating the invariants.

For comparison, note that the previously known path coloring algorithms have been obtained using weaker invariants: the algorithm from [MKR95] uses at most $(15/8)L$ colors in total and at most $(3/2)L$ colors on each pair of directed edges; the algorithms from [KP96] and [KS97] use at most $(7/4)L$ colors in total and at most $(3/2)L$ colors on each pair of directed edges.

We assume that each directed edge of the tree has load exactly L. (Otherwise, extra paths can be added to the given instance.) The problem of extending an existing partial coloring to include the paths touching a new

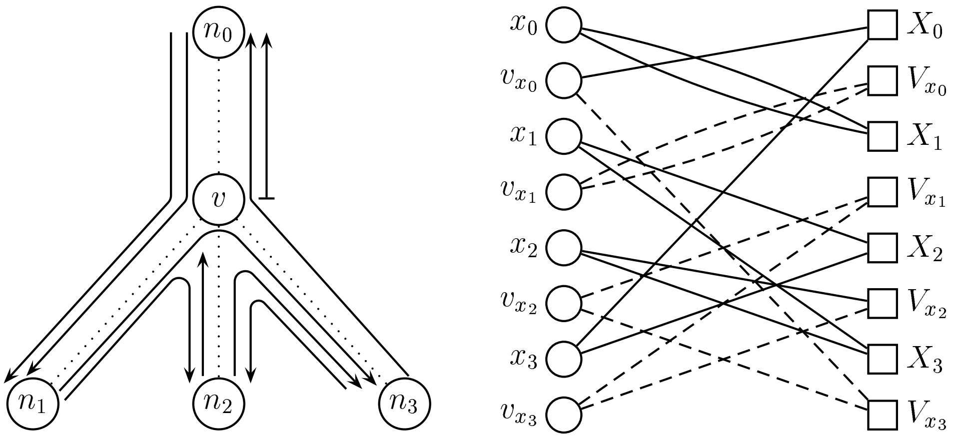


Figure 4.1: Construction of the bipartite graph G_v

node v is reduced to a constrained edge-coloring problem in a bipartite multigraph G_v. Denote by n_0 the parent of v in the dfs-tree and by $n_1, n_2, \ldots, n_k$ the children of v, where $k = \delta(v) - 1$. Note that all paths touching n_0 have already been colored, and that the goal is to color all paths touching v but not n_0 without violating the invariants. The bipartite graph G_v has left and right vertex set $\bigcup_{i=0}^{k}\{x_i, v_{x_i}\}$ and $\bigcup_{i=0}^{k}\{X_i, V_{x_i}\}$, respectively. Hence, G_v has $4\delta(v)$ vertices. Each path touching v contributes one edge to G_v as follows:

- A path running from n_i to n_j contributes an edge $\{x_i, X_j\}$.
- A path touching n_i and terminating at v contributes an edge $\{x_i, V_{x_i}\}$.
- A path starting at v and touching n_i contributes an edge $\{v_{x_i}, X_i\}$.

Note that all vertices x_i and X_i have degree L, whereas the vertices v_{x_i} and V_{x_i} may have smaller degree. In order to make G_v L-regular, dummy edges are added between vertices v_{x_i} on the left side and vertices V_{x_j} on the right side, if necessary. (More precisely, for every edge $\{x_i, X_j\}$ a dummy edge $\{v_{x_j}, V_{x_i}\}$ is added.) Figure 4.1 illustrates this construction, with dashed lines indicating dummy edges. Note that vertices in the same row of G_v (i.e., vertices that are opposite each other) cannot be adjacent.

It is easy to see that two paths touching v must be assigned different colors if and only if the corresponding edges in G_v share a vertex. Hence, any valid edge coloring of G_v constitutes a valid coloring for the paths touching v and vice versa. Since an optimal edge coloring in a bipartite multigraph can be computed in polynomial time [CH82, Sch98], this implies that the path coloring problem can be solved optimally in polynomial time for the set of paths touching a single node of a bidirected tree if no other constraint is present. In our case, however, the paths touching n_0 have been colored in a previous step, and thus the edges incident to x_0 and X_0 have already received a color. We refer to these edges as *pre-colored* edges. The colors that appear on pre-colored edges of G_v are called *single* colors if they appear only once, and *double* colors if they appear twice (i.e., on a pre-colored edge incident to x_0 and on a pre-colored edge incident to X_0). Denote by S the number of single colors, and by D the number of double colors. Invariant 2 ensures that $S + D \leq (4/3)L$. Since we assume that every edge has load L, we have $S + 2D = 2L$ and, consequently, $D \geq (2/3)L$.

We will show that the uncolored edges of G_v can be colored using at most $(1/3)L$ new colors (i.e., colors that do not appear on the pre-colored edges) such that no row of G_v sees more than $(4/3)L$ colors. (We say that a row *sees* a color c if an edge incident to a vertex of that row is colored with c. Similarly, we say that a vertex *sees* c if an incident edge is colored c. Two opposite vertices *share* a color c if both vertices have an incident edge colored c.) This implies that both invariants hold at the end of the current edge-coloring phase. Note that the new colors can actually be colors that have been used to color paths touching previous nodes, provided they do not appear on pre-colored edges. Furthermore, the running-time of our algorithm for this constrained bipartite edge-coloring problem will be of the same order as the running-time of the best algorithm for unconstrained bipartite edge coloring.

We claim that we can assume that D is *exactly* $(2/3)L$ without loss of generality. The reason is that if D is greater than $(2/3)L$, one can simply "split" an appropriate number of double colors by assigning one of the two pre-colored edges colored with the same double color a new color for the duration of this edge-coloring phase. The time required for splitting the double colors is $O(L)$. The following lemma shows that it is possible to restore a valid coloring afterwards in linear time.

Lemma 4.2.1 *Denote the original number of single and double colors on the pre-colored edges of G_v by S and D, respectively. Assume that $D > (2/3)L$ and that an edge coloring for G_v with $\leq (4/3)L$ colors per row and $\leq (5/3)L$ colors altogether has been found after splitting $s = D - (2/3)L$ double colors. Then the original colors on the pre-colored edges can be re-established without violating the invariants in time linear in the size of G_v.*

Proof: Denote by $d_1, d_2, \ldots, d_s$ the double colors that were split for the duration of the edge-coloring phase. Assume that double color d_i was split by assigning a new color n_i to the edge e_i that was previously colored d_i and

$e \leftarrow e_i$; $u \leftarrow$ left endpoint of e;
$c_1 \leftarrow d_i$; $c_2 \leftarrow n_i$;
repeat
  change color of e from c_2 to c_1;
  if $\exists$ edge $e' = \{u, u'\} \neq e$ incident to u colored c_1 **then**
    $e \leftarrow e'$; $u \leftarrow u'$; $swap(c_1, c_2)$;
  else if $C(\bar{u}) = \{c_2\}$ **then**
    $e \leftarrow$ edge $\{\bar{u}, \bar{u}'\}$ incident to $\bar{u}$ colored c_2;
    $u \leftarrow \bar{u}'$;
  else exit;
  fi
until *false*;

Figure 4.2: Algorithm for re-establishing color d_i on edge e_i

that is incident to X_0 and a left vertex u_i. The algorithm for re-establishing the original colors on the pre-colored edges handles each edge e_i in turn. The problem is that changing the color of edge e_i from n_i back to d_i may either result in more than $(4/3)L$ colors in the row containing vertex u_i, or it may cause two edges incident to u_i being colored with the same color d_i. Therefore, the algorithm must in general change the color of more than one edge.

Consider the subgraph G_v^i of G_v that contains only the edges colored d_i and n_i. Every vertex in G_v^i has degree at most two. Hence, G_v^i is a collection of paths and cycles whose edges are colored alternately with d_i and n_i. In particular, x_0 and X_0 have degree one and are endpoints of paths. The edge incident to x_0 is colored d_i, the edge incident to X_0 is colored n_i. Our algorithm (see Figure 4.2) starts by exchanging colors d_i and n_i on the path starting at X_0. Assume that this path ends at a certain vertex u, and that the color of the edge incident to u was changed from c_2 to c_1, where $\{c_1, c_2\} = \{d_i, n_i\}$. The vertex opposite u is denoted by $\bar{u}$, and the set of colors that are assigned to edges incident to $\bar{u}$ is denoted by $C(\bar{u})$. If $\bar{u}$ has degree one and if its incident edge is colored with c_2, the number of colors in the row of u and $\bar{u}$ has increased by one. In this case, the algorithm exchanges colors d_i and n_i on the path starting at $\bar{u}$ as well. Otherwise, the algorithm terminates.

It remains to show that the algorithm always terminates and that the coloring it produces satisfies the requirements. Obviously, the algorithm recolors only paths in G_v^i and never starts to recolor a cycle. Hence, the only possibility for the algorithm not to terminate is to get stuck in a cycle of paths. (We call paths $p_0, p_1, \ldots, p_{k-1}$ in G_v^i a *cycle of paths* if an endpoint of p_i is opposite an endpoint of $p_{i+1 \bmod k}$ for $0 \leq i \leq k-1$ and if $p_i \neq p_j$ for $i \neq j$.) Furthermore, it is clear that such a cycle of paths must contain the paths starting at x_0 and X_0, because otherwise the algorithm wouldn't have recolored any path of the cycle in the first place. Even if the paths starting at x_0 and X_0 are part of the same cycle of paths, however, we can show that the algorithm will not recolor the path starting at x_0 and, therefore, will not get stuck in this cycle of paths.

Assume to the contrary that it is possible to reach x_0 from X_0 by traversing paths in the same order as the algorithm in Figure 4.2. Such a traversal always passes an edge colored n_i from a right vertex to a left vertex, and an edge colored d_i from a left vertex to a right vertex. Hence, the last edge it traverses before it reaches x_0 must be colored n_i, which contradicts the fact that the only edge incident to x_0 is colored d_i.

In particular, this shows that the algorithm never recolors the edge incident to x_0 in G_v^i. Hence, the original colors on the pre-colored edges are re-established. Furthermore, the algorithm does not increase the number of colors in any other row of G_v. This follows because colors on two incident edges are simply exchanged for inner vertices of recolored paths, and because the path starting at the vertex opposite the endpoint of the previously recolored path is also recolored in case the number of colors in the current row would be increased otherwise.

It is easy to see that the running-time for re-establishing the original colors on the pre-colored edges is linear in the size of G_v, i.e., it can be bounded by $O(\delta(v)L)$. □

So now we assume that $S = (2/3)L$ and $D = (2/3)L$. Since the bipartite graph G_v is L-regular, its edges can be partitioned into L disjoint perfect matchings. Such a partitioning can be obtained using an arbitrary algorithm for edge coloring of bipartite multigraphs. Hence, the time requirement for computing the L perfect matchings is $O(T_{\text{ec}}(\delta(v), L))$. Each perfect matching is classified according to the colors on its two pre-colored edges [KP96]. Matchings with two single colors are called SS-matchings. Matchings with a single color and a double color are called ST-matchings. Matchings with two different double colors are called TT-matchings. A matching with the same double color on both pre-colored edges is a PP-matching.

We obtain 3-regular subgraphs, each containing two single colors and four (not necessarily distinct) double colors, by partitioning the L matchings into *triplets*, i.e., groups of 3 matchings, in an appropriate way. The time requirement for this partitioning into triplets will be $O(L + \delta(v))$. The uncolored edges of each triplet H can then be colored using at most one new color and reusing some of the old colors such that no row of H (except the top row with vertices x_0 and X_0) sees more than 4 colors. Hence, at most $D/2 = (1/3)L$ new colors are used altogether, and no row sees more than $(4/3)L$ colors. The running-time for coloring the uncolored edges of a triplet is linear in the size of the triplet, i.e., $O(\delta(v))$. As there are $L/3$ triplets, the running-time for coloring them is $O(\delta(v)L)$ altogether. Hence, the running-time for the coloring-extension substep is dominated by the time for partitioning the edges of G_v into L perfect matchings and amounts to $O(T_{\mathrm{ec}}(\delta(v), L))$. The details of the partitioning and edge coloring are presented in the following sections.

4.2.2 Partitioning and Coloring Strategy

Kumar and Schwabe introduce a very helpful concept that allows to color a special kind of 3-regular subgraph of G_v using at most one new color such that every row except the top row sees at most 4 colors [KS97].[1] Precisely speaking, they consider 3-regular subgraphs H of G_v with the property that of the six pre-colored edges of H, two are colored with the same *preserved* double color d, two are colored with (possibly distinct) double colors, and the remaining two are colored with single colors s and s'. In addition, they assume that the edges colored s and s' are not incident to the same vertex out of $\{x_0, X_0\}$. We refer to such subgraphs of G_v as *KS-subgraphs*. Furthermore, they call a subgraph H_1 of G_v a *gadget* if x_0 and X_0 have degree 3 in H_1 while all other vertices have degree 2. Their important result concerning KS-subgraphs can be stated as the following lemma.

Lemma 4.2.2 (Kumar and Schwabe, 1997) *Let H be a KS-subgraph of G_v with preserved double color d and single colors s and s'. If H can be partitioned into a gadget H_1 and a matching H_2 on all vertices other than x_0 and X_0, then H can be colored using colors d, s, s', and at most one new color such that no row of H except the top row sees more than 4 colors.*

The proof of this lemma is of a similar nature as the proof we will give for Lemma 4.2.4 shortly. In particular, the running-time for computing the required coloring is linear in the size of the KS-subgraph H.

Kumar and Schwabe obtain their $(7/4)L$ result by extracting and coloring KS-subgraphs until the ratio of remaining edges with double colors to edges with single colors drops from $1:1$ (they assume $(3/2)L$ colors per link, and hence $S = 2D$) to $1:2$ or lower. They have to treat the remaining uncolored edges in a rather complicated way, however, and require 3 colors in a row for every two matchings considered in the worst case.

[1] The author would like to thank Vijay Kumar for supplying a preliminary full version of [KS97] and for helpful discussions.

We improve on the $(7/4)L$ result by Kumar and Schwabe by showing how the entire graph G_v can be partitioned into triplets each of which can be colored with one new color and with at most 4 colors per row.

If color a appears on an edge e incident to x_0 in G_v, we denote the other endpoint of e by $r(a)$. Similarly, if color b appears on an edge f incident to X_0, we denote the other endpoint of f by $l(b)$.

First, we generalize Lemma 4.2.2 to the case where the triplet cannot be partitioned into gadget and matching.

Lemma 4.2.3 *Let H be a KS-subgraph of G_v with preserved double color d and single colors s and s'. If no row of H except the top row sees more than 4 colors on pre-colored edges, then the uncolored edges of H can be colored in time $O(\delta(v))$ using colors d, s, s', and at most one new color such that no row of H except the top row sees more than 4 colors.*

Proof: Denote the colors on edges incident to x_0 by d, d_1, and s, and the colors on edges incident to X_0 by d, d_2, and s'. Note that d_1 may be equal to d_2. First, we try to partition H into gadget and matching. For this purpose, remove vertices x_0 and X_0 from H and introduce dummy edges $\{l(d), r(d)\}$, $\{l(s'), r(s)\}$ and $\{l(d_2), r(d_1)\}$. The obtained graph H' is again 3-regular, and its edges can be split into three perfect matchings M_1, M_2, and M_3 in linear time using Schrijver's algorithm [Sch98]. If one of the three matchings, say M_1, does not contain a dummy edge, then H can be partitioned into the gadget $H \setminus M_1$ and the matching M_1, and the coloring method from Lemma 4.2.2, which takes time linear in the size of H, is applicable.

Otherwise, each matching contains exactly one dummy edge, and the matching containing the dummy edge $\{l(d), r(d)\}$ immediately gives a PP-matching M in H. If the double colors d_1 and d_2 are equal, the matching containing the dummy edge $\{l(d_2), r(d_1)\}$ gives a second PP-matching, the matching containing the dummy edge $\{l(s'), r(s)\}$ gives an SS-matching, and the triplet can be colored as required by coloring the uncolored edges of the first PP-matching by d, those of the second PP-matching by d_1, and those of the SS-matching by s. No new color is required. Therefore, we assume $d_1 \neq d_2$ from now on.

H can be partitioned into the PP-matching M with color d and a cycle cover $C = H \setminus M$. The uncolored edges of M can be colored with color d. If the uncolored edges of the cycle cover C can be colored using s, s', and a new color n such that no row (except the top row) sees more than 3 colors in the cycle cover, we have obtained the desired coloring: since each row sees only one color in the PP-matching and at most 3 colors in the cycle cover, the triplet is colored using at most 4 colors per row. In the remainder of this

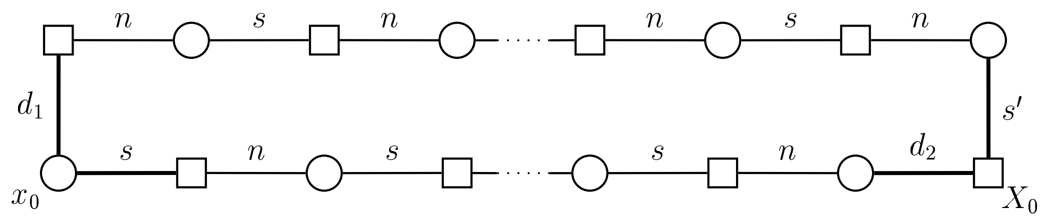


Figure 4.3: Case 1: x_0 and X_0 are contained in the same cycle

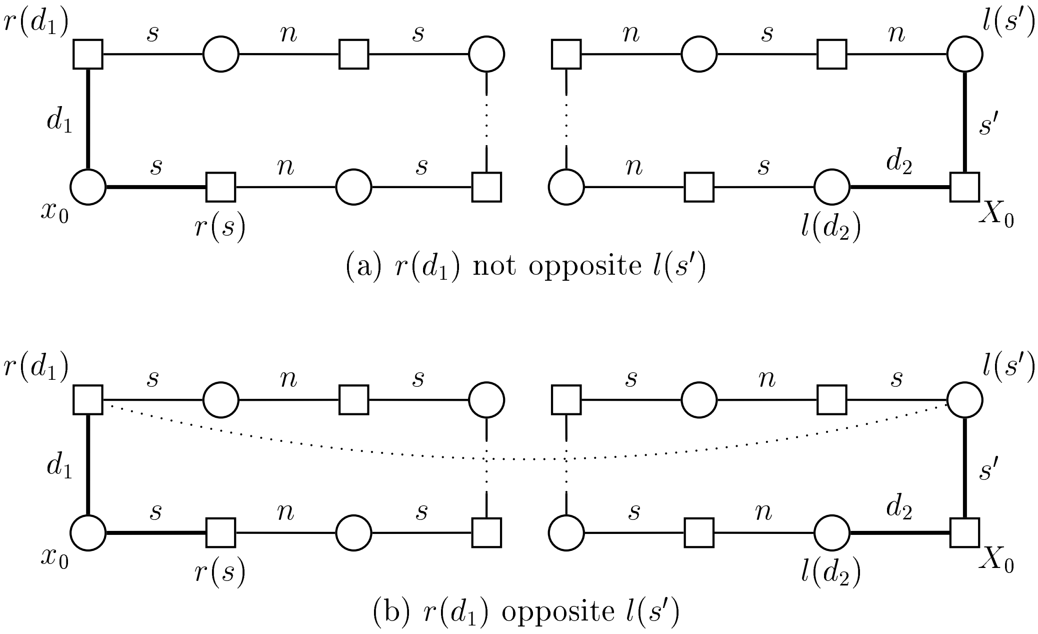
proof, we will show that in all cases except one the required coloring of the cycle cover C using 3 colors per row can be obtained. The only remaining case (Case 6) can then be handled by modifying the coloring of the PP-matching.

Note that C has two pre-colored edges with colors s and d_1 incident to x_0, and two pre-colored edges with colors s' and d_2 incident to X_0. We consider several cases regarding the structure of the cycles containing x_0 and X_0 in C.

Case 1: x_0 and X_0 are contained in the same cycle. The uncolored edges of the cycle consist of two paths of odd length. Color them by alternating n and s, starting with n. The remaining cycles are also colored by alternating n and s. In this way every vertex except x_0 and X_0 sees color n. Therefore, every pair of opposite vertices (except the top row) shares n, and we have at most 3 colors per row (see Figure 4.3).

Case 2: x_0 and X_0 are contained in different cycles, $r(s) \neq r(d_1)$, and $l(s') \neq l(d_2)$. Color all cycles not containing x_0 or X_0 by alternating n and s. Color the uncolored edges of the cycle containing x_0 by alternating colors n and s, starting with n on the edge incident to $r(s)$. If $r(d_1)$ is not opposite to $l(s')$, color the uncolored edges of the cycle containing X_0 by alternating colors n and s, starting with n on the edge incident to $l(s')$ (see part (a) of Figure 4.4). Among all vertices except x_0 and X_0, every vertex except $r(d_1)$ and $l(d_2)$ sees n, and every vertex except $l(s')$ sees s. As $l(s')$ is not opposite to $r(d_1)$ and cannot be opposite to $l(d_2)$, $l(s')$ shares n with its opposite vertex. Every other vertex (except x_0 and X_0) shares s with its opposite vertex. A coloring with 3 colors per row (except the top row) is achieved.

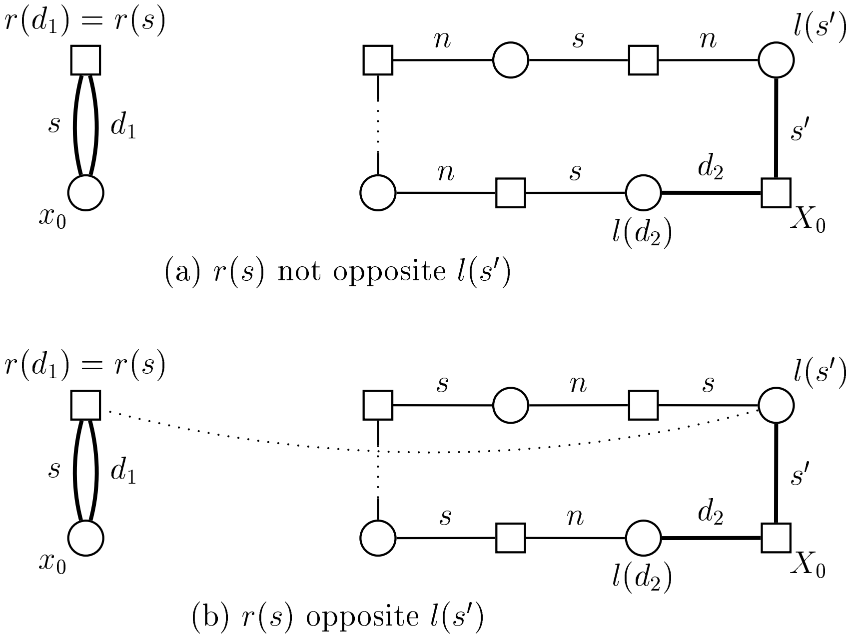
If $r(d_1)$ is opposite to $l(s')$, color the uncolored edges of the cycle containing X_0 by alternating colors n and s again, but starting with s on the edge incident to $l(s')$ (see part (b) of Figure 4.4). Now $r(d_1)$ and $l(s')$ share color s, and all other pairs of opposite vertices (except x_0 and X_0) share color n. Hence, we have a coloring with 3 colors per row (except the top row).



(a) $r(d_1)$ not opposite $l(s')$

(b) $r(d_1)$ opposite $l(s')$

Figure 4.4: Case 2: x_0 and X_0 are in separate non-degenerate cycles



(a) $r(s)$ not opposite $l(s')$

(b) $r(s)$ opposite $l(s')$

Figure 4.5: Case 3: x_0 and X_0 are in separate cycles, and the cycle containing x_0 is degenerate

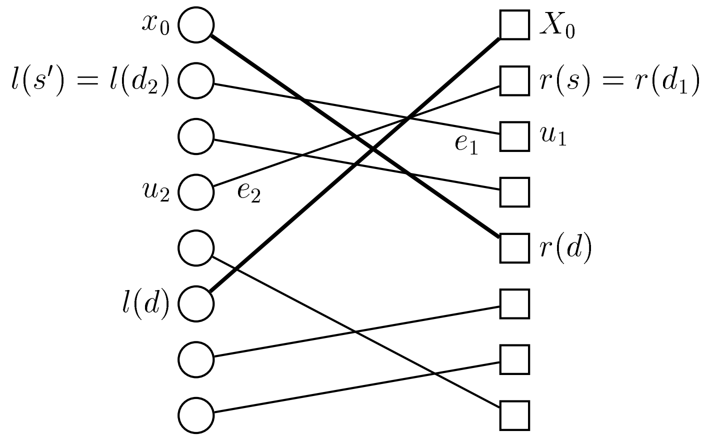


Figure 4.6: Case 6: Modified coloring of the PP-matching: e_1 is colored s, e_2 is colored s', all other edges are colored d

Case 3: x_0 and X_0 are contained in different cycles, $r(s) = r(d_1)$, and $l(s') \neq l(d_2)$. Color the cycles containing neither x_0 nor X_0 by alternating n and s. If $r(s)$ is not opposite to $l(s')$, color the uncolored edges of the cycle containing X_0 by alternating colors n and s, starting with n on the edge incident to $l(s')$. Among all vertices excluding x_0 and X_0, every vertex except $r(d_1)$ and $l(d_2)$ sees n, and every vertex except $l(s')$ sees s. As $l(s')$ is not opposite to $r(d_1)$ and cannot be opposite to $l(d_2)$, $l(s')$ shares n with its opposite vertex. Every other vertex shares s with its opposite vertex. A coloring with 3 colors per row is achieved.

If $r(d_1)$ is opposite to $l(s')$, color the uncolored edges of the cycle containing X_0 by alternating colors n and s again, but starting with s on the edge incident to $l(s')$. Now $r(d_1)$ and $l(s')$ share color s, and all other pairs of opposite vertices (except x_0 and X_0) share color n. Hence, we have a coloring with 3 colors per row.

Case 4: x_0 and X_0 are contained in different cycles, $r(s) \neq r(d_1)$, and $l(s') = l(d_2)$. This case is symmetrical to Case 3 and can be handled analogously.

Case 5: x_0 and X_0 are contained in different cycles, $r(s) = r(d_1)$, $l(s') = l(d_2)$, and $r(s)$ is not opposite $l(s')$. Color all uncolored cycles by alternating s and s'. Every vertex shares either s or s' (or both) with its opposite vertex, and thus the coloring uses at most 3 colors per row. No new color is required in this case.

Case 6: x_0 and X_0 are contained in different cycles, $r(s) = r(d_1)$, $l(s') = l(d_2)$, and $r(s)$ is opposite $l(s')$. Note that the row containing $r(s)$ and $l(s')$ sees the four colors s, s', d_1, and d_2. Since the condition of the lemma states that no row except the top row sees 5 colors on pre-colored edges, we know that $r(d) \neq r(s)$ and $l(d) \neq l(s')$. Denote the PP-matching by M. Let e_1 be the edge incident to $l(s')$ in M, and let u_1 be its other endpoint. Similarly, let e_2 be the edge incident to $r(s)$ in M, and let u_2 be its other endpoint. Note that $u_2 \neq x_0$ and $u_1 \neq X_0$ (see Figure 4.6). We cannot allow color d on e_1 or e_2, because this would give 5 colors in a row. Therefore, we color edge e_1 with color s and edge e_2 with color s'. (Note that $e_1 \neq e_2$, because vertices $r(s)$ and $l(s')$ are in the same row and, therefore, cannot be adjacent.) All other edges of M are colored d. The row containing $r(s)$ and $l(s')$ sees 4 colors now, and no other row (except the top row) sees colors d_1 or d_2. But we must take special care when coloring the uncolored edges of the cycle cover, because color s is forbidden at u_1, and s' is forbidden at u_2. However, color d is still available at u_1 and u_2, hence it may be used to color an edge $\{u_1, u_2\}$ in the cycle cover, in case such an edge exists.

The cycle cover C consists of two degenerate (i.e., having length two) cycles containing x_0 and X_0, and a number of disjoint cycles, each with an even number of uncolored edges. We want to color the uncolored edges of C using colors s, s', and n such that u_1 does not see s and u_2 does not see s' in C.

If u_1 and u_2 are contained in different cycles of C, color the cycle containing u_1 by alternating s' and n, and the cycle containing u_2 as well as all other cycles by alternating s and n.

If u_1 and u_2 are contained in the same cycle, and if this cycle has length 2, we color this cycle with colors n and d, and all other cycles by alternating n and s.

If u_1 and u_2 are contained in the same cycle, if this cycle has length greater than 2, and if an edge $\{u_1, u_2\}$ is part of the cycle, we color $\{u_1, u_2\}$ with color d and the rest of the cycle by alternating n and s, starting with n at u_1 and u_2. All other cycles are also colored by alternating n and s.

If u_1 and u_2 are contained in the same cycle, if this cycle has length greater than 2, and if no edge $\{u_1, u_2\}$ is part of the cycle, then the cycle consists of two paths from u_1 to u_2 with odd length at least 3. One path is colored by alternating n and s, starting and ending with n. The other path is colored by starting with s' at u_1 and then alternating n and s, ending with s at u_2.

In all subcases of Case 6, we have achieved at most 4 colors per row, because all rows except the top row and the row with $r(s)$ and $l(s')$ see only the colors s, s', n, and d.

It is clear that the case analysis and coloring rules given above can be implemented to run in time linear in the size of H. As H is a 3-regular graph with $4\delta(v)$ vertices, the running-time for edge coloring H as required is $O(\delta(v))$. □

Now that we have Lemma 4.2.3 available as a tool for coloring KS-subgraphs, our algorithm tries to select KS-subgraphs satisfying the condition of the lemma whenever possible. In some cases, however, a different kind of triplet must be colored: a triplet that contains, like a KS-subgraph, two single color edges and four double color edges, but without any double color occurring twice. If such a triplet H can be partitioned into a gadget H_1 and a matching H_2 on all vertices except x_0 and X_0, the following lemma shows that H_1 can, under certain conditions, be colored using at most three colors per row (except the top row), and without using any new color. This is possible because degenerate cases are excluded by requiring that pre-colored edges colored with the double colors occurring in H are not parallel. The coloring must reuse double colors carefully in order to make sure that no conflict with the pre-colored edges outside H_1 is introduced. As the edges of H_2 can be colored with a new color, the lemma implies that H can be colored as required: using one new color and at most four colors per row (except the top row). The proof of the lemma is rather long and consists of a thorough case analysis; the reader may wish to skip the proof on first reading and continue to read from page 86 instead.

Lemma 4.2.4 *Let H_1 be a subgraph of G_v that is a gadget with single color s and double colors a and a' incident to x_0, and with single color s' and double colors b and b' incident to X_0. Assume that H_1 does not contain parallel pre-colored edges. Furthermore, assume that the set of pre-colored edges of G_v with colors in $\{a, a', b, b'\}$ does not contain parallel edges. Then H_1 can be colored in linear time without using any new color by re-using s, s', a, a', b, and b' such that opposite vertices (except x_0 and X_0) share at least one color.*

Proof: As H_1 is a gadget, it can be seen as a collection of three paths, each running from x_0 to X_0 or looping back from x_0 or X_0 to its starting point, and a certain number of cycles involving only vertices other than x_0 and X_0. The cycles have even length and are colored by alternating colors s and s'. There are two possibilities for the remaining paths: either all three of them run from x_0 to X_0, or only one runs from x_0 to X_0 while the other two loop back to their starting point. Kumar and Schwabe [KS97] refer to such gadgets as *football-shaped* gadgets and *dumbbell-shaped* gadgets, respectively. The vertices adjacent to x_0 are named v_1, v_2, and v_3, and those adjacent to

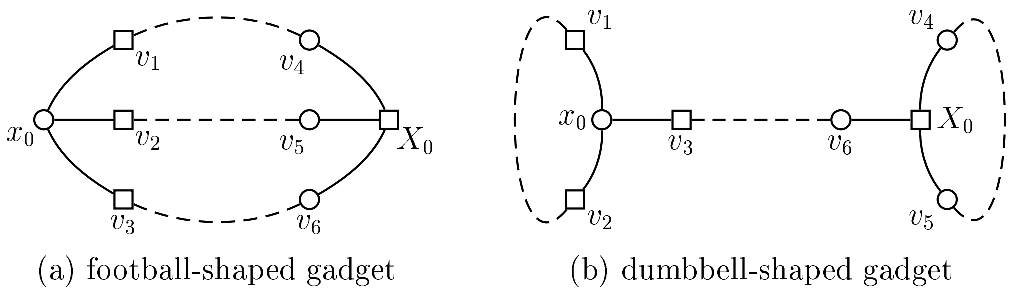


Figure 4.7: The two kinds of gadgets

X_0 are named v_4, v_5, and v_6. The pre-colored edge incident to v_i is denoted by e_i. For football-shaped gadgets, we assume that one of the three paths runs from v_1 to v_4, one from v_2 to v_5, and one from v_3 to v_6. For dumbbell-shaped gadgets, we assume that one of the three paths runs from v_1 to v_2, one from v_3 to v_6, and one from v_4 to v_5. Figure 4.7 shows the gadget shapes and the naming of vertices adjacent to x_0 and X_0. Note that dumbbell-shaped gadgets are non-degenerate (i.e., $v_1 \neq v_2$ and $v_4 \neq v_5$) because there are no parallel pre-colored edges.

We study different cases depending on the arrangement of colors on the pre-colored edges. Note that we omit symmetrical cases that arise from exchanging colors a and a' or b and b'. Cases F1 and F2 pertain to football-shaped gadgets, and cases D1 to D3 deal with dumbbell-shaped gadgets. When we use expressions like *all vertices* or *every vertex*, we refer to the vertices of G_v excluding x_0 and X_0.

Case F1: s and s' are on different paths.
Let the colors on e_1 to e_6 be a, a', s, s', b', and b, respectively.

Case F1.1: v_1 is not opposite v_5 or v_6.
Color all three paths by alternating colors s and s', starting with s at v_1 and with s' at v_2 and v_3 (see Figure 4.8). All vertices except v_1 see color s'. All vertices except v_2, v_5, and v_6 see color s. v_1 cannot be opposite v_2. As v_1 is not opposite v_5 or v_6, the coloring is correct.

Case F1.2: v_1 is opposite v_5.
All paths are colored by alternating colors s and s', starting with s at v_1 and v_2 and with s' at v_3. v_1 and v_5 share s. The only other vertex that does not see s' is v_2, and the only vertex that does not see s is v_6. Hence, the coloring is valid unless v_2 is opposite v_6 (see part (a) of Figure 4.9).

If v_2 is opposite v_6, we replace the color s on the edge incident to v_2 by b. (Note that $v_2 \neq r(b)$.) Unless v_2 and v_5 are adjacent, this coloring is correct because v_2 and v_6 share b, whereas all remaining vertices share s or s' (see

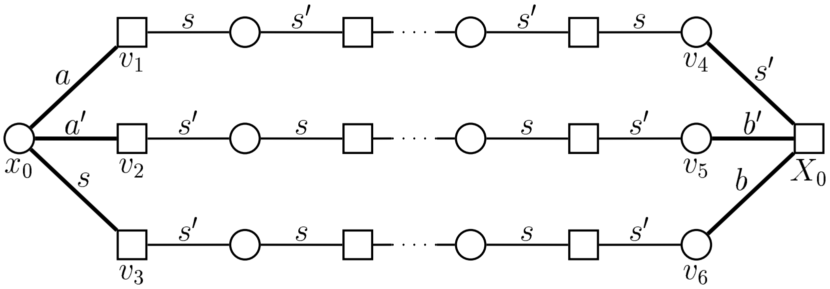
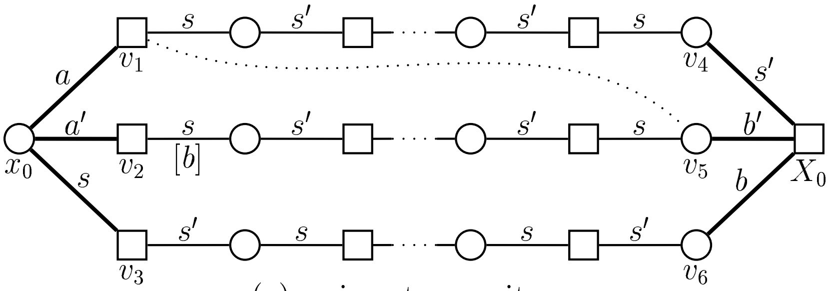
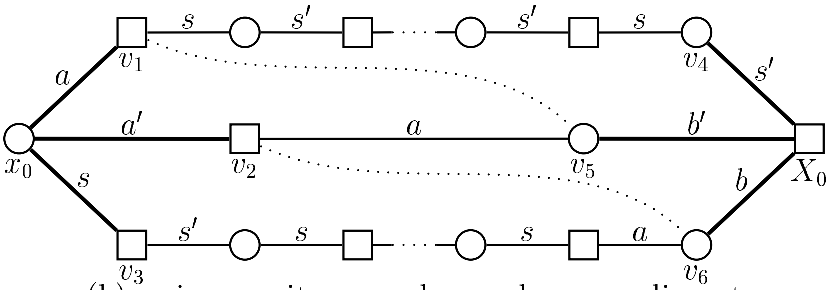


Figure 4.8: Case F1.1: v_1 is not opposite v_5 or v_6



(a) v_2 is not opposite v_6
[v_2 is opposite v_6, and v_2 is not adjacent to v_5]



(b) v_2 is opposite v_6, and v_2 and v_5 are adjacent

Figure 4.9: Case F1.2: v_1 is opposite v_5

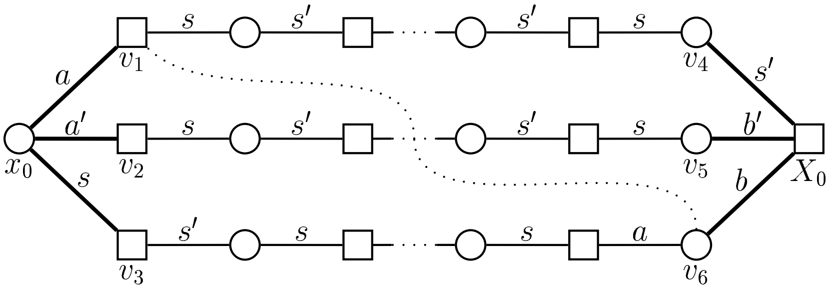


Figure 4.10: Case F1.3: v_1 is opposite v_6

part (a) of Figure 4.9, color in square brackets). If v_2 and v_5 are adjacent, we color the edge $\{v_2, v_5\}$ and the edge incident to v_6 but not to X_0 with a. (Note that $v_5 \neq l(a)$ and $v_6 \neq l(a)$.) This coloring is valid because v_1 and v_5 share a, v_2 and v_6 share a, and all other vertices share s (see part (b) of Figure 4.9).

Case F1.3: v_1 is opposite v_6.
Color all paths by alternating s and s', starting with s at v_1 and v_2 and with s' at v_3. Re-color the edge incident to v_6 with color a. (Note that $v_6 \neq l(a)$.) The coloring is valid because v_1 and v_6 share a and all other vertices share s.

Case F2: s and s' are on the same path.
Let the colors on e_1 to e_6 be s, a, a', s', b, and b', respectively. Consider the vertex v_4. Denote the vertex different from X_0 that is adjacent to v_4 by u. Note that $u = v_1$ is possible. Since the pre-colored edges with colors a and a' incident to X_0 (these edges are contained in $G_v \setminus H_1$) are not parallel, either a or a' can be used to color the edge $\{u, v_4\}$. Without loss of generality, assume that a can be used for this purpose.

Case F2.1: v_3 is not opposite v_4.
Color all three paths by alternating colors s and s', starting with s' at v_1 and with s at v_2 and v_3. The color on edge $\{u, v_4\}$ is replaced by a. Every vertex except v_4 sees color s, and v_4 sees colors s' and a. The only vertex that does not see s' and a and that could be opposite v_4 is v_3. Hence, the coloring is valid (see Figure 4.11).

Case F2.2: v_3 is opposite v_4, u is not opposite v_5 or v_6.
Color all three paths by alternating colors s' and s, starting with s'. The color on edge $\{u, v_4\}$ is replaced by a. Every vertex except u sees color s', and u sees color s. The only vertices that do not see s and could be opposite u are v_5 and v_6, because v_2 is a right vertex and v_3 is opposite v_4. Hence, the coloring is valid (see Figure 4.12).

Case F2.3: v_3 is opposite v_4, u is opposite v_5.
Color the paths by alternating s and s', starting with s' at v_1 and v_3 and with s at v_2. The color on edge $\{u, v_4\}$ is replaced by a. v_3 and v_4 share s'. All other vertices except v_6 see s. v_6 sees s'. The only vertex that does not see s' and could be opposite v_6 is v_2. Hence, the coloring is valid unless v_6 is opposite v_2 (see part (a) of Figure 4.13). If v_6 is opposite v_2, replace the color on the edge incident to v_6 but not to X_0 by color a (see part (b) of Figure 4.13). Now, v_2 and v_6 share a, v_3 and v_4 share s', and all other vertices share s (if v_3 is adjacent v_6, v_3 and v_4 share color a).

Case F2.4: v_3 is opposite v_4, u is opposite v_6.
The second coloring from Case F2.3 is valid in this case as well, because v_3 and v_4 share s' (or a, in case v_3 and v_6 are adjacent), u and v_6 share a, and

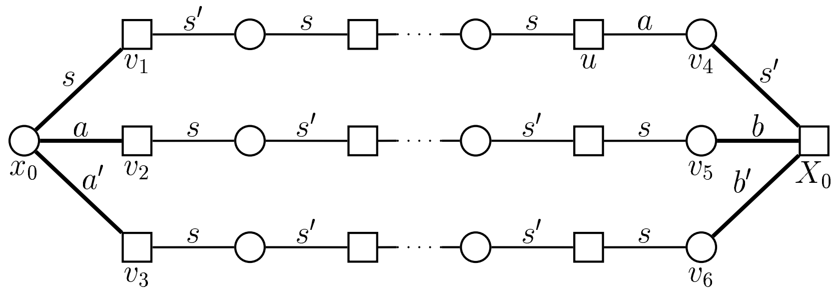


Figure 4.11: Case F2.1: v_3 is not opposite v_4

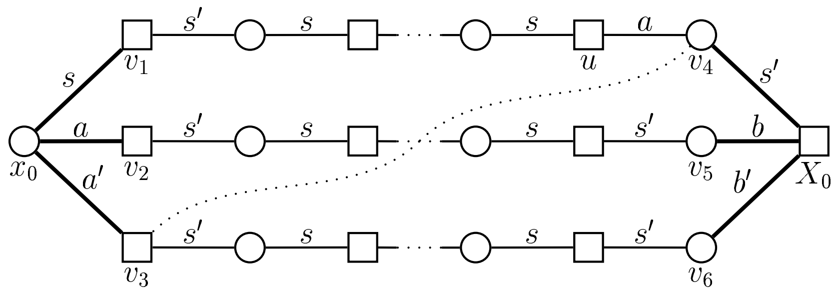
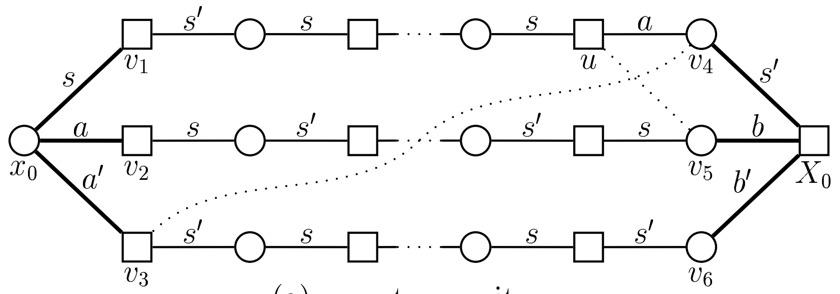
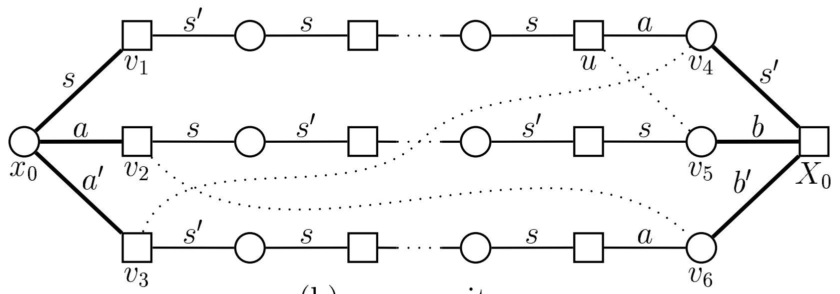


Figure 4.12: Case F2.2: v_3 is opposite v_4, u is not opposite v_5 or v_6



(a) v_2 not opposite v_6



(b) v_2 opposite v_6

Figure 4.13: Case F2.3: v_3 is opposite v_4, u is opposite v_5

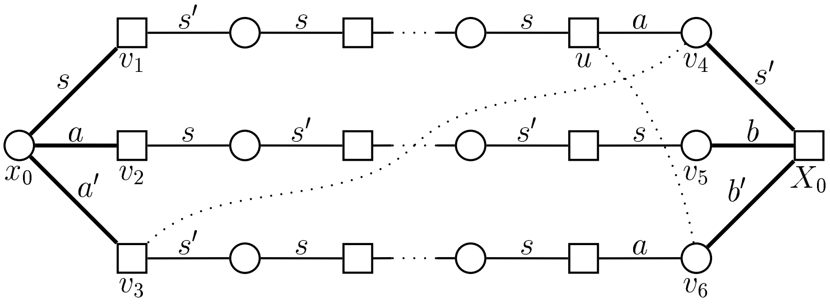


Figure 4.14: Case F2.4: v_3 is opposite v_4, u is opposite v_6

all other vertices share s (see Figure 4.14).

We have seen how H_1 can be colored as required if it is a football-shaped gadget. For dumbbell-shaped gadgets, we distinguish cases D1 to D3.

Case D1: The path from x_0 to X_0 contains one single color.
Without loss of generality, we assume that the colors on e_1 to e_6 are a, a', s, s', b', and b, respectively. In addition, if either v_1 or v_2 is opposite v_5, we name the vertices such that v_2 is opposite v_5. Figure 4.15 illustrates Cases D1.1–D1.3.

Case D1.1: v_1 is not opposite v_6.
Color the three paths by alternating s' and s, starting with s' at v_2, v_3, and v_5. All vertices except v_1 see color s'. v_1 sees color s. The only vertices that do not see s are v_2, v_5, and v_6. v_2 cannot be opposite v_1 because both are right vertices. v_5 cannot be opposite v_1 because of our naming conventions. Therefore, none of the possibly problematic vertices is opposite v_1. Hence, v_1 shares s with its opposite vertex, and all other vertices share s'.

Case D1.2: v_1 is opposite v_6, v_2 is not opposite v_5.
Color the paths by alternating s' and s, starting with s' at v_1, v_3, and v_5. Now, v_2 shares s with its opposite vertex, and all other vertices share s'.

Case D1.3: v_1 is opposite v_6, v_2 is opposite v_5.
Use the coloring from Case D1.2 and replace color s by b' on the edge incident to v_2. v_2 and v_5 share b', and all other vertices share s'.

Case D2: The path from x_0 to X_0 contains no single color.
Without loss of generality, we assume that the colors on e_1 to e_6 are s, a, a', s', b, and b', respectively. Figure 4.16 illustrates Cases D2.1–D2.4.

Case D2.1: v_2 is not opposite v_5 or v_6.
Color the three paths by alternating s' and s, starting with s' at v_1, v_3, and v_5. All vertices except v_2 see color s'. v_2 sees color s. The only vertices

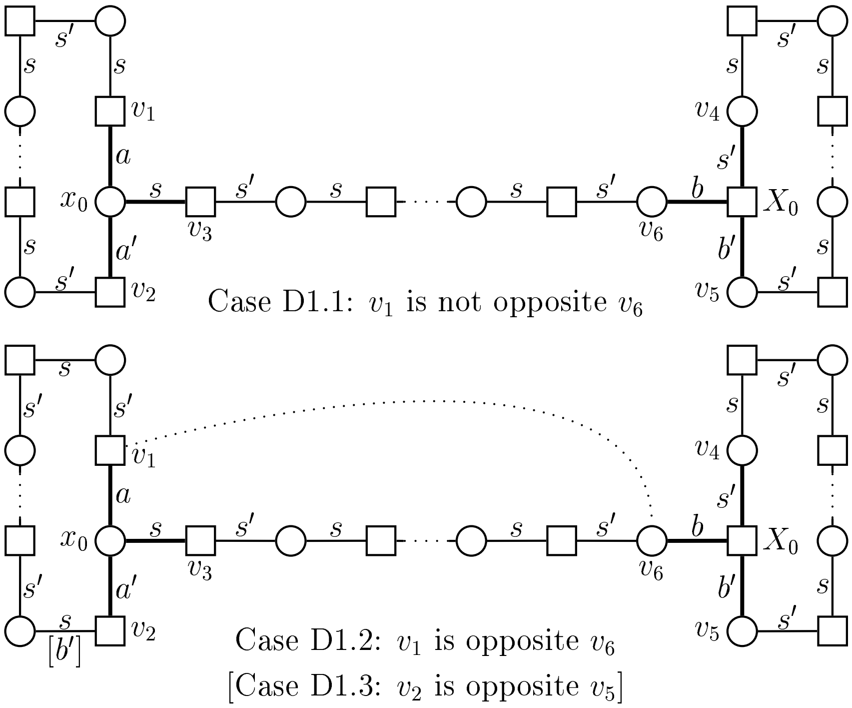


Figure 4.15: Case D1: One single color on the path from x_0 to X_0

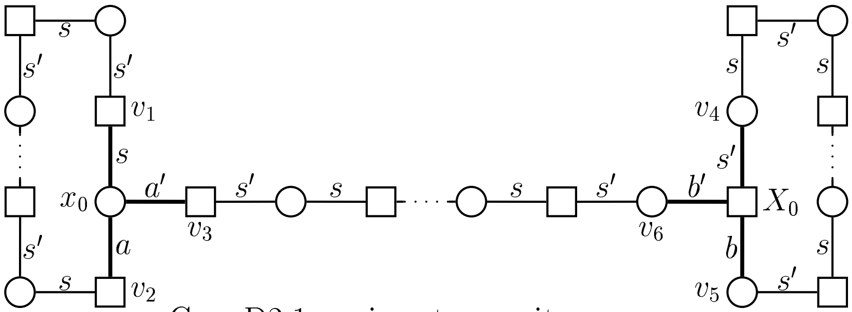
that do not see s and could possibly be opposite v_2 are v_5 and v_6. Hence, the coloring is valid.

Case D2.2: v_2 is opposite v_5.
Color the paths by alternating s and s', starting with s at v_2, v_3, and v_4. Replace color s' by a on the edge incident to v_5. (Note that $v_5 \neq l(a)$.) Now, v_2 and v_5 share color a, and all other vertices share s.

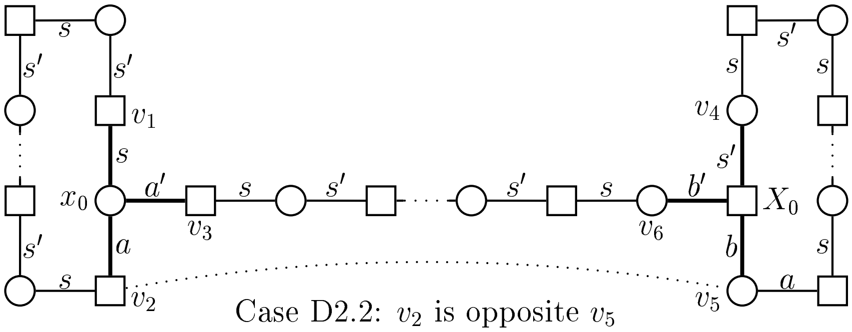
Case D2.3: v_2 is opposite v_6, v_3 is not opposite v_5.
Color the paths by alternating s and s' just like in Case D2.2, without replacing color s' by a on one edge afterwards. Every vertex except v_5 sees s. v_5 sees s'. The only vertex that does not see s' and that could possibly be opposite v_5 is v_3. Hence, v_5 shares s' with its opposite vertex, and all other vertices share s.

Case D2.4: v_2 is opposite v_6, v_3 is opposite v_5.
Use the coloring from Case D2.3 and replace the color s' by a' on the edge incident to v_5. v_3 and v_5 share a', and all other vertices share s.

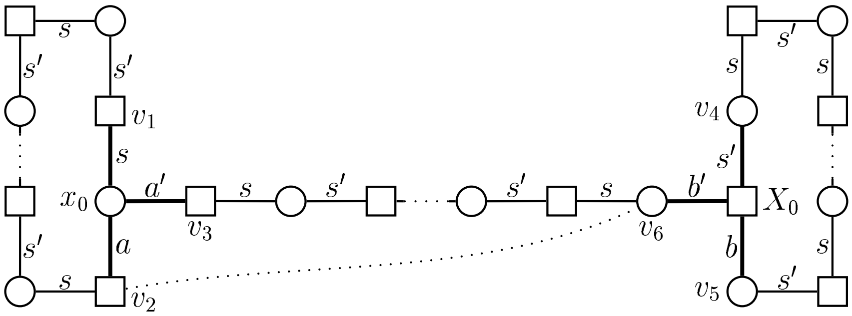
Case D3: The path from x_0 to X_0 contains s and s'.
Without loss of generality, we assume that the colors on e_1 to e_6 are a, a', s, b, b', and s', respectively. Denote by u the vertex that is adjacent to v_6 but



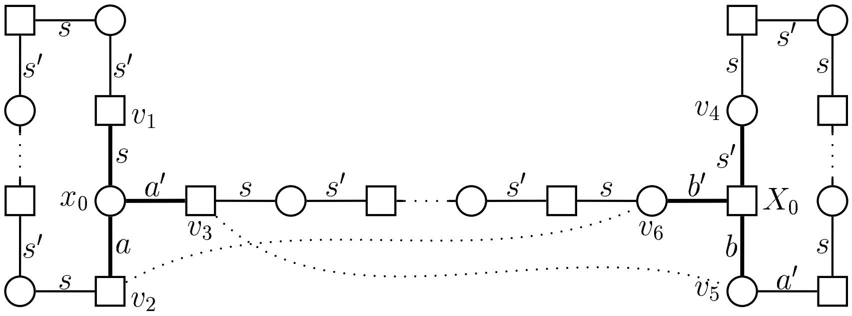
Case D2.1: v_2 is not opposite v_5 or v_6



Case D2.2: v_2 is opposite v_5



Case D2.3: v_2 is opposite v_6, v_3 is not opposite v_5



Case D2.4: v_2 is opposite v_6, v_3 is opposite v_5

Figure 4.16: Case D2: No single color on the path from x_0 to X_0

not to X_0. Note that $u = v_3$ is possible. One of the edges colored a and a' outside H_1 can be incident to v_6, but not both of them. Hence, either a or a' can be used to color the edge $\{u, v_6\}$. Without loss of generality, assume that a is this color. Figures 4.17 and 4.18 illustrate Cases D3.1–D3.4 and D3.5–D3.6, respectively.

Case D3.1: v_5 is not opposite v_1 or u, and v_2 is not opposite v_4.
Color the three paths by alternating s and s', starting with s' at v_2, v_3, and v_5. Recolor edge $\{u, v_6\}$ with color a. All vertices except v_2, v_5, and v_6 see color s. All vertices except v_1, u, and v_4 see color s'. Note that u cannot be opposite v_6 since these two vertices are adjacent. Furthermore, note that v_1 and v_6 share a if they are opposite each other. Hence, the only problematic pairs of vertices are v_1–v_5, u–v_5, and v_2–v_4. Since none of these is a pair of opposite vertices, the coloring is valid.

Case D3.2: v_5 is opposite v_1, and u is not opposite v_4.
Use the coloring from Case D3.1 and exchange colors s and s' on the path from v_4 to v_5. v_1 and v_5 share s. Every other vertex except u sees s'. u sees s. The only vertex that could be opposite u and that does not see s is v_4. Hence, the coloring is valid.

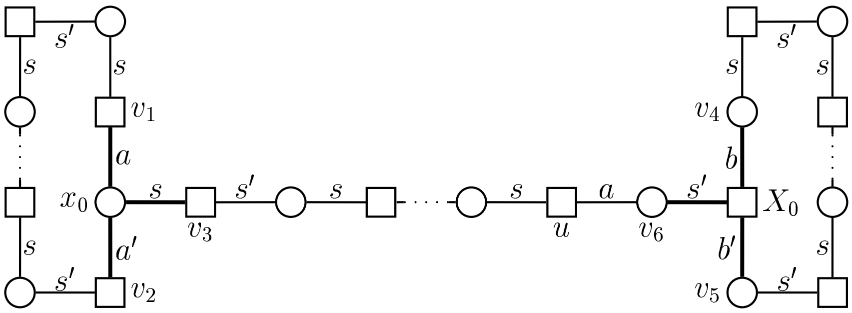
Case D3.3: v_5 is opposite v_1, u is opposite v_4, v_2 is not opposite v_6.
Use the coloring from Case D3.1 and exchange colors s and s' on the path from v_1 to v_2. v_1 and v_5 share s', and u and v_4 share s. All other vertices except v_2 see s'. v_2 sees s. The only vertex that does not see s and that could be opposite v_2 is v_6. Hence, the coloring is valid.

Case D3.4: v_5 is opposite v_1, u is opposite v_4, v_2 is opposite v_6.
Use the coloring from Case D3.1 and replace the color s' by a on the edge incident to v_5. (Note that $v_5 \neq l(a)$.) Now, v_1 and v_5 share a, v_2 and v_6 share s', and all other vertices share s. The coloring is valid.

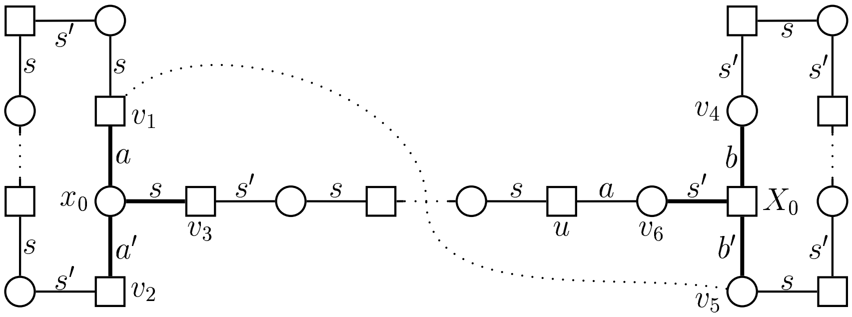
Case D3.5: v_2 is opposite v_4.
Use the same coloring as in Case D3.2. v_2 and v_4 share s'. All other vertices except v_6 see s. v_6 sees a and s'. The only vertex that sees neither a nor s' is v_5, which cannot be opposite v_6. Hence, the coloring is valid.

Case D3.6: v_5 is opposite u.
Again, use the same coloring as in Case D3.2. v_5 and u share s. All other vertices except v_1 see s'. v_1 sees a and s. The only vertex that sees neither a nor s and that can be opposite v_1 is v_4. Hence, the coloring is valid unless v_1 is opposite v_4. If v_1 is indeed opposite v_4, replace color s by b on the edge incident to v_1. Now, v_1 and v_4 share color b, u and v_5 share color s, and all other vertices share color s'.

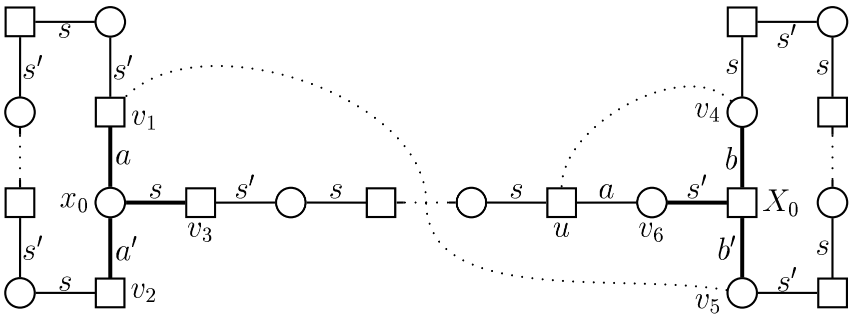
The foregoing case analysis has shown that the gadget H_1 can be colored as required in all cases. It is easy to see that the resulting coloring algorithm can be implemented to run in time linear in the size of the gadget. □



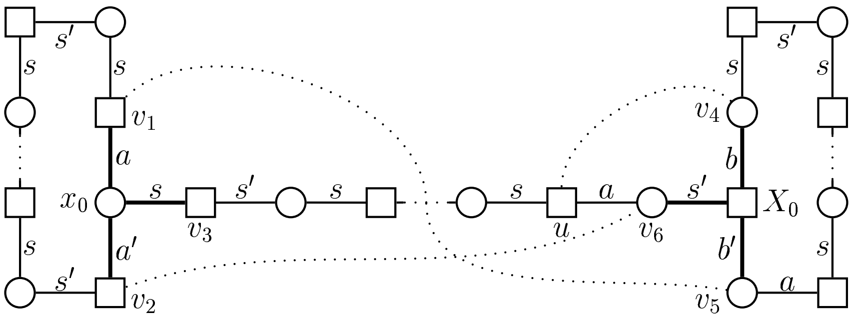
Case D3.1: v_5 not opposite v_1 or u; v_2 not opposite v_4



Case D3.2: v_5 is opposite v_1; u is not opposite v_4

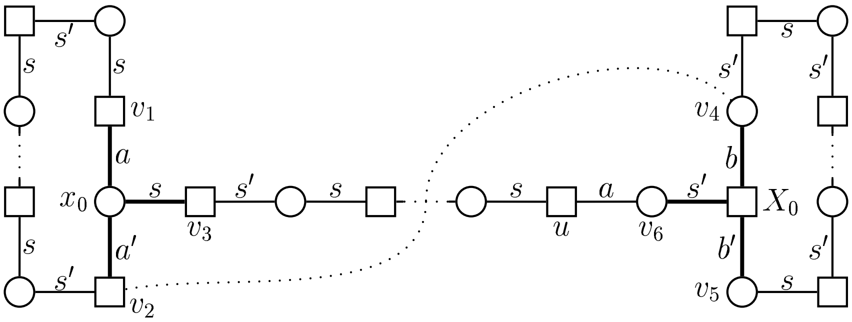


Case D3.3: v_5 opposite v_1; u opposite v_4; v_2 not opposite v_6

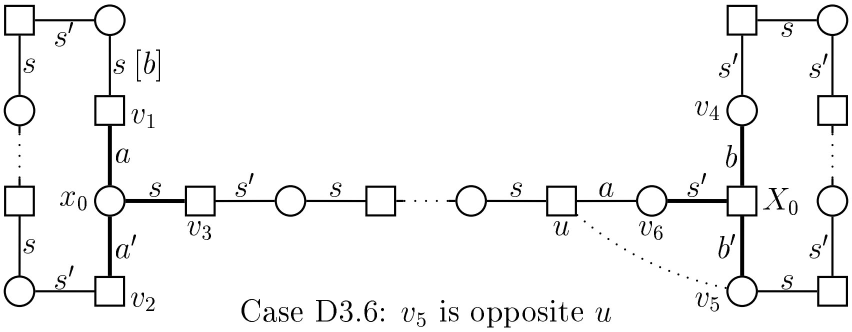


Case D3.4: v_5 opposite v_1; u opposite v_4; v_2 opposite v_6

Figure 4.17: Cases D3.1–D3.4: s, s' are on the path from x_0 to X_0



Case D3.5: v_2 is opposite v_4



Case D3.6: v_5 is opposite u
[v_1 is opposite v_4]

Figure 4.18: Cases D3.5–D3.6: s and s' are on the path from x_0 to X_0

In Section 4.2.3, we show how KS-subgraphs satisfying the condition of Lemma 4.2.3 and other triplets whose coloring is straightforward can be extracted from G_v until a configuration out of a limited set of alternatives is obtained. In Section 4.2.4, we show how to make use of Lemma 4.2.4 to deal with these configurations whose partitioning is more involved. In Section 4.2.5, we present the modifications that are necessary for the cases $L = 3\ell + 1$ and $L = 3\ell + 2$. Section 4.2.6 explains why the resulting algorithm is optimal in the class of local greedy algorithms. Section 4.2.7 reports experience with an implementation of the algorithm.

4.2.3 Partitioning into Triplets (Simple Cases)

Like in [KP96], ST- and TT-matchings are viewed as a collection of chains and cycles. Given a matching M, we refer to the color of the edge incident to x_0 as the *left color* of M, and to the color of the edge incident to X_0 as the *right color* of M. A sequence $\langle M_0, M_1, \ldots, M_{l-1}\rangle$ of $l > 1$ TT-matchings is a *cycle* of length l if the right color of M_i is equal to the left color of $M_{i+1 \bmod l}$ for all $0 \le i \le l-1$. A sequence $\langle M_0, M_1, \ldots, M_{l-1}\rangle$ of $l > 1$ matchings is a *chain* of length l if M_0 and M_{l-1} are ST-matchings, $M_1, M_2, \ldots, M_{l-2}$ are TT-matchings, and the right color of M_i is equal to the left color of M_{i+1} for all $0 \leq i \leq l-2$. The terms *l-cycle* and *l-chain* refer to cycles and chains of length l, respectively.

Let us introduce some notation. We use letters a, a', b, b', ... to denote double colors, and letters s, s', t, t', ... to denote single colors. For instance, an SS-matching with left color s and right color s', a 2-chain, a 3-chain, and a 5-cycle are represented as follows:

$$\begin{matrix} s \\ s' \end{matrix} \qquad \begin{matrix} s & a \\ a & s' \end{matrix} \qquad \begin{matrix} s & a & b \\ a & b & s' \end{matrix} \qquad \begin{matrix} a & b & c & d & e \\ b & c & d & e & a \end{matrix}$$

A matching is specified by writing its left color above its right color, and a sequence of matchings is given by writing down the individual matchings from left to right.

The set of all matchings can be grouped into SS-matchings, chains, PP-matchings, and cycles in time $O(L)$. Next, the chains and cycles are preprocessed so that the resulting chains and cycles do not contain parallel pre-colored edges. If the i-th and j-th matching in a cycle or chain contain parallel pre-colored edges incident to, say, x_0, then the parallel pre-colored edges can be exchanged, and this results in a shorter cycle or chain (or SS-matching) and an additional cycle. For instance, if $r(b) = r(e)$ in the following example, the 6-chain can be divided into a 3-chain and a 3-cycle.

$$\begin{matrix} s & a & \mathbf{b} & c & d & \mathbf{e} \\ a & b & c & d & e & s' \end{matrix} \longrightarrow \begin{matrix} s & a & \mathbf{b} \\ a & b & s' \end{matrix} + \begin{matrix} \mathbf{e} & c & d \\ c & d & e \end{matrix}$$

This preprocessing can be implemented to run in time $O(\delta(v) + L)$. The benefit obtained is that any KS-subgraph containing two matchings from the same cycle or chain now satisfies the condition of Lemma 4.2.3 and can be colored as required in linear time.

Since we have $S = D$, the ratio of the number of edges with double colors to the number of edges with single colors is $2:1$ altogether. The only groups of matchings that have a smaller ratio than that are SS-matchings and 2-chains. Therefore, whenever there is a group whose number of double color edges is more than twice the number of its single color edges, there must be a sufficient number of SS-matchings and 2-chains to outweigh this imbalance. As we extract only 3-regular subgraphs with four double color edges and two single color edges, the ratio of double color edges to single color edges in the remaining graph will again be $2:1$.

The following groups of matchings can be combined with SS-matchings and 2-chains easily in order to obtain KS-subgraphs satisfying the condition of Lemma 4.2.3.

A cycle of even length can be divided into pairs of matchings such that every pair of matchings has a common double color. Each pair can either be combined with an SS-matching to obtain one KS-subgraph, or with two 2-chains to obtain two KS-subgraphs. For example, a 4-cycle can be combined with two 2-chains and one SS-matching:

$$\begin{matrix} a & b & c & d \\ b & c & d & a \end{matrix} + \begin{matrix} s & e \\ e & s' \end{matrix} + \begin{matrix} t & f \\ f & t' \end{matrix} + \begin{matrix} u \\ u' \end{matrix} = \begin{matrix} u & a & b \\ u' & b & c \end{matrix} + \begin{matrix} s & e & c \\ e & s' & d \end{matrix} + \begin{matrix} t & f & d \\ f & t' & a \end{matrix}$$

There is always a sufficient number of SS-matchings or 2-chains available for such reductions because the ratio of double color edges to single color edges is $2:1$ altogether. In particular, when no SS-matchings are left, there must be at least one 2-chain for every remaining matching of the cycle.

A chain of odd length can be divided into two parts. The two matchings at the beginning of the chain and the matching at the end together constitute a KS-subgraph. If the chain has length 3, we are finished. Otherwise, a sequence of TT-matchings of even length remains. This sequence has the property that any two consecutive matchings share a double color. Hence, this sequence can be handled just like a cycle of even length.

Two chains of even length, at least one of which has length at least 4, can also be combined with SS-matchings or 2-chains. Denote the longer chain by C_1 and the other one by C_2. Combine the first two matchings of C_1 with the last matching of C_2, and the first matching of C_2 with the last two matchings of C_1. Both combinations result in KS-subgraphs. We are left with zero, one, or two sequences of even length such that consecutive matchings in each sequence share a double color. Hence, these sequences can be handled like cycles of even length.

A 2-chain and a cycle of odd length can be handled by combining the 2-chain with the first matching of the cycle to obtain a KS-subgraph. The remainder of the cycle is a sequence of even length with the property that consecutive matchings share a double color, and it can be handled like a cycle of even length.

A 2-chain and a PP-matching can be colored without any new color. Color the PP-matching with its double color and each of the matchings in the 2-chain with its single color. Since the number of colors that appear in this 3-regular subgraph is 4 altogether, the constraint on the number of colors per row is satisfied.

Two PP-matchings and an SS-matching can also be colored without using any new color. Color each PP-matching with its double color and the SS-matching with one of its single colors. Again, there are only 4 colors involved altogether.

Applying these reductions in arbitrary order repeatedly as long as possible, we are left with SS-matchings, at most one PP-matching, at most one chain of even length greater than 2, and a number of cycles of odd length. (If there was a 2-chain left, the fact that the ratio of double color edges to single color edges is $2:1$ altogether would imply that there is also at least one PP-matching, cycle, or chain of length greater than 3. Hence, the 2-chain should have been combined with one of these.) Section 4.2.4 shows how these remaining matchings can be colored without violating the invariants.

4.2.4 Partitioning into Triplets (Difficult Cases)

First, we show how to handle two cycles of odd length.

Lemma 4.2.5 *Given two cycles, each without parallel pre-colored edges, of length $2i+1$ ($i \geq 1$) and $2j+1$ ($j \geq 1$), respectively, and $i+j+1$ SS-matchings, these $n = 3(i+j+1)$ matchings can be colored using at most $n/3$ new colors such that no row sees more than $(4/3)n$ colors.*

Proof: Denote the cycles by C_1 and C_2. Pick an arbitrary SS-matching M with single colors s and s'. Pick a matching M_1 with colors a and b from C_1 such that none of the pre-colored edges of M_1 is parallel to an edge colored s or s' in M. Pick a matching M_2 with colors a' and b' from C_2 with the same property. These matchings exist because both cycles have length at least 3 and do not contain parallel pre-colored edges.

Next, check whether any of the pre-colored edges colored a or b in C_1 is parallel to a pre-colored edge colored a' or b' in C_2. This can be checked by testing the conditions $r(a) = r(a')$, $r(a) = r(b')$, $r(b) = r(a')$, $r(b) = r(b')$, $l(a) = l(a')$, $l(a) = l(b')$, $l(b) = l(a')$, and $l(b) = l(b')$. If one of these cases occurs, a pre-colored edge e_1 in C_1 that is parallel to a pre-colored edge e_2 in C_2 has been found. Exchanging these two pre-colored edges turns $C_1 \cup C_2$ into a single cycle C of even length. C can be viewed as consisting of two subsequences, one originating from C_1 and one from C_2, such that pre-colored edges within a subsequence are not parallel. Choose a pair of consecutive matchings from C such that the pair consists of one matching from C_1 and one matching from C_2 (there are two possibilities for choosing such a pair; it doesn't matter which one is selected). The pair contains either M_1 or M_2. Hence, M and the two matchings from the pair represent a KS-subgraph satisfying the condition of Lemma 4.2.3 and can be colored as required. The remainder of C is a sequence of TT-matchings of even length that consists of one even subsequence of TT-matchings originating from C_1 and a second even subsequence of TT-matchings originating from C_2. By combining each pair of consecutive TT-matchings from this sequence with an arbitrary SS-matching we obtain KS-subgraphs that can be colored according to Lemma 4.2.3. (As each such KS-subgraph contains either two TT-matchings originating from C_1 or two TT-matchings originating from C_2, no row except the top row can see 5 colors on pre-colored edges.)

If no pre-colored edge colored a or b in C_1 is parallel to a pre-colored edge colored a' or b' in C_2, we consider the triplet $H = M \cup M_1 \cup M_2$. Remove the vertices x_0 and X_0 from H and add dummy edges $\{r(a), l(b')\}$, $\{r(a'), l(b)\}$, and $\{r(s), l(s')\}$. The obtained graph H' is again 3-regular and can be partitioned into three perfect matchings in linear time using the edge-coloring algorithm by Schrijver [Sch98]. If each of the matchings contains exactly one dummy edge, H can be split into an SS-matching with colors s and s', a TT-matching with colors a and b', and a TT-matching with colors a' and b. Exchanging the new matchings for the old ones turns the cycles C_1 and C_2 into a single cycle C of even length. We claim that C can be colored by combining pairs of consecutive TT-matchings with arbitrary SS-matchings if the matching with colors a' and b is considered the first matching of C. To see this, consider the two resulting pairs of matchings that contain a TT-matching originating from H: the first one contains the matching with colors a' and b as well as the original matching from C_1 with colors b and, say, c. We know $r(a') \neq r(b)$, because this case has been handled above, and $l(b) \neq l(c)$, because these two pre-colored edges both originate from C_1. Similar reasoning can be applied to the second pair containing a matching originating from H. Hence, each of the two pairs containing a matching originating from H can be combined with an SS-matching to give a KS-subgraph that can be colored according to Lemma 4.2.3. All other pairs of consecutive matchings from C involve two matchings originating from either C_1 or C_2, and they can be combined with SS-matchings in the same way.

If one of the matchings obtained from H' does not contain a dummy edge, this matching gives a partitioning of H into gadget H_1 and matching H_2. In that case, we can, according to Lemma 4.2.4, color H_1 using only its old colors such that no row except the top row sees more than 3 colors. H_2 can then be colored using a single new color. Therefore, H is colored such that no row except the top row sees more than 4 colors. When H has been colored, the remaining sequences $C_1 \setminus M_1$ and $C_2 \setminus M_2$ have even length and the property that consecutive matchings share a double color; they can be colored as shown in Section 4.2.3, reusing only colors not in $\{a, b, a', b'\}$. □

Since this lemma allows us to combine pairs of cycles of odd length with SS-matchings, we are left with SS-matchings, at most one PP-matching, at most one cycle of odd length, and at most one chain of even length. In addition, we note that the number of edges with double colors is divisible by 4 since the number of double colors is even ($D = 2\ell$ by assumption). Hence, only the following cases can occur (each with the appropriate number of SS-matchings):

(a) one PP-matching and one cycle of odd length

(b) one PP-matching and one chain of even length

(c) one cycle of odd length and one chain of even length

These are handled by the following three lemmas.

Lemma 4.2.6 *Given a PP-matching, a cycle of length* $2i+1$ *(*$i \geq 1$*) without parallel pre-colored edges, and* $i+1$ *SS-matchings, these* $n = 3(i+1)$ *matchings can be colored using at most* $n/3$ *new colors such that no row sees more than* $(4/3)n$ *colors.*

Proof: Pick an arbitrary SS-matching M with colors s and s'. Pick a matching M' with double colors a and b from the cycle such that none of its pre-colored edges is parallel to an edge colored s or s' in M. Consider the 3-regular subgraph H obtained as the union of the PP-matching, the SS-matching M, and the TT-matching M'. H is a KS-subgraph satisfying the condition of Lemma 4.2.3. Hence, H can be be colored using only the color of the PP-matching, colors s, s', and at most one new color such that no row except the top row sees more than 4 colors.

When H is colored as required, we are left with a sequence of matchings of even length such that consecutive matchings share a double color, and SS-matchings. These can be colored as shown in Section 4.2.3. □

Lemma 4.2.7 *Given a PP-matching, a chain of length* $2i$ *(*$i \geq 2$*) without parallel pre-colored edges, and* $i-1$ *SS-matchings, these* $n = 3i$ *matchings can be colored using at most* $n/3$ *new colors such that no row sees more than* $(4/3)n$ *colors.*

Proof: Pick an arbitrary SS-matching M with colors s and s'. If the chain contains a TT-matching M' with colors a and b such that none of its pre-colored edges is parallel to a pre-colored edge of M, the matchings M, M' and the PP-matching are a KS-subgraph that can be colored according to Lemma 4.2.3. We are left with a prefix and a suffix of the chain, one of odd length and one of even length. Combine either the first two matchings of the prefix with the last matching of the suffix, or the first matching of the prefix with the last two matchings of the suffix. Color the resulting KS-subgraph according to Lemma 4.2.3. Make the choice such that two (possibly empty) even-length sequences of TT-matchings remain. These sequences are such that consecutive matchings share a double color, and they can be combined with SS-matchings as shown in Section 4.2.3.

Now consider the case that all TT-matchings of the chain contain a pre-colored edge that is parallel to a pre-colored edge in M. In this case, the chain must actually be a 4-chain. Furthermore, the second and third matching of the 4-chain must each have exactly one pre-colored edge that is parallel to a pre-colored edge in M. Consider the KS-subgraph H obtained by combining the SS-matching M, the PP-matching, and the second matching of the chain. No row of this triplet except the top row sees 5 colors on pre-colored edges (otherwise, that row would see two colors from the TT-matching and two colors from the SS-matching, implying that both pre-colored edges in the TT-matching are parallel to a pre-colored edge in the SS-matching, a contradiction), and thus it can be colored as required according to Lemma 4.2.3. The remaining three matchings of the 4-chain again form a KS-subgraph that can be colored in this way. □

Lemma 4.2.8 *Given a cycle of length $2i+1$ ($i \geq 1$) and a chain of length $2j$ ($j \geq 2$), each without parallel pre-colored edges, and $i+j-1$ SS-matchings, these $n = 3(i+j)$ matchings can be colored using at most $n/3$ new colors such that no row sees more than $(4/3)n$ colors.*

Proof: Denote the cycle by C and the chain by N. Pick an arbitrary SS-matching M with single colors s and s'. Pick a matching M' with double colors a and b from the cycle such that none of its pre-colored edges is parallel to an edge colored s or s' in M. Such a matching M' exists because the cycle has length at least three.

Next, check whether N contains a TT-matching M'' such that none of its pre-colored edges is parallel to a pre-colored edge in M. If such a matching M'' does not exist, N must actually be a 4-chain and each of its two TT-matchings must have exactly one pre-colored edge that is parallel to a pre-colored edge in M. In this case, exchange both pairs of parallel pre-colored edges. Consequently, $N \cup M$ is either turned into a 3-chain and a 2-chain or into two 2-chains and a PP-matching. In the former case, the 3-chain is a KS-subgraph that can be colored according to Lemma 4.2.3, and the 2-chain can be combined with the odd cycle C as shown in Section 4.2.3. In the latter case, one 2-chain can be combined with the PP-matching and the other 2-chain can be combined with the odd cycle C (again as shown in Section 4.2.3).

Now consider the case that N contains a TT-matching M'' such that none of its pre-colored edges is parallel to a pre-colored edge in M. Let a' and b' be its double colors. Next, check whether any of the pre-colored edges colored a or b in C is parallel to a pre-colored edge colored a' or b' in N. This can be checked by testing the conditions $r(a) = r(a')$, $r(a) = r(b')$, $r(b) = r(a')$, $r(b) = r(b')$, $l(a) = l(a')$, $l(a) = l(b')$, $l(b) = l(a')$, and $l(b) = l(b')$. If one of these cases occurs, a pre-colored edge e_1 in C that is parallel to a pre-colored edge e_2 in N has been found. Exchanging these two pre-colored edges turns $C \cup N$ into a single chain N' of odd length. N' can be viewed as consisting of three subsequences: a prefix originating from N, an internal subsequence originating from C, and a suffix originating from N. Pre-colored edges within a subsequence are not parallel. Choose a pair of consecutive TT-matchings from N' by choosing M'' and either the matching before or after M'' in N' such that the pair consists of one matching from C and one matching from N. Hence, M and the two matchings from the pair represent a KS-subgraph satisfying the condition of Lemma 4.2.3 and can be colored as required. The remaining matchings in N' originating from C form an even-length subsequence of TT-matchings such that consecutive matchings share a double color. Hence, these matchings can be combined with SS-matchings as shown in Section 4.2.3. Now we are left with a prefix of N and a suffix of N, each consisting of at least one matching and one of them with even and the other with odd length. As in the proof of Lemma 4.2.7, combine either the first two matchings of the prefix with the last matching of the suffix, or the first matching of the prefix with the last two matchings of the suffix, and color the resulting KS-subgraph according to Lemma 4.2.3. Make the choice such that two (possibly empty) even-length sequences of TT-matchings remain. These sequences are such that consecutive matchings share a double color, and they can be combined with SS-matchings as shown in Section 4.2.3.

If no pre-colored edge colored a or b in C is parallel to a pre-colored edge colored a' or b' in N, we consider the triplet $H = M \cup M' \cup M''$. Remove the vertices x_0 and X_0 from H and add dummy edges $\{r(a), l(b')\}$, $\{r(a'), l(b)\}$, and $\{r(s), l(s')\}$. The obtained graph H' is again 3-regular and can be partitioned into three perfect matchings in linear time using the edge-coloring algorithm by Schrijver [Sch98]. If each of the matchings contains exactly one dummy edge, H can be split into an SS-matching with colors s and s', a TT-matching M_1 with colors a' and b, and a TT-matching M_2 with colors a and b'. Exchanging the new matchings for the old ones turns $C \cup N$ into a single chain N' of even length. N' consists of a prefix from N, followed by M_1, followed by an even number of matchings from C, followed by M_2, followed by a suffix from N. M_1 and its successor matching (with double colors b and, say, c) can be combined with an arbitrary SS-matching to obtain a KS-subgraph that can be colored according to Lemma 4.2.3. (Note that $r(a') \neq r(b)$ because the case $r(a') = r(b)$ has been handled above, and that $l(b) \neq l(c)$ because these edges originate from the same cycle.) M_2 and its predecessor in N' can be handled in the same way. What remains is a (possibly empty) even-length sequence of matchings from C, which can be combined with SS-matchings as shown in Section 4.2.3, and a prefix and a suffix of matchings originating from N, one of even length and the other of odd length, and these can be handled as above.

If one of the matchings obtained from H' does not contain a dummy edge, this matching gives a partitioning of H into gadget H_1 and matching H_2. In that case, we can color H_1 using only its old colors such that no row except the top row sees more than 3 colors (Lemma 4.2.4). H_2 can then be colored using a single new color. Therefore, H is colored such that no row except the top row sees more than 4 colors. When H has been colored, $C \setminus M'$ leaves an even sequence of TT-matchings behind (which can be combined with SS-matchings as shown in Section 4.2.3), and the prefix and suffix resulting from $N \setminus M''$ are such that one has even length and the other has odd length (so they can be handled as above). Note that the double colors that were reused to color H (by application of Lemma 4.2.4) are not reused in any other triplet. □

It is easy to see that the coloring methods used in the proofs of the four lemmas in this section are such that the time requirement per triplet is linear in the size of the triplet.

At this point, we have accomplished the goal of showing for the case $L = 3\ell$ that the uncolored edges of the bipartite graph G_v can be colored in time $O(T_{\mathrm{ec}}(\delta(v), L))$ such that the invariants are maintained. The next section shows how to modify the invariants and generalize the algorithm for the cases $L = 3\ell + 1$ and $L = 3\ell + 2$.

4.2.5 Dealing with $L = 3\ell + 1$ and $L = 3\ell + 2$

If the load L is not a multiple of three, the invariants introduced in Section 4.2.1 have to be modified. For the case $L = 3\ell + 1$, they become:

Invariant 1': The number of colors used in total is at most $\lceil (5/3)L \rceil = 5\ell + 2$.

Invariant 2': The number of colors used on a pair of directed edges with opposite directions is at most $4\ell + 2$.

Invariant 2' implies $S + D \leq 4\ell + 2$. Taking into account $S + 2D = 6\ell + 2$, we have $D \geq 2\ell$. Furthermore, if $D > 2\ell$ we can argue along the lines of Lemma 4.2.1, and hence we assume $D = 2\ell$ without loss of generality. $D = 2\ell$ implies $S = 2\ell + 2$.

Among all SS-matchings, PP-matchings, chains, and cycles of G_v, only SS-matchings and 2-chains contain more single colors than double colors. Since we have two more single colors than double colors altogether, there must either be one SS-matching or two 2-chains. If there is an SS-matching M, we consider the graph $G' = G_v \setminus M$. G' is 3ℓ-regular and has 2ℓ single colors and 2ℓ double colors on its pre-colored edges. Hence, G' can be colored with 4ℓ colors per row and 5ℓ colors altogether as shown in the previous sections. The matching M can be colored using any of its single colors. This results in at most $4\ell + 2$ colors per row and at most $5\ell + 2$ colors altogether, as required.

If there is no SS-matching, there must be two 2-chains P_1 and P_2. In this case, we consider the graph $G' = G_v \setminus (P_1 \cup P_2)$. Since $P_1 \cup P_2$ is a 4-regular subgraph with 4 single colors and 2 double colors, G' is $3(\ell - 1)$-regular and has $2(\ell - 1)$ single colors and $2(\ell - 1)$ double colors on its pre-colored edges. Again, G' can be colored with $4(\ell - 1)$ colors per row and $5(\ell - 1)$ colors altogether as shown in the previous sections. The four matchings of $P_1 \cup P_2$ can be colored without any new colors by coloring each of them with its single color. In addition to the colors used in G', these are at most 6 more colors per row and altogether. Hence, we have colored G_v using at most $4\ell - 4 + 6 = 4\ell + 2$ colors per row and at most $5\ell - 5 + 6 = 5\ell + 1$ colors altogether, as required. This concludes the discussion of the case $L = 3\ell + 1$.

For the case $L = 3\ell + 2$, we use the following invariants:

Invariant 1": The number of colors used in total is at most $\lceil (5/3)L \rceil = 5\ell + 4$.

Invariant 2": The number of colors used on a pair of directed edges with opposite directions is at most $4\ell + 4$.

Invariant 2" implies $S + D \leq 4\ell + 4$. As $S + 2D = 6\ell + 4$, we have $D \geq 2\ell$ and can again assume $D = 2\ell$ without loss of generality. Since $D = 2\ell$ implies $S = 2\ell + 4$, there must be a sufficient number of SS-matchings or 2-chains in G_v to account for the additional single colors. More precisely, there must either be two SS-matchings, one SS-matching and two 2-chains, or four 2-chains. In the following, we show that all three possibilities can be dealt with without violating the invariants.

If there are two SS-matchings M_1 and M_2, the graph $G' = G_v \setminus (M_1 \cup M_2)$ is 3ℓ-regular with 2ℓ single colors and 2ℓ double colors. G' can be colored with 4ℓ colors per row and 5ℓ colors altogether, and each of M_1 and M_2 can be colored with any of its single colors. The resulting coloring uses at most $4\ell + 4$ colors per row and $5\ell + 4$ colors altogether.

If there is one SS-matching M and two 2-chains P_1 and P_2, the graph $G' = G \setminus (M \cup P_1 \cup P_2)$ is $3(\ell - 1)$-regular with $2(\ell - 1)$ single colors and $2(\ell - 1)$ double colors. Color G' with $4(\ell - 1)$ colors per row and $5(\ell - 1)$ colors altogether. Color M with any of its single colors, and color each of the matchings in P_1 and P_2 with its respective single color. The number of colors used in $M \cup P_1 \cup P_2$ is 8, and we obtain a coloring with at most $4(\ell - 1) + 8 = 4\ell + 4$ colors per row and at most $5(\ell - 1) + 8 = 5\ell + 3$ colors altogether.

Finally, if there are no SS-matchings, we have four 2-chains P_i, $1 \leq i \leq 4$, and consider the graph $G' = G \setminus (P_1 \cup P_2 \cup P_3 \cup P_4)$. G' is $3(\ell - 2)$-regular with $2(\ell - 2)$ single colors and $2(\ell - 2)$ double colors, and it can be colored using $4(\ell - 2)$ colors per row and $5(\ell - 2)$ colors altogether. Coloring each of the remaining matchings in the 2-chains P_1, P_2, P_3, and P_4 with its respective single color, we need 12 more colors in addition to the colors used in G' and obtain a coloring with at most $4(\ell - 2) + 12 = 4\ell + 4$ colors per row and at most $5(\ell - 2) + 12 = 5\ell + 2$ colors altogether.

4.2.6 An Optimal Local Greedy Algorithm

In the previous sections we have shown that there are invariants that can be maintained at each coloring-extension substep by a greedy algorithm. The invariants imply that the number of colors used to color a given set of paths with load L is at most $\lceil (5/3)L \rceil$. The time requirement for one coloring-extension substep at a node v is $O(T_{\text{ec}}(\delta(v), L))$, because it is dominated by the time for partitioning the edges of the L-regular graph G_v into L perfect matchings. Consequently, the time requirement for the execution of the whole algorithm can be bounded by $\sum_{v \in V} O(T_{\text{ec}}(\delta(v), L)) = O(T_{\text{ec}}(N, L))$. Hence, we obtain the following theorem, which is the main result of this chapter.

Theorem 4.2.9 *Given a set of directed paths in a bidirected tree with N nodes such that the load on each directed edge is at most L, a coloring for the paths using at most $\lceil (5/3)L \rceil$ colors can be computed in time $O(T_{\text{ec}}(N, L))$.*

The exact running-time depends on the subroutine used to solve the unconstrained edge-coloring problem for regular bipartite multigraphs. Using the edge-coloring algorithm due to Cole and Hopcroft [CH82], a running-time of $O(NL(\log N + \log L))$ is achieved for path coloring in bidirected trees. Using Schrijver's algorithm [Sch98], a running-time of $O(NL^2)$ is obtained.

A *local greedy algorithm* for the path coloring problem is an algorithm that assigns colors to the paths touching a certain start node of the tree first and then visits the remaining nodes of the tree in depth-first search order (or any other order that ensures that the next node is adjacent to a previously visited node, i.e., any order derived from the general graph search procedure given in Section 2.1.2). At each node v it extends the existing partial coloring to include the paths touching v that have not been colored at a previous node. In addition, the algorithm must satisfy two requirements: First, the algorithm must be *greedy* in the sense that it never changes the color of a path once the color has been assigned to that path. Second, the algorithm must make *local* decisions in the sense that the only information about the paths touching the current node v that it takes into account during the coloring-extension substep is which edges incident to the current node these paths use.

It is easy to see that our algorithm belongs to this class of local greedy algorithms. All previous algorithms for path coloring in bidirected trees [MKR95, KP96, KS97] belong to this class as well. Local greedy algorithms are well-suited for distributed implementation of wavelength assignment in all-optical WDM networks: one node of the network can initiate the wavelength assignment by assigning wavelengths to the connection requests touching it, and then transfer control to its neighbors who can proceed to extend the wavelength assignment independently and in parallel.

For a given deterministic local greedy algorithm A and any positive integer L, an *adversary* can construct an instance of path coloring in a bidirected binary tree such that A uses at least $\lfloor (5/3)L \rfloor$ colors while an optimal solution uses only L colors [Jan97]. Hence, our algorithm is optimal within the class of local greedy algorithms with respect to the number of colors used in the worst case. It remains an interesting open problem whether a different approach to path coloring in bidirected trees can lead to an improved approximation algorithm.

4.2.7 Implementation and Experiments

The author has implemented the approximation algorithm for path coloring in bidirected trees presented in the preceding sections. The implementation was carried out in C++ [Str97] using the LEDA class library [MN95]. Schrijver's algorithm [Sch98] has been used as the edge-coloring subroutine, so the implemented path coloring algorithm has a worst-case running-time of $O(NL^2)$ for inputs consisting of a tree with N nodes and paths with maximum load L.

The source code of the implemented algorithm is structured as follows:

- Coloring KS-subgraphs according to Lemma 4.2.3: 3,500 lines of code
- Coloring gadgets according to Lemma 4.2.4: 1,000 lines of code
- Schrijver's edge-coloring algorithm [Sch98]: 600 lines of code
- Constrained bipartite edge-coloring algorithm (using the three subroutines above; including preprocessing of chains and cycles, and selection of triplets): 3,600 lines of code
- Main control structure of algorithm (visiting the nodes of the tree in dfs order and constructing the bipartite graphs): 500 lines of code

This adds up to 9,200 lines of code (228,000 bytes). The comparatively big code size is mainly caused by the huge number of similar cases that must be distinguished and treated in a slightly different way in the coloring routines. The user interface of the algorithm is provided by a function that takes as arguments a tree T and a list L of paths in T (the paths are presented as pairs of nodes). It returns the list of colors assigned to the paths by the algorithm.

In addition to the algorithm itself, a graphical user interface allowing easy demonstration of the algorithm has been implemented. The user can edit the tree network, enter a set of paths manually or have it created randomly, let the algorithm assign colors to the paths, and view the assignment. The LEDA classes `GraphWin` and `window` were convenient to use for this purpose and led to a quick completion of the code for this user interface.

Experiments with the implementation were conducted on a Pentium PC (MMX, 166 MHz) running the Linux operating system. The observed increase in running-time for randomly generated inputs of growing size was linear in N and slightly super-linear in L. The running-times for path coloring in a nearly balanced 5-ary tree with 100 nodes ranged from approximately 1 second for $L = 20$ to less than 12 seconds for $L = 110$. It should be noted that no attempt was made to tune the code to achieve faster running-times.

For sets of paths that create the same load on every edge of the tree, the number of wavelengths used by the algorithm was close to $(5/3)L$ in almost all experiments. This should not come as a surprise, because the algorithm does not try to use fewer than $(5/3)L$ colors if possible: if the existing partial coloring uses few colors and, consequently, the bipartite graph G_v contains many double colors, the algorithm gives away this advantage by splitting some of the double colors (Lemma 4.2.1) temporarily. For comparison, the simple greedy algorithm (mentioned in Section 4.2.1), which uses $2L - 1$ colors in the worst case, was also implemented. It turned out that the simple greedy algorithm used about 20 percent fewer colors than our algorithm on many inputs. Hence, it is meaningful to run both algorithms and use the better of the two colorings: in this way, the $\lceil (5/3)L \rceil$ worst-case guarantee can be combined with better average-case behavior. Additional heuristics might improve the number of colors used in the average case even further. The implementation of our algorithm and the experiments we conducted are described in more detail in [EJ98a].

Chapter 5

MaxPC and MaxPP in Bidirected Trees

For a given bidirected tree $T = (V, E)$, set P of directed paths in T, and number W of available colors, the *maximum path coloring* (MaxPC) problem is to compute a subset $P' \subseteq P$ and a W-coloring of P', and the *maximum path packing* (MaxPP) problem is to compute a subset $P' \subseteq P$ with maximum load W. For both problems, the goal is to maximize the cardinality of P'. For a given instance of MaxPC or MaxPP, we denote by P^* an arbitrary optimal solution. The MaxPC problem is equivalent to finding a maximum W-colorable subgraph in the conflict graph of the given paths. MaxPC applies to wavelength assignment in optical networks without wavelength converters; with this application in mind, we will use the terms *colors* and *wavelengths* interchangeably. MaxPP applies to all-optical networks with wavelength converters, to optical networks with space-division multiplexing, or to networks with circuit switching.

In this chapter, we present complexity results and approximation algorithms for MaxPC and MaxPP in bidirected trees. Note that an algorithm is a ρ-approximation algorithm for MaxPC or MaxPP if it always outputs a set P' whose cardinality is at least a $(1/\rho)$-fraction of the cardinality of an optimal solution. In Section 5.1, we begin by showing that MaxPP and MaxPC are equivalent if the given tree is a star and that both problems can be solved optimally in polynomial time in this case. Then, both problems are shown to be solvable optimally in polynomial time if the maximum degree Δ of the given tree network and the number W of available wavelengths are bounded by a constant. If either Δ or W can be arbitrarily large, both problems are proved $\mathcal{NP}$-hard.

In Section 5.2, we adapt the approximation algorithm for integral multicommodity flow in undirected trees due to Garg, Vazirani, and Yannakakis

[GVY93] and obtain a 2-approximation algorithm for MaxPP with arbitrary W in bidirected trees. If $W = 1$, i.e., only one wavelength is available, then MaxPC and MaxPP are equivalent to the maximum edge-disjoint paths problem, and in Section 5.3 we give a family of polynomial-time approximation algorithms with approximation ratio $5/3+\varepsilon$ for this case, where ε can be chosen arbitrarily small. In Section 5.4 we obtain, for MaxPC with arbitrary W, a 2.22-approximation for trees of arbitrary degree and a 1.58-approximation for trees whose degree is bounded by a constant.

5.1 Complexity of MaxPC and MaxPP

5.1.1 Stars

Recall that a star consists of a node v and a number of nodes $u_1,\ldots,u_\Delta$ that are adjacent to v but not adjacent to each other. A set of paths in a star with maximum load L can always be colored with L colors, because coloring paths with maximum load L in a star is equivalent to edge coloring a bipartite multigraph with maximum degree L (see Section 4.2.1). Therefore, the MaxPC and MaxPP problems are equivalent for stars. Furthermore, the problem of selecting a maximum number of paths from a set P of paths in a star such that the selected paths can be colored with W colors can be reduced to the b-matching problem in an undirected bipartite multigraph $G' = (V_1 \cup V_2, E')$ as follows. The construction of G' is the same as the construction of G_v in Section 4.2.1, but without the addition of dummy edges.

For each node u_i, $1 \leq i \leq \Delta$, V_1 contains two vertices x_i and v_{x_i}, and V_2 contains two vertices X_i and V_{x_i}. For each path from v to some u_i in P, add an edge $\{v_{x_i}, X_i\}$ to E'. For each path from some u_i to v in P, add an edge $\{x_i, V_{x_i}\}$ to E'. For each path from some u_i to some u_j in P, add an edge $\{x_i, X_j\}$ to E'. Observe that each path in P corresponds to one edge in G', and that two paths can be assigned the same color if and only if the corresponding edges in G' do not share an endpoint. Hence, the problem of selecting a maximum number of paths in P that can be colored with W colors is equivalent to the problem of selecting a maximum number of edges in E' that can be colored with W colors.

As the edges of a bipartite multigraph with maximum degree W can always be colored with W colors, it is sufficient to select a maximum cardinality subset E'' of E' such that the multigraph $G'' = (V_1 \cup V_2, E'')$ has maximum degree W. This is a special case of the capacitated b-matching problem (see Section 2.1.5), which can be solved in polynomial time [GLS88, pp. 257–259].

Hence, MaxPC and MaxPP can be solved optimally in polynomial time for stars.

Note that this solution to the MaxPC problem in stars extends to the weighted case, where each path in the input is associated with a benefit and the goal is to maximize the sum of the benefits of all accepted paths. This extension is achieved simply by using an algorithm for the weighted version of the capacitated b-matching problem, which can also be solved in polynomial time. Another possible extension is to solve MaxPP in the case that the number of available wavelengths can be different on different edges; it suffices to choose the values $b(v)$ for the nodes in G' appropriately.

5.1.2 W and Δ Bounded by a Constant

Now we assume that $T = (V, E)$ is a tree whose maximum degree Δ is bounded by a constant and that the number W of available wavelengths is also bounded by a constant. We show that MaxPC and MaxPP can both be solved optimally in polynomial time in this case.

First, we present the algorithm for MaxPC. Recall that L denotes the maximum load of a directed edge in T. In the following, a *colored set* is a set of paths together with a coloring for the paths. The number of paths in a colored set M is denoted by $|M|$. We use a bottom-up computation to compute values $f(v, P_v) \in \mathbb{N}$ for all $v \in V$ and for all $P_v \in \mathcal{P}_v$. Here, $\mathcal{P}_v$ is the set of all colored subsets P_v of P such that the following two conditions hold.

1. P_v contains only paths touching v and its parent $p(v)$.
2. The paths in P_v are colored using only colors from $\{1, \ldots, W\}$ such that intersecting paths receive different colors.

The value $f(v, P_v)$ will represent the maximum number of paths contained in the subtree rooted at v such that these paths together with the paths in P_v can be colored using W colors (without changing the colors of the paths in P_v).

The values $f(v, P_v)$ can be computed as follows. First, $f(v, P_v)$ is initialized to zero for every node v and all valid colored sets $P_v \in \mathcal{P}_v$. For the leaves of the tree, no further computation is necessary; they are considered ready. An internal node v of T is processed only after all its children are ready, i.e., after the values $f(w, P_w)$ for all its children w and all colored sets $P_w \in \mathcal{P}_w$ are available. While the node v is processed, enumerate all possible colored sets Q_v containing paths touching v that are colored using at most W colors. For a neighbor w of v, denote by $Q_v[w]$ the colored subset of Q_v

that contains all colored paths touching v and w. For every colored set Q_v, the values $f(v_i, Q_v[v_i])$ for all children v_i of v and the value $|Q_v \setminus Q_v[p(v)]|$ are added up. If the result is greater than the previous value for $f(v, Q_v[p(v)])$, that value is updated.

At the root node u with children $u_1, \ldots, u_k$, determine for all colored sets Q_u the term $|Q_u| + \sum_{i=1}^{k} f(u_i, Q_u[u_i])$. The maximum value m (taken over all colored sets Q_u) of this term gives exactly the maximum number of paths in P that can be colored with W colors. By storing for each pair (v, P_v) which colored set Q_v yielded the maximum value of $f(v, P_v)$, the algorithm can output not only the value m, but also m paths together with a valid W-coloring in the end.

The running-time of the algorithm is bounded by a polynomial in L and $|V|$ and, therefore, polynomial in the size of the input. In fact, the worst-case running-time for processing a node v of T can be estimated as follows. There are at most $\binom{L}{0} + \binom{L}{1} + \cdots + \binom{L}{W} \leq WL^W$ possibilities to choose up to W paths going through an edge e with load $\leq L$. As v has $\leq \Delta$ outgoing and $\leq \Delta$ incoming edges, at most $(WL^W)^{2\Delta} = L^{O(1)}$ different sets of paths touching v must be considered. Each of these contains at most $2\Delta W$ paths; therefore there are at most $W^{2\Delta W} = O(1)$ possible colorings for each such set. Hence, for each node v only a polynomial number of colored sets Q_v must be considered, and each of them can be processed in polynomial time. As there are only $|V|$ nodes to be processed, the overall running-time is bounded by a polynomial in L and $|V|$.

In order to solve the MaxPP problem in the case of bidirected trees with bounded degree and constant W, a very similar bottom-up computation can be used. The details are left to the reader.

Note that the algorithms for MaxPP and MaxPC in bidirected trees of bounded degree with bounded number of wavelengths, like the algorithm for stars, extend to the weighted variants of the MaxPC and MaxPP problems: It suffices to redefine the value $f(v, P_v)$ to represent the maximum total weight of paths contained in the subtree rooted at v such that these paths together with the paths in P_v can be colored using W colors. Furthermore, the algorithms can also be generalized to the case where the set of available wavelengths varies from link to link and to the case where wavelength converters with limited wavelength conversion are allowed. In addition, variants of the algorithm give exact algorithms for integral multicommodity flow in bidirected or undirected trees of bounded degree, if the edge capacities are bounded by a constant.

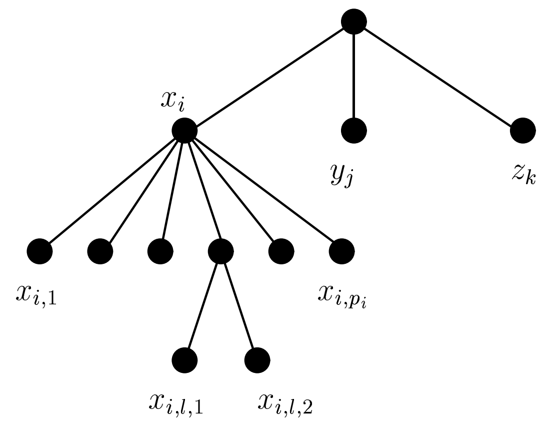


Figure 5.1: Construction of the tree for the 3D-matching problem

5.1.3 Arbitrary Maximum Degree Δ

Now we consider the case that the maximum degree of the tree T can be arbitrary while the number W of available wavelengths is bounded by a constant. If the constant is 1, i.e., if $W = 1$, the MaxPC problem is equivalent to the problem of determining a maximum cardinality subset P' of P such that the paths in P' are edge-disjoint, i.e., to the maximum edge-disjoint paths problem. This problem can be shown $\mathcal{NP}$-hard by a reduction adapted from the proof for the $\mathcal{NP}$-hardness of integral multicommodity flow in undirected trees with edge capacities one or two from [GVY93, Theorem 6].

Theorem 5.1.1 *The maximum edge-disjoint paths problem is $\mathcal{NP}$-hard for bidirected trees.*

Proof: We reduce the $\mathcal{NP}$-complete 3D-matching problem [GJ79] to the maximum edge-disjoint paths problem for bidirected trees. An instance of the 3D-matching problem consists of three disjoint sets X, Y, Z with $|X| = |Y| = |Z| = n$ and a set of triples $S \subseteq \{\, (x_i, y_j, z_k) \mid x_i \in X, y_j \in Y, z_k \in Z \,\}$. The goal is to decide whether S contains n disjoint triples.

Given an instance of the 3D-matching problem, we construct a bidirected tree T of depth three as follows. The root of T has $3n$ children: one for each $x_i \in X$, one for each $y_j \in Y$, and one for each $z_k \in Z$. Each node x_i has p_i children $x_{i,1},\ldots,x_{i,p_i}$, where p_i is the number of occurrences of x_i in S. Each $x_{i,j}$ has two children $x_{i,j,1}$ and $x_{i,j,2}$. See Figure 5.1 for a sketch of this construction; note that only a small subset of the nodes of the tree is actually shown there, and that pairs of oppositely directed edges are depicted as undirected edges for the sake of simplicity.

Now, we create a set P of paths in T. Number the occurrences of x_i in the triples of S from 1 to p_i arbitrarily. For every triple in S we add three paths to P. Let triple (x_i, y_j, z_k) contain the l-th occurrence of x_i in S. Then we add a path from $x_{i,l,1}$ to y_j, a path from z_k to $x_{i,l,2}$, and a path from $x_{i,l,1}$ to $x_{i,l,2}$.

We claim that S contains n disjoint triples if and only if P contains a subset P' of at least $|S| + n$ edge-disjoint paths. Assume that S contains n disjoint triples $(x_{i_1}, y_{j_1}, z_{k_1})$,...,$(x_{i_n}, y_{j_n}, z_{k_n})$. Let triple $(x_{i_t}, y_{j_t}, z_{k_t})$ contain the l_t-th occurrence of x_{i_t}. Then the following $|S| + n$ paths form a set P' of edge-disjoint paths: for each t, $1 \leq t \leq n$, choose the path from $x_{i_t,l_t,1}$ to y_{j_t}, the path from z_{k_t} to $x_{i_t,l_t,2}$, and $p_{i_t} - 1$ paths from $x_{i_t,l,1}$ to $x_{i_t,l,2}$ for $l \in \{1, \ldots, p_{i_t}\} \setminus \{l_t\}$.

Conversely, assume that there is a subset P' of P containing at least $|S| + n$ edge-disjoint paths. Note that P' can contain at most one path entering the subtree rooted at x_i from above and at most one path leaving the subtree rooted at x_i. The only possibility for P' to contain more than p_i paths using edges of the subtree rooted at x_i is to contain one path from $x_{i,l,1}$ to $x_{i,l,2}$ for $p_i - 1$ values of l and two additional paths, one from $x_{i,l_i,1}$ to some y_j and one from some z_k to $x_{i,l_i,2}$. In that case, P' contains $p_i + 1$ paths using edges of the subtree rooted at x_i. The only way for P' to contain at least $|S| + n$ paths is that P' contains exactly $p_i + 1$ paths using edges of the subtree rooted at x_i for every i, $1 \leq i \leq n$. In that case, the triples $(x_i, y_j, z_k) \in S$ containing the l_i-th occurrence of x_i form a set of n disjoint triples. □

In fact, the statement of the theorem can be strengthened (as observed in [GVY93]) as follows: Note that the maximum 3D-matching problem is MAX SNP-complete [Pap94] and $\mathcal{APX}$-complete [MPS98] even if each of the x_i, y_i, and z_i may occur in at most a constant number of triples in S [Kan91]. The reduction used in the proof of Theorem 5.1.1 is an L-reduction [PY91] and an AP-reduction [CKST95] if we reduce from this bounded variant of the 3D-matching problem. Hence, we can conclude that the maximum edge-disjoint paths problem in bidirected trees is MAX SNP-hard and $\mathcal{APX}$-hard. This implies that there is no polynomial-time approximation scheme for the problem unless $\mathcal{P} = \mathcal{NP}$.

So we know that MaxPC is $\mathcal{NP}$-hard (and even $\mathcal{APX}$-hard) already for $W = 1$. Furthermore, MaxPC is also $\mathcal{NP}$-hard if $W = c$ for any fixed $c > 1$. To see this, note that the maximum edge-disjoint paths problem can be reduced to MaxPC with $W = c$ by simply adding $c - 1$ dummy paths of length one for every directed edge of the tree; this effectively reduces the number of wavelengths available for the original paths to one.

5.1.4 Arbitrary Number W of Wavelengths

If the number of available wavelengths is part of the input and is not bounded by a constant, we show that MaxPC and MaxPP are $\mathcal{NP}$-hard already for binary trees. Therefore, bounding the maximum degree by a constant at least three does not help in this case. (The case $\Delta = 2$, i.e., T is a chain, can be solved optimally in polynomial time for arbitrary W.)

The $\mathcal{NP}$-hardness of MaxPC follows easily from the $\mathcal{NP}$-hardness proof for path coloring in binary trees given in Theorem 3.1.9 in Section 3.1.2. There it is shown that it is $\mathcal{NP}$-complete to decide whether a set P of paths in a binary tree T with maximum load L can be colored using L wavelengths. However, P can be colored using L wavelengths if and only if the output of MaxPC on input T, P and $W = L$ is P (and a coloring for P). Hence, MaxPC is $\mathcal{NP}$-hard already for binary trees if W can be arbitrary.

Next, we show that the MaxPP problem is also $\mathcal{NP}$-hard already for binary trees if W can be arbitrary. We will prove this using a modification of the proof of Theorem 5.1.1 in the preceding section. First, we present a helpful lemma. Call a path consisting of a single directed edge a *one-edge path*. Informally, the lemma states that it is always the right decision to accept as many one-edge paths as possible.

Lemma 5.1.2 *Let an instance of MaxPP be given by a bidirected tree $T = (V, E)$, a number W of available wavelengths, and a set P of directed paths in T. For every directed edge $e \in E$, let $s(e)$ denote the number of one-edge paths in P consisting only of edge e. Then there is an optimal solution $P^* \subseteq P$ for this instance of MaxPP such that P^* contains, for each edge e, exactly* $\min\{s(e), W\}$ *one-edge paths from P that consist only of edge e.*

Proof: Let Q be an optimal solution that contains, for some edge e, strictly less than $\min\{s(e), W\}$ paths that consist only of edge e. Hence, there must be a path $p \in P \setminus Q$ consisting only of edge e. If the paths in Q created load less than W on edge e, we could add p to Q without exceeding the load W on any edge, a contradiction to the fact that Q is an optimal solution. Hence, edge e has load exactly W in Q, and Q contains at least one path p' through e that does not consist only of edge e. Replacing p' by p in Q gives another optimal solution Q' with one additional one-edge path. This process can be iterated until the condition of the lemma is satisfied. $\square$

We remark that this lemma holds also for the MaxPC problem.

Theorem 5.1.3 *MaxPP is $\mathcal{NP}$-hard for binary trees, if W can assume arbitrary values.*

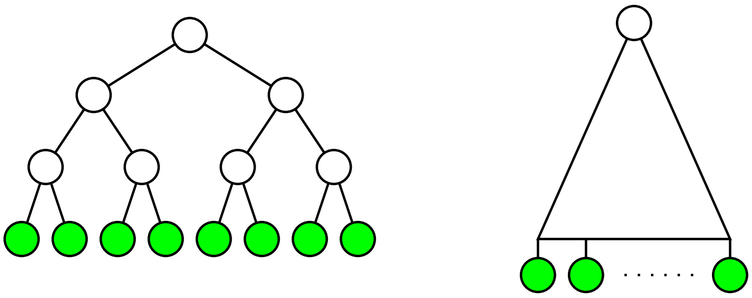


Figure 5.2: A complete binary tree (left) is sketched as a triangle (right)

Proof: As in the proof of Theorem 5.1.1, we reduce the $\mathcal{NP}$-hard 3D-matching problem to the MaxPP problem. Here, however, we are only allowed to construct an instance of MaxPP using paths in a *binary* tree, and we will use a larger number W of available wavelengths and a larger depth of the tree in order to compensate this restriction.

Let an instance S of the 3D-matching problem be given as in the proof of Theorem 5.1.1. Again, denote by p_i the number of occurrences of x_i in triples in S, $1 \le i \le n$.

Let $d = \lceil \log n \rceil$. Note that a complete binary tree of depth d has $2^d \ge n$ leaves. We construct a binary tree T as follows (see Figure 5.3; for the sake of simplicity, complete binary trees are sketched as triangles, as indicated in Figure 5.2). Start with a complete binary tree with four leaves. Call its root r, and call its leftmost three leaves x, y and z, respectively. Identify each of x, y and z with the root of a distinct complete binary tree of depth d. Call the leftmost n leaves of x's subtree $x_1, \ldots, x_n$, those of y's subtree $y_1, \ldots, y_n$, and those of z's subtree $z_1, \ldots, z_n$. Now identify each of $x_1, \ldots, x_n$ with the root of a distinct complete binary tree such that x_i's subtree has depth $\lceil \log p_i \rceil$ for $1 \le i \le n$ and, therefore, at least p_i leaves. Call the leftmost p_i leaves of x_i's subtree $x_i^1, \ldots, x_i^{p_i}$. Finally, attach to each node x_i^j, where $1 \le i \le n$ and $1 \le j \le p_i$, two children a_i^j and b_i^j. This construction results in a binary tree T whose size is polynomial in n and $|S|$.

We set the number of available wavelengths for the MaxPP instance to $W = n$. Finally, the set P of paths in T is created. Number the occurrences of x_i in the triples of S from 1 to p_i arbitrarily. For every triple in S we add $n+2$ paths to P. Let triple (x_i, y_j, z_k) contain the l-th occurrence of x_i in S. Then we add a path from a_i^l to y_j, a path from z_k to b_i^l, and n paths from a_i^l to b_i^l. The former two paths are called *long paths*, the latter n paths *short paths*. Note that there are exactly $2|S|$ long paths and $n|S|$ short paths altogether. Finally, add $n-1$ one-edge paths on each of the edges $(x_i, p(x_i))$, $(p(x_i), x_i)$, $(p(y_i), y_i)$ and $(z_i, p(z_i))$ for $1 \le i \le n$. These

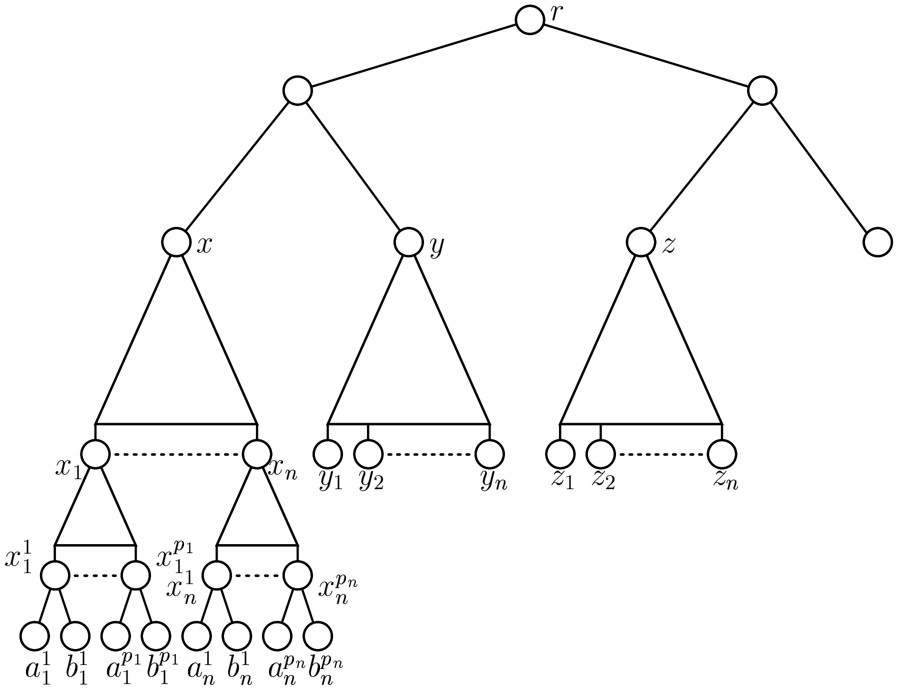


Figure 5.3: Reduction from 3D-matching problem to MaxPP in binary trees

are $4n(n-1)$ one-edge paths altogether.

We claim that S contains n disjoint triples if and only if P contains a subset P' of at least $n|S|+4n(n-1)+n$ paths with maximum load W. Assume that S contains n disjoint triples, and let these triples be $(x_{i_1}, y_{j_1}, z_{k_1}), \ldots,$ $(x_{i_n}, y_{j_n}, z_{k_n})$. Let the t-th triple, triple $(x_{i_t}, y_{j_t}, z_{k_t})$, contain the l_t-th occurrence of x_{i_t}. Then the following $n|S|+4n(n-1)+n$ paths form a set $P' \subseteq P$ of paths with maximum load $W=n$: for each t, $1 \le t \le n$, choose the path from $a_{i_t}^{l_t}$ to y_{j_t}, the path from z_{k_t} to $b_{i_t}^{l_t}$, $n-1$ short paths from $a_{i_t}^{l_t}$ to $b_{i_t}^{l_t}$, and n short paths from $a_{i_t}^{l}$ to $b_{i_t}^{l}$ for each of the $p_{i_t}-1$ remaining values of l, i.e., for $l \in \{1, \ldots, p_{i_t}\} \setminus \{l_t\}$. Finally, the $4n(n-1)$ one-edge paths can be added without increasing the maximum load above W.

Conversely, assume that there is an optimal subset P' of P containing at least $n|S|+4n(n-1)+n$ paths with maximum load W. By Lemma 5.1.2, we can assume that P' contains all the $4n(n-1)$ one-edge paths from P. Hence, it must contain at least $n|S|+n$ additional paths, each of which uses edges in the subtree rooted at some x_i. Note that the one-edge paths create load $W-1$ on the edges $(x_i, p(x_i))$, $(p(x_i), x_i)$, $(p(y_i), y_i)$ and $(z_i, p(z_i))$ for $1 \le i \le n$. Therefore, P' can then contain at most one long path entering the subtree rooted at x_i, at most one long path leaving the subtree rooted at x_i, at most one long path reaching y_i from its parent, and at most one long path running from z_i to its parent. The only possibility for P' to contain more than np_i paths using edges of the subtree rooted at x_i is to contain n paths from a_i^l to b_i^l for $p_i - 1$ values of l and $n+1$ additional paths for the remaining value of l, say l_i: one from $a_i^{l_i}$ to some y_j, one from some z_k to $b_i^{l_i}$, and $n-1$ paths from $a_i^{l_i}$ to $b_i^{l_i}$. In that case, P' contains $np_i + 1$ paths using edges of the subtree rooted at x_i. The only way for P' to contain at least $n|S| + n$ paths using edges in the subtrees rooted at some x_i is that P' contains exactly $np_i + 1$ paths using edges of the subtree rooted at x_i for every i, $1 \leq i \leq n$. In that case, the triples $(x_i, y_j, z_k) \in S$ containing the l_i-th occurrence of x_i form a set of n disjoint triples. □

5.2 Approximating MaxPP

We propose a greedy algorithm for finding a large subset P' of P with maximum load W that works as follows. Initially, set $P' = \emptyset$. Then process all nodes of the tree in order of non-increasing levels (i.e., bottom-up). When processing node v, consider the paths whose lca is v in arbitrary order. Insert each such path into P' if this does not increase the maximum load of P' above W. In the end, output P'. Note that this algorithm is an adaptation of the 2-approximation algorithm for integral multicommodity flow in undirected trees due to Garg, Vazirani, and Yannakakis [GVY93]; their algorithm also works bottom-up, considers the demands with lca v one by one, and satisfies as much of each demand as possible without violating the edge capacities.

Theorem 5.2.1 *The algorithm outputs a subset P' of P with maximum load W and with cardinality at least $|P^*|/2$, where P^* is a maximum cardinality subset of P with maximum load at most L. Hence, the algorithm is a 2-approximation algorithm for MaxPP in bidirected trees.*

Proof: First, introduce some notation. Consider the state of the algorithm after processing i paths. The set of all paths can be partitioned into four subsets:

- a set A_i of paths that have been inserted into P'
- a set R_i of paths that have been processed, but not inserted into P' because this would have exceeded maximum load W
- a set A_i' of paths that have not yet been processed, but will be inserted into P' later on

- a set R'_i of paths that have not yet been processed and that will not be inserted into P' later on

Fix an arbitrary optimal solution P^*. We use induction on i to prove the following claim:
Claim: After i paths are processed, P^* can be partitioned into two subsets C_i and D_i such that

(a) $|A_i| \geq |C_i|/2$,

(b) the paths in D_i have not yet been processed by the algorithm,

(c) the paths in $A_i \cup D_i$ have maximum load at most W, and

(d) the paths in C_i that have not yet been processed will not be included into P' by the algorithm.

Initially, the claim holds for $C_0 = \emptyset$ and $D_0 = P^*$. If the claim holds after all $|P|$ paths are processed by the algorithm, we have $C_{|P|} = P^*$, $D_{|P|} = \emptyset$, and, by (a), $|P'| = |A_{|P|}| \geq |P^*|/2$.

Assume that the claim holds after $i-1$ paths are processed by the algorithm. We must show that the claim holds also after the i-th path p has been processed. Let v be the lca of p. The following cases can occur.

Case 1: The algorithm does not insert p into P'. In this case, including p in P' would have exceeded the maximum load W, and hence $p \notin D_{i-1}$. Instead, $p \in C_{i-1}$ or $p \notin P^*$. In either case, the claim holds for $C_i = C_{i-1}$ and $D_i = D_{i-1}$.

Case 2: The algorithm inserts p into P', and $p \in P^*$. In this case, (d) implies $p \notin C_{i-1}$ and, therefore, $p \in D_{i-1}$. Now set $C_i = C_{i-1} \cup \{p\}$ and $D_i = D_{i-1} \setminus \{p\}$. The cardinality of A_i and of C_i has increased by one, hence (a) is satisfied. Obviously, (b) and (d) are satisfied as well. Condition (c) holds, because $A_i \cup D_i = A_{i-1} \cup D_{i-1}$. Therefore, the claim holds.

Case 3: The algorithm inserts p into P', and $p \notin P^*$. In this case, the cardinality of A_i increases by one, and we show that conditions (a) to (d) can be satisfied by moving at most two paths from D_{i-1} to C_{i-1} to obtain C_i and D_i. Obviously, doing so will necessarily satisfy conditions (a) and (b). However, we must choose the paths to be moved from D_{i-1} to C_{i-1} carefully in order to satisfy conditions (c) and (d).

If the set $A_i \cup D_{i-1}$ has maximum load at most W, no path needs to be moved from D_{i-1} to C_{i-1} and we can choose $C_i = C_{i-1}$ and $D_i = D_{i-1}$. Otherwise, adding the path p to $A_{i-1} \cup D_{i-1}$ creates load $W+1$ on some edge(s) in the subtree rooted at v. Assume that p uses the edges (v_i, v) and (v, v_j) for some children v_i and v_j of v. (If p begins or ends at v, a similar reasoning can be applied.) All paths from D_{i-1} that intersect p must also use the edge (v_i, v) or (v, v_j) (or both), because the paths in D_{i-1} have not yet been processed by the algorithm and, consequently, the levels of their lcas are not larger than the level of v. Let D_p be the subset of D_{i-1} of all paths that intersect p. The set $A_{i-1} \cup D_p \cup \{p\}$ has maximum load $W + 1$.

More precisely, that set has load $W + 1$ on some edge(s) of p and load $\leq W$ on all other edges of T. It is possible to choose at most two paths p_1, p_2 from D_p such that $A_{i-1} \cup (D_p \setminus \{p_1, p_2\}) \cup \{p\}$ has maximum load at most W. For example, if e_1 and e_2 are the lowest upward resp. downward edges of T that are contained in $W + 1$ paths of $A_{i-1} \cup D_p \cup \{p\}$, one can choose an arbitrary path in $D_p \cap R'_i$ containing e_1 as p_1 and an arbitrary path in $D_p \cap R'_i$ containing e_2 as p_2. Such paths must exist. As p_1 contains the subpath from e_1 to v and p_2 contains the subpath from v to e_2, the set $A_{i-1} \cup (D_p \setminus \{p_1, p_2\}) \cup \{p\}$ has maximum load W. Therefore, the set $A_{i-1} \cup (D_{i-1} \setminus \{p_1, p_2\}) \cup \{p\}$ has maximum load W as well.

We observe that the sets $C_i = C_{i-1} \cup \{p_1, p_2\}$ and $D_i = D_{i-1} \setminus \{p_1, p_2\}$ satisfy (c) and (d): $A_i \cup D_i$ has maximum load W as shown above, and the paths that have been newly inserted into C_i were taken from R'_i and will, therefore, not be inserted into P' later on. This concludes Case 3.

Hence, it has been shown that the claim holds after i paths are processed assuming that it was satisfied after $i - 1$ paths were processed. Therefore, the claim still holds after all paths are processed. □

The proof we have just given for the theorem is elementary and self-contained. It is also possible to derive the theorem from the fact that the approximation algorithm for integral multicommodity flow in undirected trees given in [GVY93] has approximation ratio 2. To see this, observe that the approximation algorithm from [GVY93] works also for bidirected trees: the proof of approximation ratio 2 using primal-dual arguments carries over without difficulties. Then, note that the only difference between the multicommodity flow problem in [GVY93] and the MaxPP problem is that with the MaxPP problem, no commodity can have flow greater than one.

Our greedy algorithm for MaxPP in a bidirected tree T with edge capacity W behaves like the algorithm from [GVY93] in a slightly extended tree T': for each path (commodity) p from a node u to a node w, add two new nodes u_p and w_p, add two unit capacity edges (u_p, u) and (w, w_p), and replace p by a path from u_p to w_p. It is easy to see that the multicommodity flow problem in the resulting tree T' is equivalent to the MaxPP problem in the original tree, and that our greedy algorithm produces the same solution as the algorithm from [GVY93] on this instance.

5.3 Approximation Algorithms for $W = 1$

For $W = 1$ the MaxPC and MaxPP problems are equivalent. If the degree of the tree is bounded by a constant, an optimal solution can be obtained in polynomial time (Section 5.1.2), but if the degree is arbitrary, the problem is $\mathcal{NP}$-hard (Section 5.1.3). The algorithm from the previous section achieves approximation ratio 2 also for $W = 1$. In this section we present an improved algorithm for MaxPC and MaxPP with $W = 1$, i.e., for the maximum edge-disjoint paths problem. More precisely, we present, for any given $\varepsilon > 0$, a polynomial-time $(5/3+\varepsilon)$-approximation algorithm. The main idea that leads to this improvement is to consider all paths with the same lca simultaneously instead of one by one.

Fix any $\varepsilon > 0$. Let an instance of the maximum edge-disjoint paths problem be given by a bidirected tree T and a set P of directed paths in T. Denote by P' the solution computed by the approximation algorithm and by P^* an optimal solution for the given instance. In the first pass, our algorithm processes the nodes of the tree bottom-up; at a node v, it tries to decide for the paths with lca v whether they should be accepted or rejected. Let P_v denote the subset of all paths $(u, w) \in P$ with $lca(u, w) = v$ that do not intersect any of the paths that have been accepted by the algorithm at a previous node and that do not use any edges that have been *reserved* or *fixed* by the algorithm (see below). For the sake of simplicity, we can assume without loss of generality that we have $u \neq v \neq w$ for all paths $(u, w) \in P_v$; otherwise, we could add an additional child to v for each path in P_v starting or ending at v and make the path start or end at this new child instead. We say that two paths (u_1, w_1) and (u_2, w_2) with lca v are *equivalent* if they use the same two edges incident to v, i.e., if their top edges are the same. If all paths in a set Q of paths with lca v are equivalent, we say that *v has only one equivalence class of paths in Q*. If v has only one equivalence class of paths in P_v, we say that v has only one equivalence class.

As mentioned above, the algorithm processes the nodes of T in order of non-increasing levels (i.e., bottom-up) in the first pass. When it processes node v, it tries to determine for the paths in P_v whether they should be included in P' (these paths are called *accepted*) or not (these paths are called *rejected*). Sometimes, however, the algorithm cannot make this decision right away. In these cases the algorithm will leave some paths in an intermediate state and resolve them later on. The possibilities for paths in such intermediate states are

- undetermined paths,
- groups of deferred paths,

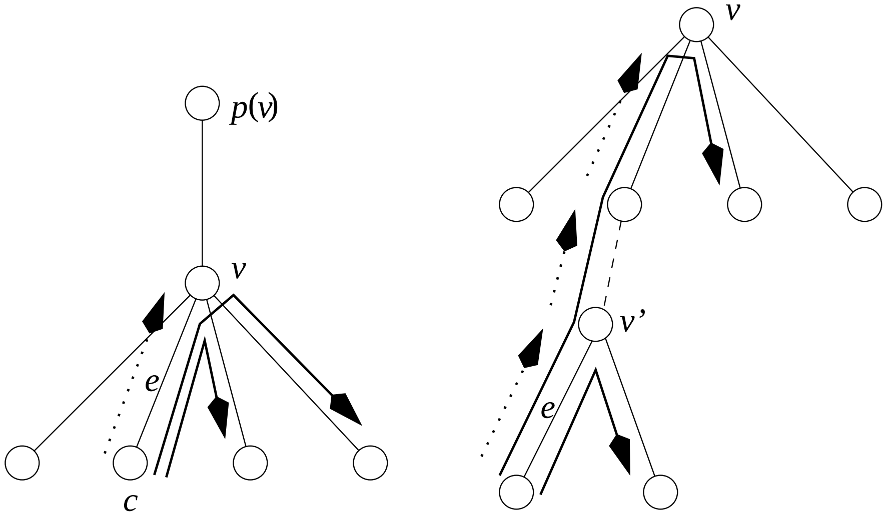


Figure 5.4: Possible configurations of a group of deferred paths

- groups of exclusive paths, and
- groups of 2-exclusive paths.

We refer to undetermined paths, groups of exclusive paths, and groups of 2-exclusive paths (but not groups of deferred paths) as *unresolved* paths.

In the following we give explanations regarding these possible groups of paths in intermediate states. First, the algorithm will sometimes leave a single path p of P_v in an undetermined state. If v has only one equivalence class of paths, accepting a path $p \in P_v$ might cause the algorithm to miss the chance of accepting two paths with an lca of smaller level later on. Hence, the algorithm could at best achieve a 2-approximation. On the other hand, if the algorithm rejects all paths in P_v whenever v has only one equivalence class of paths, it will not accept any paths in an instance where all nodes have at most one equivalence class of paths. Therefore, instead of accepting or rejecting the paths in P_v right away, the algorithm picks one of them and makes it an *undetermined path*. All other paths in P_v, if any, are rejected, and the undetermined path will be accepted or rejected at a later node.

A second situation in which the algorithm does not accept or reject all paths in P_v right away is sketched in Figure 5.4 (left). (Here and in the following, pairs of oppositely directed edges are drawn as undirected edges in all figures.) In this situation, the algorithm decides to accept one of several intersecting paths from P_v, but it defers the decision which one of them to accept. The intersecting paths are called a *group of deferred paths*. All paths in a group of deferred paths use the same edge incident to v and to a child c of v. In the figure, this is the edge (c, v). (Analogous arguments apply to the case where the edge (v, c) is shared by the deferred paths.) Furthermore, each deferred path uses also an edge (v, c') connecting v and a child $c' \neq c$, and not all of the deferred paths use the same such edge. If the algorithm decides to defer such a group of paths, it marks the edge (c, v) as *reserved* (assuring that no path accepted at a node processed after v can use the edge), but leaves all edges (v, c') for children $c' \neq c$ available. The reserved edge is indicated by a dotted arrow in Figure 5.4. The motivation for introducing groups of deferred paths is as follows: first, the reserved edge blocks at most one path with lca of smaller level that could be accepted in an optimal solution; second, no matter which path using the edge $(p(v), v)$ is accepted at a node processed after v, that path uses at most one of the edges (v, c'), and as there is still at least one deferred path that does not use that particular edge (v, c'), the algorithm can pick such a deferred path in a second pass which proceeds top-down. When processing later nodes during the first pass, the algorithm can actually treat the group of deferred paths as a single accepted path using only the reserved edge of the deferred paths.

In some cases the algorithm will create a group of deferred paths from intersecting paths with two different least common ancestors v and v', where v is an ancestor of v'. Such a case is depicted in Figure 5.4 (right). Assume that the edge e incident to v' that is shared by the paths in such a group of deferred paths is directed towards the root. (The case that e is directed towards the leaves is symmetrical.) Then the algorithm marks all edges on the path from e to v as reserved. These reserved edges are indicated by dotted arrows in Figure 5.4. (Actually, it would be sufficient to reserve only the top edge among these, i.e., the edge incident to v, because every path with lca of smaller level than v intersecting any of the reserved edges must intersect this top edge as well.) Again, no matter which paths with lca of smaller level than v not intersecting the reserved edges are accepted by the algorithm later on, there is still one of the deferred paths that can be accepted.

A *group of exclusive paths* is sketched in Figure 5.5 (left). Such a group consists of one path q (called the *lower path*) contained in the subtree rooted at a child c of v and one path p (called the *higher path*) with lca v that intersects q. At most one of the two paths can be accepted, but if the algorithm picks the wrong one this choice may cause the algorithm to accept only one path while the optimal solution would accept the other path plus one or two additional paths. Hence, the algorithm defers the decision which path to accept until a later node. For now, it only marks the top edge of path q that is intersected by p as fixed (indicated by a dotted arrow in Figure 5.5). A group of exclusive paths has the following property.

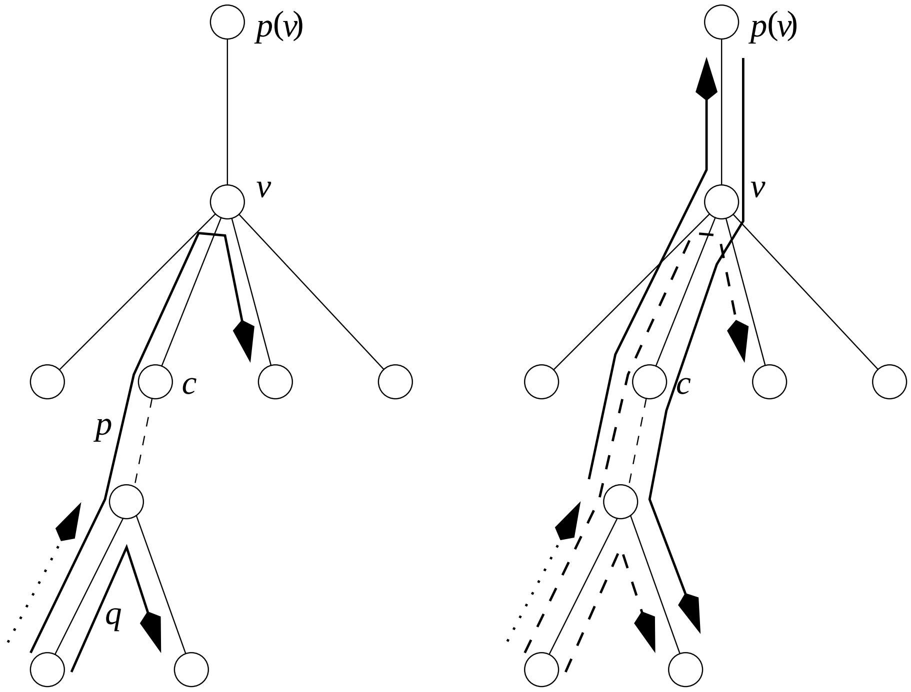


Figure 5.5: Possible configuration of a group of exclusive paths (left), and situation in which both exclusive paths are blocked (right)

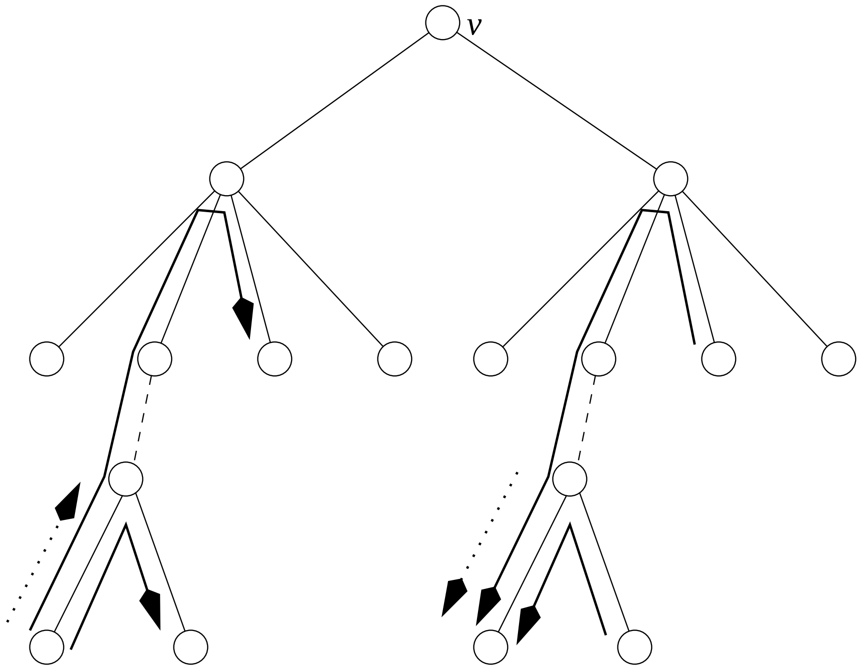


Figure 5.6: Group of 2-exclusive paths consisting of a pair of independent groups of exclusive paths

Property (E): As long as at most one path touching v but not using the fixed edge is accepted at a later node, either p or q can still be accepted. Only when two paths touching v are accepted at a later node, they may block p and q from being accepted.

See the right-hand side of Figure 5.5 for an example of how two paths accepted at a later node can block both exclusive paths. While processing later nodes, the algorithm will try to avoid this whenever possible.

The last types of unresolved paths are sketched in Figures 5.6 and 5.7. These *groups of* 2*-exclusive paths* consist of a set of four paths at most two of which can be accepted. More precisely, the first possibility for a group of 2-exclusive paths is to consist of two independent groups of exclusive paths (Figure 5.6), i.e., of two groups of exclusive paths such that the fixed edge of one group is directed towards the root and the fixed edge of the other group is directed towards the leaves. Furthermore, the two groups must either be contained in disjoint subtrees (as shown in Figure 5.6), or only their lower paths are contained in disjoint subtrees and their higher paths do not intersect each other. A pair of independent groups of exclusive paths has two fixed edges, i.e., the fixed edges of both groups (indicated by dotted arrows in Figure 5.6).

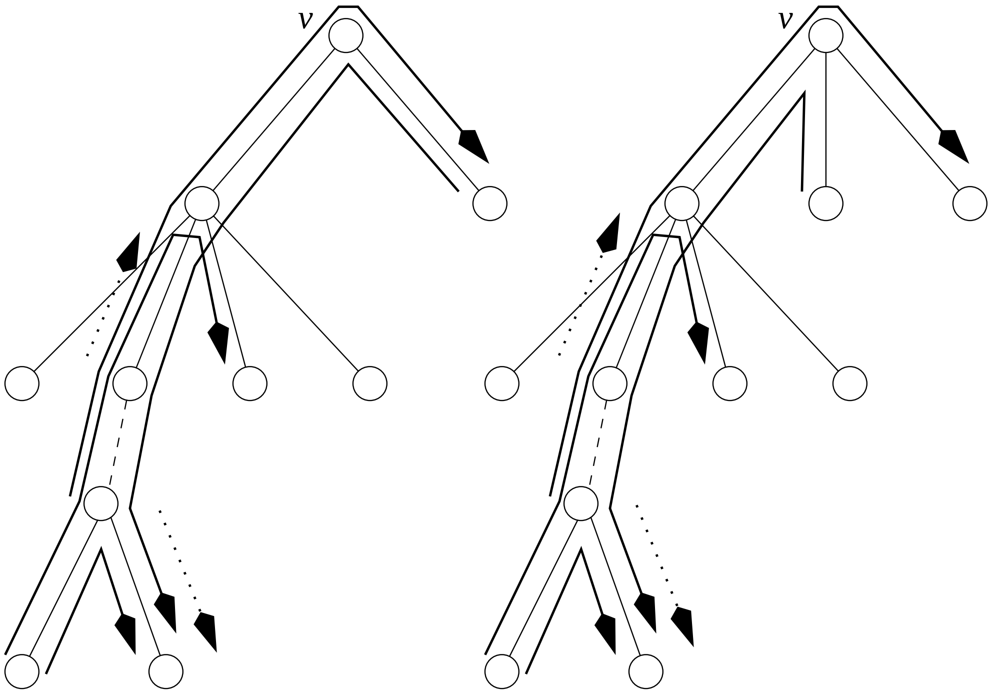


Figure 5.7: Further configurations of groups of 2-exclusive paths

The second possibility for a group of 2-exclusive paths is to consist of a group of exclusive paths contained in a subtree rooted at a child of v and two paths p_1 and p_2 with lca v that intersect the exclusive paths (but not their fixed edge) in a way such that accepting p_1 and p_2 would block both of the exclusive paths from being accepted (Figure 5.7). Two edges are marked fixed, namely the top edge of the higher exclusive path intersected by a path with lca v and the top edge of the lower exclusive path intersected by a path with lca v. The fixed edges and their directions are indicated by dotted arrows in Figure 5.7.

A group of 2-exclusive paths has the following property.

Property (2E): If at most one path touching v but not using a fixed edge is accepted at a later node, two paths from the group of 2-exclusive paths can be accepted. If two paths touching v but not using a fixed edge are accepted at a later node, at least one path from the group of 2-exclusive paths can be accepted.

While processing later nodes, the algorithm will try to avoid accepting two paths touching v such that only one path from the group of 2-exclusive paths can be accepted.

This concludes the description of the possibilities of paths being put into an intermediate state by the algorithm. When the algorithm has finished processing a node v, the subtree rooted at v will contain at most one of the following:

- one undetermined path, or
- one group of exclusive paths, or
- one group of 2-exclusive paths.

All other paths in the subtree are accepted, rejected, or member of a group of deferred paths.

When the algorithm has processed the root node at the end of the first pass, it can simply resolve the remaining paths that are in an unresolved state (if any) in a greedy manner: if there is an undetermined path, accept it; if there is a group of exclusive paths, pick one of them arbitrarily and accept it; if there is a group of 2-exclusive paths, pick two edge-disjoint paths arbitrarily and accept them. Now, all paths are accepted, rejected, or deferred, and the algorithm processes the nodes of the tree in a second pass in reverse order (order of non-decreasing levels, i.e., top-down) and determines at each node v that is the lca of one or more groups of deferred paths which of the deferred paths to accept. After this second pass, the algorithm outputs the set P' of all accepted paths.

5.3.1 Invariants

In the next subsection we will present the details of how the algorithm proceeds during the first pass. At the same time, we will show that the approximation ratio achieved by the algorithm is $5/3 + \varepsilon$, where ε is an arbitrarily small positive constant. For establishing this, we will prove by induction that the following invariants can be maintained. These invariants hold before the first node of T is processed, and they hold again each time an additional node of T has been processed. A node v is called a *root of a processed subtree* if the node v has already been processed, but its parent has not.

Invariant A: For every root v of a processed subtree, all paths in that subtree are accepted, rejected, or deferred except if one of the following cases occurs:

- The subtree contains one undetermined path. All other paths contained in the subtree are accepted, rejected, or deferred. No edge in the subtree is marked as fixed.

- The subtree contains one group of exclusive paths. All other paths contained in the subtree are accepted, rejected, or deferred. The only edge marked fixed in the subtree is the one corresponding to the group of exclusive paths.
- The subtree contains one group of 2-exclusive paths. All other paths contained in the subtree are accepted, rejected, or deferred. The only edges marked fixed in the subtree are the two corresponding to the group of 2-exclusive paths.

Invariant B: Let A be the set of all paths that have already been accepted by the algorithm. Let F be the set of all paths in P whose lca has not yet been processed and which are not blocked by any of the accepted paths, by reserved edges of deferred paths, or by fixed edges. Let d be the number of groups of deferred paths that are contained in processed subtrees. Let U be the set of all undetermined paths. Let X be the union of all groups of exclusive paths and groups of 2-exclusive paths. Then there is a subset $O \subseteq F \cup U \cup X$ of edge-disjoint paths satisfying the following conditions:

(a) $|P^*| \le (5/3+\varepsilon)(|A|+d)+|O|$

(b) for every group of exclusive paths, O contains one path from that group; for every group of 2-exclusive paths, O contains two paths from that group

Intuitively, the set O represents a subset of P containing paths that could still be accepted by the algorithm and that has the following property: if the algorithm accepts at least a $1/(5/3+\varepsilon)$-fraction of the paths in O (in addition to the paths it has already accepted), its output is a $(5/3+\varepsilon)$-approximation of the optimal solution.

Observe that the invariants are satisfied initially with $A=\emptyset$, $d=0$, $F=P$, $U=\emptyset$, $X=\emptyset$, and $O=P^*$ an arbitrary optimal solution. Whereas it will be easy to see from the description of the algorithm that Invariant A is indeed maintained throughout the first pass, special care must be taken to prove that Invariant B is maintained as well. For this purpose, we will show how the set O that establishes Invariant B before node v is processed can be manipulated so as to satisfy Invariant B also after node v is processed. In particular, a certain number of paths must be replaced in O or removed from O in order to satisfy $O \subseteq F \cup U \cup X$ and to keep O a set of edge-disjoint paths after v is processed, and it must be shown that the number of paths removed from O is at most $(5/3+\varepsilon)(a_v+d_v)$ if the algorithm accepts a_v additional paths and creates d_v new groups of deferred paths while processing v (thus satisfying condition (a) of Invariant B). Condition (b) must only be considered explicitly when a new group of exclusive paths or group of 2-exclusive paths is created by the algorithm.

If the invariants are satisfied after the root node is processed, we have $F = \emptyset$, $O \subseteq U \cup X$, and $|P^*| \le (5/3+\varepsilon)(|A|+d)+|O|$. At this time, there may still be one undetermined path, one group of exclusive paths, or one group of 2-exclusive paths. If there is an undetermined path, the algorithm accepts it. If there is a group of exclusive paths, the algorithm accepts one of them arbitrarily. If there is a group of 2-exclusive paths, the algorithm accepts two edge-disjoint paths of them arbitrarily. The algorithm accepts $|O|$ additional paths in this way, and the resulting set A satisfies $|P^*| \le (5/3+\varepsilon)(|A|+d)$.

In the second pass, the algorithm processes the nodes of the tree in reverse order, i.e., according to non-decreasing levels (top-down). At each node v that is the lca of at least one group of deferred paths, it accepts one path from each of the groups of deferred paths such that these paths are edge-disjoint to all previously accepted paths and to each other. This can always be done due to the definition of groups of deferred paths. Hence, the number of paths accepted by the algorithm increases by d in the second pass, and in the end we have $|P^*| \le (5/3+\varepsilon)|A|$. This establishes the main theorem of this chapter.

Theorem 5.3.1 *For every fixed $\varepsilon > 0$, there is a polynomial-time approximation algorithm for the maximum edge-disjoint paths problem in bidirected trees with approximation ratio $5/3+\varepsilon$.*

5.3.2 Details of the First Pass

Recall that the algorithm processes the nodes of the given tree T in order of non-increasing levels (bottom-up) in the first pass. Assume that the algorithm is just about to process node v. Recall that $P_v \subseteq P$ is the set of all paths with lca v that do not intersect any previously accepted path nor any fixed or reserved edge. Let U_v be the set of undetermined paths contained in subtrees rooted at children of v. Let X_v be the union of (paths in) groups of exclusive paths and groups of 2-exclusive paths contained in subtrees rooted at children of v. In the following, we explain how the algorithm processes node v and determines which of the paths in $P_v \cup U_v \cup X_v$ should be accepted, rejected, deferred, or left (or put) in an unresolved state.

Note that for a given set of paths with lca v the problem of determining a maximum cardinality subset of edge-disjoint paths is equivalent to finding a maximum matching in a bipartite graph (see Section 5.1.1 or Section 4.2.1 and Figure 4.1 on page 66 for an explanation of the reduction) and can thus be done in polynomial time. Whenever we use an expression like *compute a maximum number of edge-disjoint paths in* $S \subseteq P_v$ in the following, we imply that the computation should be carried out by employing this reduction to maximum matching.

Furthermore, we will use the following property of bipartite graphs: for $s = 1$ or $s = 2$, the fact that a maximum matching in a bipartite graph G has cardinality s implies that there are s vertices in G such that every edge is incident to at least one of these s vertices. (The property holds for arbitrary values of s and is known as the König theorem [Kön31]; see, e.g., the book by Berge [Ber76, pp. 132–133].)

Observe that each child of the current node v is the root of a processed subtree, which may, by Invariant A, contain at most one of the following:

- one undetermined path, or
- one group of exclusive paths, or
- one group of 2-exclusive paths.

Let k be the number of children of v that have an undetermined path in their subtree, let ℓ be the number of children of v that have a group of exclusive paths, and let m be the number of children of v that have a group of 2-exclusive paths. We use the expression *subtrees with exclusive paths* to refer to all subtrees rooted at children of v with either a group of exclusive paths or with a group of 2-exclusive paths.

Note that one main difficulty lies in determining which of the paths in $U_v \cup X_v$ should be accepted and which should be rejected. For example, if a path in U_v is accepted, it may block two edge-disjoint paths in P_v (or even paths with a least common ancestor of smaller level) from being accepted; if the path is rejected, the algorithm may lose one path as compared to an optimal solution without gaining any benefit. Similar difficulties exist for paths in X_v. If $k+\ell+m$ is bounded by a constant, all possible combinations of accepting and rejecting paths in $U_v \cup X_v$ can be tried out in polynomial time, but if $k + \ell + m$ is large, the algorithm must proceed in a different way in order to make sufficiently good decisions. Hence, we will distinguish the case that $k + \ell + m$ is small and the case that $k + \ell + m$ is large. The exact threshold for determining when $k + \ell + m$ is considered large, and consequently the running-time of the algorithm, depends on the constant ε.

As introduced in Section 5.3.1, let F denote the set of all paths whose lca is v or a node that is processed after v and which are not blocked by an accepted path or by a reserved or fixed edge before v is processed. Let U be the set of all undetermined paths before v is processed, let X be the union of all (paths in) groups of exclusive paths and groups of 2-exclusive paths before v is processed, and let d be the number of groups of deferred paths that are contained in processed subtrees before v is processed. Similarly, let F', U', X', and d' denote the respective sets or values after v is processed. Furthermore, denote by a_v the number of paths that are newly accepted while processing v, and denote by d_v the number of groups of deferred paths that are newly created while processing v.

Before v is processed, Invariant B implies that there is a set $O \subseteq F \cup U \cup X$ satisfying conditions (a) and (b). In order to satisfy Invariant B after v is processed, some paths in O may have to be replaced by other paths and some paths may have to be removed from O. The set derived from O in this way is referred to as O'.

When the algorithm processes v, it takes the paths in $P_v \cup U_v \cup X_v$ and possibly accepts some paths, rejects some paths, creates one or more new groups of deferred paths, creates a new undetermined path, creates a new group of exclusive paths, or creates a new group of 2-exclusive paths. Then all paths intersecting a newly accepted path or the reserved edge of a newly created group of deferred paths must be removed from O. Note that at most two such paths can have an lca of smaller level than v, because all such paths with lca of smaller level must use the edge $(v, p(v))$ or $(p(v), v)$. In addition, paths rejected by the algorithm must be removed or replaced in O in order to satisfy $O' \subseteq F' \cup U' \cup X'$. Furthermore, if a new group of exclusive paths or group of 2-exclusive paths is created, it must be ensured that O' contains one or two paths from that group, respectively, in order to maintain condition (b) of Invariant B.

In each single case of the following case analysis, it must be shown that conditions (a) and (b) of Invariant B are satisfied for O'. In particular, the number of paths deleted from O in order to obtain O' must be at most $(5/3 + \varepsilon)(a_v + d_v)$. As the value $|A| + d$ increases by $a_v + d_v$ while v is processed, this implies that condition (a) of Invariant B holds also after v is processed, i.e., $|P^*| \leq (5/3 + \varepsilon)(|A'| + d') + |O'|$. Recall that we say that a path p runs from u to w if it contains the directed path from u to w as a subpath.

Case 1: $k + \ell + m \leq \max\{3, 2/\varepsilon\}$. The algorithm can try out all combinations of accepting or rejecting unresolved paths in the subtrees rooted at children of v: for undetermined paths there are two possibilities (accepting or rejecting the path), for groups of exclusive paths there are two possibilities (accepting the lower path or accepting the higher path), and for groups of 2-exclusive paths there are either four possibilities (in the case of a pair of independent groups of exclusive paths as shown in Figure 5.6 on page 117:

accepting the lower or higher path in one group and the lower or higher path in the other group) or two relevant possibilities (in the cases shown in Figure 5.7 on page 118: accepting the lower or higher path of the group of exclusive paths contained in the group of 2-exclusive paths and the edge-disjoint path among the remaining two paths; note that accepting no path of the group of exclusive paths and only the remaining two paths blocks more paths from F than any of the other two possibilities, hence we do not need to consider this third possibility) of accepting two edge-disjoint paths of the group. Hence, the number of possible combinations is bounded from above by $2^{k+\ell}4^m = O(1)$. For each of these combinations, the algorithm can compute a maximum number of edge-disjoint paths in P_v not intersecting any of the paths from $U_v \cup X_v$ that are (tentatively) accepted for the current combination. Let s be the maximum, over all combinations, of the number of tentatively accepted paths from $U_v \cup X_v$ plus the number of maximum edge-disjoint paths in P_v. If $s = 0$, we have $k = \ell = m = 0$ and $P_v = \emptyset$, and the algorithm does nothing and proceeds with the next node. Otherwise, we distinguish the following cases.

Case 1.1: $s = 1$. As $s \geq k + \ell + 2m$, $s = 1$ implies $m = 0$ and $k + \ell \leq 1$.

Case 1.1.1: $k = \ell = m = 0$. If v has only one equivalence class of paths, pick one of them, say p, arbitrarily and make it an undetermined path. Reject all other paths in P_v. If O contains a path $p' \neq p$ from P_v, replace p' by p in O to obtain O', otherwise let $O' = O$. The invariants are satisfied.

If v has more than one equivalence class of paths, there must be an edge e incident to v that is shared by all paths in P_v (as a consequence of the König theorem). Make P_v a group of deferred paths with reserved edge e (cf. left-hand side of Figure 5.4 on page 114). Hence, we have $d_v = 1$. O may contain at most one path intersecting edge e, either a path from P_v or a path with lca of smaller level. It suffices to delete this path from O in order to obtain a valid set O'. Therefore, the invariants are satisfied after v is processed.

Case 1.1.2: $k = 1$, $\ell = m = 0$. There is one child c that has an undetermined path p with lca w in its subtree. If $P_v = \emptyset$, the algorithm does nothing and leaves p in its undetermined state. If $P_v \neq \emptyset$, all paths in P_v must intersect p in the same edge, say in the edge (u, w) with $w = p(u)$. (The case that they intersect p in an edge $(p(u), u)$ is symmetrical.) The algorithm picks an arbitrary path q from P_v and makes $\{p, q\}$ a group of exclusive paths (cf. Figure 5.5 on page 116). All other paths in P_v are rejected. If O does not contain any path from $P_v \cup U_v$, it may contain a path p' whose

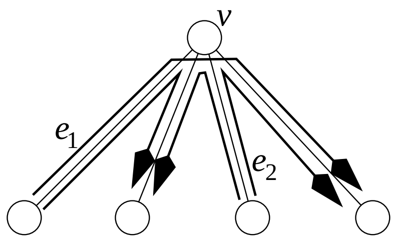


Figure 5.8: Case 1.2.1 (a): All sets of two edge-disjoint paths use the same four top edges

lca has smaller level than v and that uses the edge (u, w) (and possibly a second path with lca of smaller level), or it may contain two paths p_1 and p_2 with lca of smaller level not using (u, w) that intersect both p and q. In the former case, remove p' from O, and in the latter case, remove p_1 from O. In both cases, either p or q does not intersect a path from the resulting set O due to Property (E) and can therefore be inserted into O to obtain O'. If O contains a path p' from $P_v \cup U_v$ already, this path can be replaced by p or q if $p' \neq p, q$. In any case the invariants are satisfied after v is processed. In particular, $|O|$ does not decrease.

Case 1.1.3: $k = m = 0$, $\ell = 1$. There is one child of v that has a group of exclusive paths in its subtree. As any path from P_v could be combined with a path from the group of exclusive paths to obtain two edge-disjoint paths and because we have assumed $s = 1$, $\ell = 1$ implies $P_v = \emptyset$. Hence, the algorithm does nothing at node v and leaves the group of exclusive paths in its intermediate state.

Case 1.2: $s = 2$. Observe that $k + \ell + 2m \leq s = 2$.

Case 1.2.1: $k = \ell = m = 0$. As $s = 2$, there must be two edges incident to v such that all paths in P_v use at least one of these two edges (by the König theorem). Let e_1 and e_2 be two such edges.

Case (a): All possible sets of two edge-disjoint paths from P_v use the same four edges incident to v. See Figure 5.8 for an example. The algorithm picks two arbitrary edge-disjoint paths from P_v, accepts them, and rejects all other paths from P_v. Hence, we have $a_v = 2$. It suffices to remove at most three paths from O in order to obtain a valid set O': if O contains at most one path from P_v, removing this path and at most two paths with lca of smaller level is sufficient; if O contains two paths from P_v, removing these two paths is sufficient, because they use the same top edges as the paths accepted by

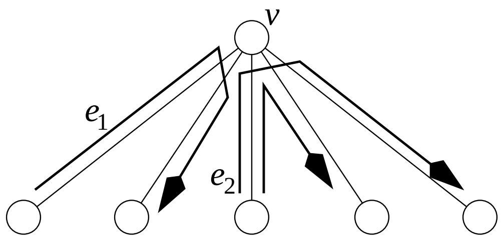


Figure 5.9: Case 1.2.1 (b): There is only one equivalence class of paths using edge e_1, but more than one class using edge e_2

the algorithm, and O cannot contain any further path intersecting the paths accepted by the algorithm. Hence, the invariants are satisfied.

In the following, let D be the set of all paths in P_v that intersect all other paths from P_v. In other words, a path $p \in P_v$ is in D if P_v does not contain a path q that is edge-disjoint to p.

Note that if Case (a) does not apply, it follows that either the paths in $P_v \setminus D$ using edge e_1 or those using edge e_2 must form more than one equivalence class of paths.

Case (b): There is only one equivalence class C of paths in $P_v \setminus D$ using edge e_1, but more than one equivalence class of paths in $P_v \setminus D$ using edge e_2 and not intersecting a path from C. See Figure 5.9. (The case with e_1 and e_2 exchanged is analogous. Furthermore, note that the case that there is only one equivalence class C of paths in $P_v \setminus D$ using edge e_1 and only one equivalence class of paths in $P_v \setminus D$ using edge e_2 and not intersecting a path from C satisfies the condition of Case (a).) The algorithm picks a path p from C arbitrarily, accepts p, and makes the paths using edge e_2 and not intersecting p a group of deferred paths with reserved edge e_2. All other paths in P_v are rejected. We have $a_v = 1$ and $d_v = 1$. It suffices to remove at most three paths from O in order to obtain a valid set O': if O contains at most one path from P_v, removing that path and at most two paths with lca of smaller level is sufficient; if O contains two paths from P_v, these paths must also use both top edges of p and the newly reserved edge, and thus removing these two paths from O is sufficient. Hence, the invariants are satisfied.

Case (c): There is more than one equivalence class of paths in $P_v \setminus D$ using edge e_1, there is more than one equivalence class of paths in $P_v \setminus D$ using edge e_2, and Cases (a) and (b) do not apply. The algorithm makes the paths in $P_v \setminus D$ using e_1 a group of deferred paths with reserved edge e_1, and the paths in $P_v \setminus D$ using e_2 a group of deferred paths with reserved edge e_2. All other paths in P_v are rejected. Note that no matter which paths with lca of smaller level are accepted by the algorithm later on, there are still

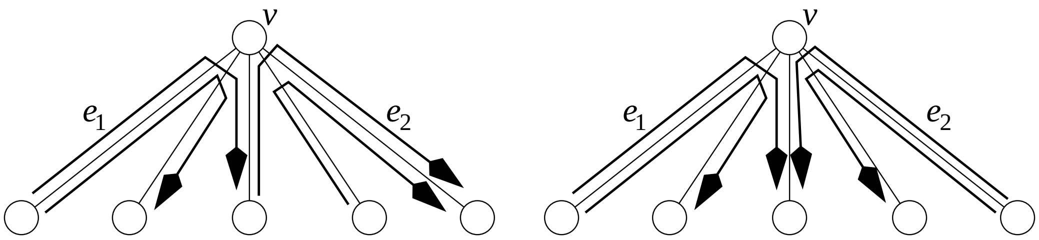


Figure 5.10: Case 1.2.1 (c): Configurations in which two groups of deferred paths can be created

two paths, one in each of the two groups of newly deferred paths, that are edge-disjoint from these paths with lca of smaller level and from each other.

To see this, assume first that e_1 is directed towards v and e_2 is directed towards a child of v. See the left-hand side of Figure 5.10. Obviously, no path from the group of deferred paths with reserved edge e_1 can intersect a path from the group of deferred paths with reserved edge e_2, and vice versa. Now, a path with lca of smaller level using edge $(v, p(v))$ and not intersecting a reserved edge can only intersect a path from the group of deferred paths with reserved edge e_2, and a path with lca of smaller level using edge $(p(v), v)$ and not intersecting a reserved edge can only intersect a path from the group of deferred paths with reserved edge e_1. In any case, there will still be at least one path in each of the two groups of deferred paths that can be accepted in the second pass. Next, assume that e_1 and e_2 are both directed towards v. See the right-hand side of Figure 5.10. (The case that they are both directed towards a child of v is symmetrical.) Among the paths with lca of smaller level not intersecting a reserved edge, only a path using the edge $(p(v), v)$ can intersect a path from either of the two groups of deferred paths. If, after accepting one such path with lca of smaller level later on, only one path from the two groups of deferred paths could be accepted in the second pass, Case (a) would actually apply for this configuration (see Figure 5.8).

Hence, we have $d_v = 2$ after creating the two new groups of deferred paths. It suffices to remove at most three paths from O in order to obtain a valid set O': if O contains at most one path from P_v, removing this path and at most two paths with lca of smaller level is sufficient; if O contains two paths from P_v, these paths use e_1 and e_2 as well, and removing these two paths from O is sufficient, because O cannot contain any further path intersecting a reserved edge of the newly deferred paths. The invariants are satisfied.

Case 1.2.2: $k = 1$, $\ell = m = 0$. There is one child of v that has an undetermined path in its subtree. We begin by making some simple observations.

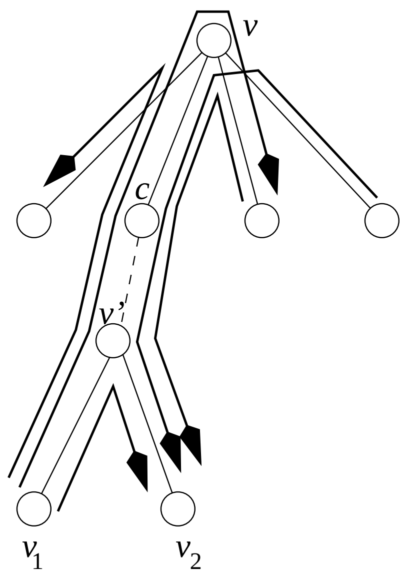


Figure 5.11: Case 1.2.2.2: P_v contains paths running from v_1 to several children of v, and from several children of v to v_2

First, the set of paths in P_v not intersecting the undetermined path must not contain two edge-disjoint paths. Hence, there must be an edge e incident to v that is shared by all paths in P_v not intersecting the undetermined path. Second, observe that $s = 2$ implies that the maximum number of edge-disjoint paths in P_v is also at most two. Hence, there must be two edges e_1 and e_2 incident to v such that every path in P_v uses at least one of these two edges.

Let the lca of the undetermined path be v', and let c be the child of v whose subtree contains the undetermined path (possibly $c = v'$). Let v_1 and v_2 be children of v' such that the undetermined path uses the edges (v_1, v') and (v', v_2). We consider a number of subcases regarding the structure of the paths in P_v.

Case 1.2.2.1: $P_v = \emptyset$. This case cannot occur, because we have $s = 2$.

Case 1.2.2.2: P_v contains paths running from v_1 to children c' of v for at least two different $c' \neq c$, and P_v contains paths running from children c' of v to v_2 for at least two different $c' \neq c$. See Figure 5.11. In this case, all paths from P_v not intersecting the undetermined path must use the edge (c, v) or the edge (v, c), because otherwise P_v would contain more than two edge-disjoint paths. The algorithm rejects the undetermined path, makes the paths in P_v using the edge (c, v) a group of deferred paths with reserved edge (c, v), and makes the paths in P_v using the edge (v, c) a second group of deferred paths with reserved edge (v, c).

It suffices to remove at most three paths from O in order to obtain a valid set O': if O contains at most one path from $P_v \cup U_v$, removing this path and at most two paths with lca of smaller level is sufficient; if O contains two paths from $P_v \cup U_v$, these two paths use at least one of the edges (c, v) and (v, c), and O can contain at most one path with lca of smaller level intersecting the other of these two edges. Hence, the invariants are satisfied.

Case 1.2.2.3: $P_v \neq \emptyset$, and v has only one equivalence class of paths in P_v. The algorithm accepts the undetermined path and an arbitrary edge-disjoint path from P_v. We have $a_v = 2$. It suffices to remove at most three paths from O in order to obtain a valid set O': if O contains at most one path from $P_v \cup U_v$, removing this path and at most two paths with lca of smaller level is sufficient; if O contains two paths from $P_v \cup U_v$, removing these two paths from O is sufficient, because O cannot contain any path with lca of smaller level intersecting a path accepted by the algorithm. Hence, the invariants are satisfied.

Case 1.2.2.4: $P_v \neq \emptyset$, no path in P_v intersects the undetermined path, and v has more than one equivalence class of paths in P_v. There must be an edge e incident to v that is shared by all paths in P_v. The algorithm accepts the undetermined path and makes P_v a group of deferred paths with reserved edge e. It suffices to remove at most three paths from O in order to obtain a valid set O': if O contains at most one path from $P_v \cup U_v$, removing this path and at most two paths with lca of smaller level is sufficient; if O contains two paths from $P_v \cup U_v$, it must contain the undetermined path and one path from P_v, and removing these two paths is sufficient. Hence, the invariants are satisfied.

Case 1.2.2.5: P_v contains paths running from v_1 to some $c' \neq c$, but all these paths run to the same c', and P_v contains paths running from some $c'' \neq c$ to v_2, but all these paths run from the same c'' to v_2. Note that $c' = c''$ is possible. The configuration of the paths in P_v intersecting the undetermined path is depicted in Figure 5.12, on the left-hand side for the case $c' = c''$ and on the right side for the case $c' \neq c''$. Let P'_v be the set of all paths in P_v that do not intersect the undetermined path. We distinguish several cases regarding the paths in P'_v.

Case (a): P'_v is empty. The algorithm accepts an arbitrary path running from v_1 to c' and an arbitrary path running from c'' to v_2. All other paths in $P_v \cup U_v$ are rejected. We have $a_v = 2$. It suffices to remove at most three paths from O in order to obtain a valid set O': if O contains at most one

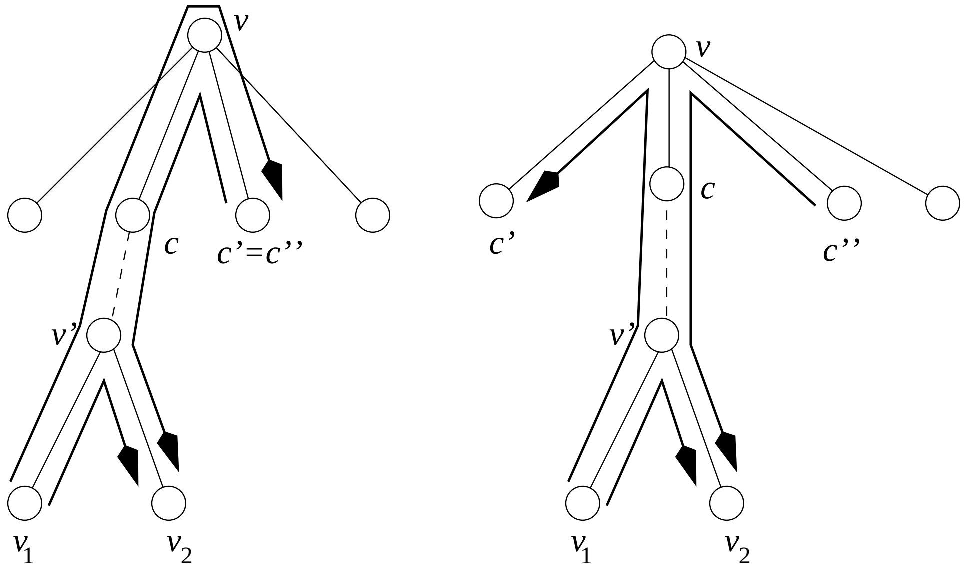


Figure 5.12: Case 1.2.2.5 (a): Configurations of paths in P_v intersecting the undetermined path

path from $P_v \cup U_v$, removing this path and at most two paths with lca of smaller level is sufficient; if O contains two paths from $P_v \cup U_v$, these two paths block the same edges incident to v (the edges (c, v), (v, c), (c'', v), and (v, c')) as the paths accepted by the algorithm, and removing these two paths is sufficient. Hence, the invariants are satisfied.

Case (b): All paths in P'_v run from c to the same child d of v, possibly $d = c'$ or $d = c''$. (The case that all paths in P'_v run from a child d to c is symmetrical.) See Figure 5.13. The algorithm accepts an arbitrary path from P'_v and the undetermined path. All other paths in P_v are rejected. Hence, $a_v = 2$. It suffices to remove at most three paths from O in order to obtain a valid set O': if O contains at most one path from $P_v \cup U_v$, removing that path and at most two paths with lca of smaller level is sufficient; if O contains two paths from $P_v \cup U_v$, these paths also block the edges (c, v) and (v', v_2), and O can contain at most one path with lca of smaller level intersecting a path accepted by the algorithm (i.e., a path with lca of smaller level using the edge (v, d)). Hence, the invariants are satisfied.

Case (c): All paths in P'_v run from c'' to c'. Note that this can occur only if $c' \neq c''$. See Figure 5.14. The algorithm accepts an arbitrary path from P'_v and the undetermined path. All other paths in P_v are rejected. Hence, $a_v = 2$. It suffices to remove at most three paths from O in order to obtain a valid set O': if O contains at most one path from $P_v \cup U_v$, removing this path and at most two paths with lca of smaller level is sufficient; if O

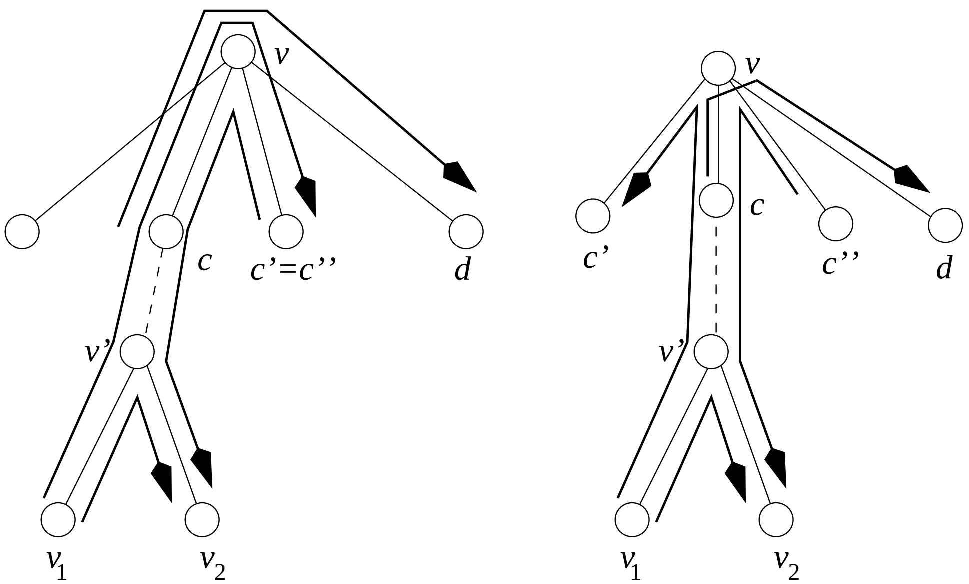


Figure 5.13: Case 1.2.2.5 (b): All paths in P'_v run from c to d

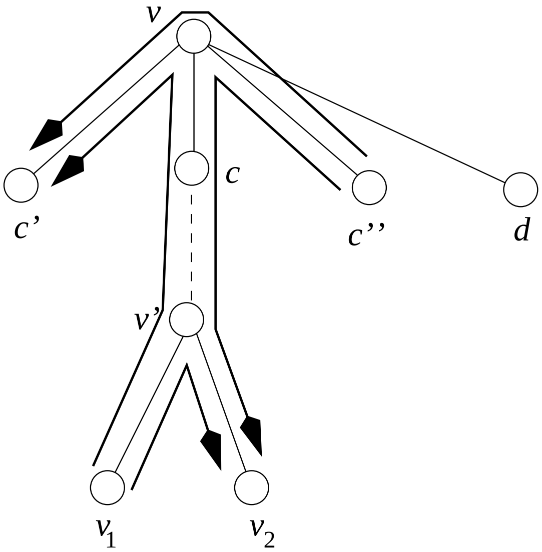


Figure 5.14: Case 1.2.2.5 (c): All paths in P'_v run from c'' to c'

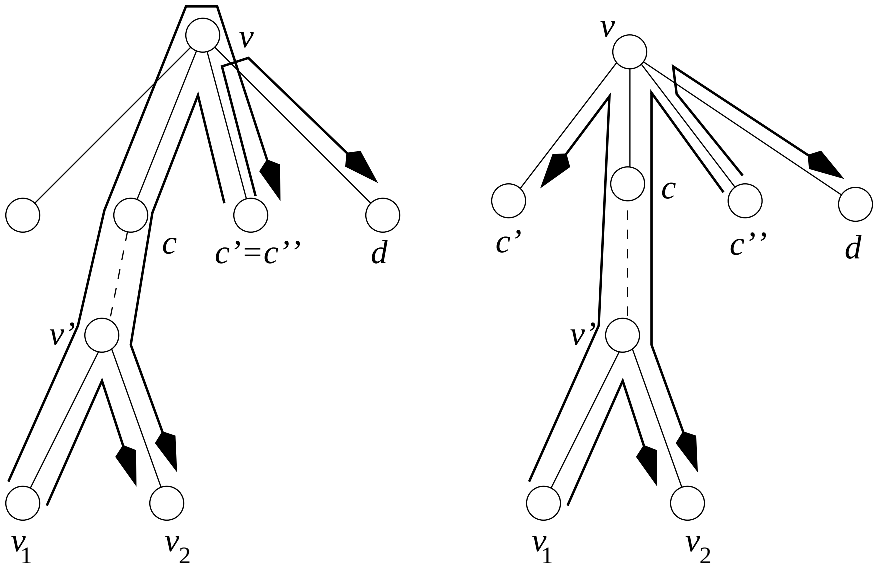


Figure 5.15: Case 1.2.2.5 (d): All paths in P_v' run from c'' to d

contains two paths from $P_v \cup U_v$, these paths also block the edges (v_1, v'), (v', v_2), (c'', v), and (v, c'), and O cannot contain any path with lca of smaller level intersecting a path accepted by the algorithm. Hence, the invariants are satisfied.

Case (d): All paths in P_v' run from c'' to the same child $d \neq c, c'$ of v. (The case that all paths in P_v' run from the same child $d \neq c, c''$ of v to c' is symmetrical.) See Figure 5.15. The algorithm accepts an arbitrary path from P_v' and the undetermined path. All other paths in P_v are rejected. Hence, $a_v = 2$. It suffices to remove at most three paths from O in order to obtain a valid set O': if O contains at most one path from $P_v \cup U_v$, removing this path and at most two paths with lca of smaller level is sufficient; if O contains two paths from $P_v \cup U_v$, these paths must also block the edges (v_1, v') and (c'', v), and O can contain at most one path with lca of smaller level intersecting a path accepted by the algorithm, because such a path with lca of smaller level must use the edge $(p(v), v)$. Hence, the invariants are satisfied.

Case (e): There is more than one equivalence class of paths in P_v', and the edge e incident to v that is used by all paths in P_v' is one of the edges (v, c), (c, v), (v, c'), or (c'', v). See Figure 5.16 (only the case $c' = c''$ is shown, because the case $c' \neq c''$ is not substantially different). The algorithm makes the paths in P_v' a group of deferred paths with reserved edge e, and it accepts the undetermined path. All other paths in P_v are rejected. We have $d_v = 1$ and $a_v = 1$. It suffices to remove at most three paths from O in order to obtain a valid set O': if O contains at most one path from $P_v \cup U_v$, removing

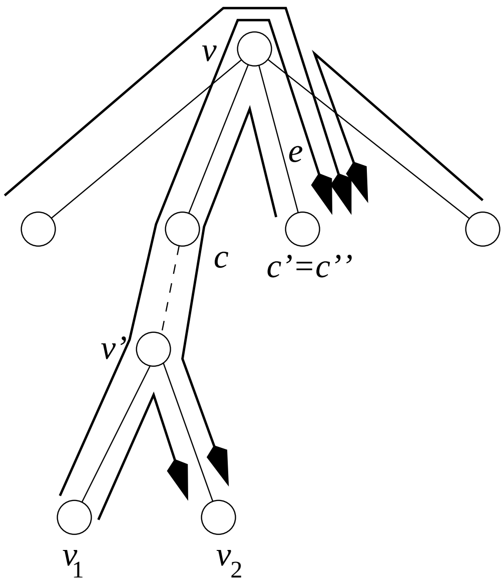


Figure 5.16: Case 1.2.2.5 (e): P'_v contains more than one equivalence class of paths

this path and at most two paths with lca of smaller level is sufficient; if O contains two paths from $P_v \cup U_v$, these paths must also block edge e and either (v_1, v') or (v', v_2), and O can contain at most one path with lca of smaller level intersecting a path accepted by the algorithm. Hence, the invariants are satisfied.

Case (f): There is more than one equivalence class of paths in P'_v, and the edge e incident to v that is used by all paths in P'_v is none of the edges (v, c), (c, v), (v, c'), or (c'', v). Assume that $e = (d, v)$ for some $d \neq c, c''$. (The case that $e = (v, d)$ for some $d \neq c, c'$ is analogous.) As $s = 2$, each path from P'_v must intersect one of the two edge-disjoint paths from P_v that intersect the undetermined path. Hence, there are exactly two equivalence classes of paths in P'_v: paths running from d to c, and paths running from d to c'. (In particular, the case $d = c'$ cannot occur.) See Figure 5.17. The algorithm accepts an arbitrary path running from d to c' and the undetermined path. Hence, we have $a_v = 2$. Observe that any combination of two edge-disjoint paths from $P_v \cup U_v$ blocks at least as many paths with lca of smaller level than three of the four top edges of the paths accepted by the algorithm. Hence, if O contains two paths from $P_v \cup U_v$, it can contain at most one path with lca of smaller level intersecting a path accepted by the algorithm, and it suffices to remove at most three paths from O to obtain O'. If O contains at most one path from $P_v \cup U_v$, removing this path and at most two paths with lca

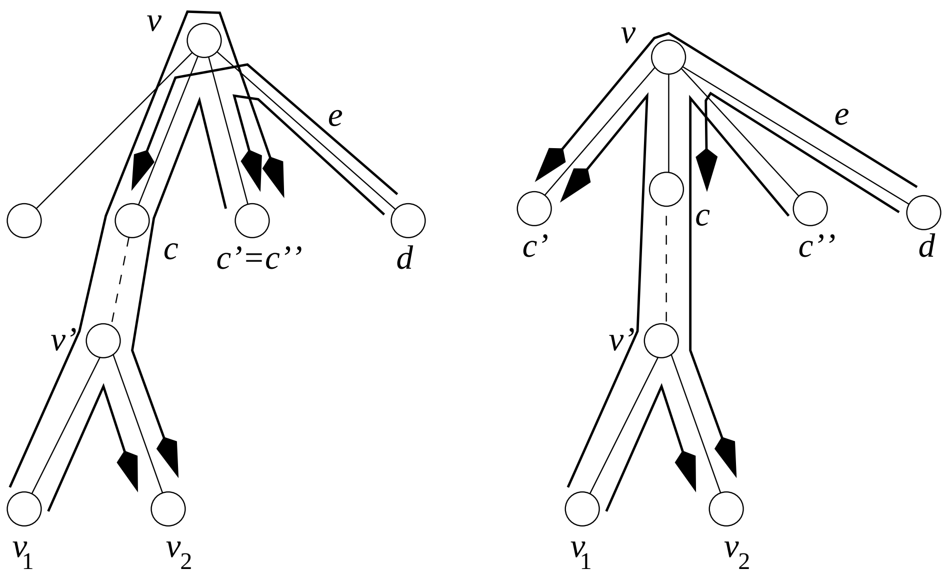


Figure 5.17: Case 1.2.2.5 (f): Special case of P'_v with two equivalence classes of paths

of smaller level is sufficient. The invariants are satisfied.

Case 1.2.2.6: P_v contains paths running from v_1 to some child $c' \neq c$ of v, but P_v does not contain paths running from some child $c'' \neq c$ of v to v_2. See Figure 5.18. (The case with paths running from c'' to v_2 but no paths running from v_1 to c' is analogous.) Let P'_v be the set of all paths in P_v that do not intersect the undetermined path. Note that $P'_v \neq \emptyset$, because $s = 2$.

Case 1.2.2.6.1: There is only one equivalence class of paths in P'_v. The algorithm accepts an arbitrary path from P'_v and the undetermined path. All other paths in P_v are rejected. We have $a_v = 2$. It suffices to remove at most three paths from O in order to obtain a valid set O': if O contains at most one path from $P_v \cup U_v$, removing this path and at most two paths with lca of smaller level is sufficient; if O contains two paths from $P_v \cup U_v$, these paths must consist of one path from P'_v and one path using the edge (v_1, v'), and O can contain at most one path with lca of smaller level intersecting the paths accepted by the algorithm. Hence, the invariants are satisfied.

Case 1.2.2.6.2: There is more than one equivalence class of paths in P'_v. Let e be the edge shared by all paths in P'_v. The algorithm makes P'_v a group of deferred paths with reserved edge e, and it accepts the undetermined path. All other paths in P_v are rejected. We have $a_v = 1$ and $d_v = 1$. It suffices to remove at most three paths from O in order to obtain a valid set O': if

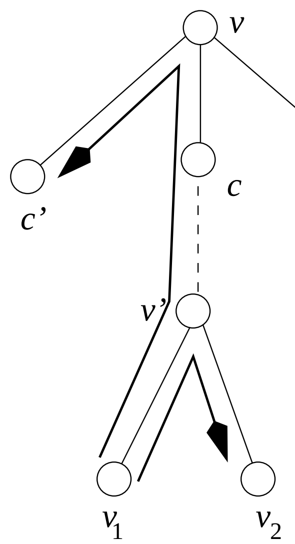


Figure 5.18: Case 1.2.2.6: P_v contains paths coming from v_1, but no paths running to v_2

O contains at most one path from $P_v \cup U_v$, removing this path and at most two paths with lca of smaller level is sufficient; if O contains two paths from $P_v \cup U_v$, these paths must consist of one path from P'_v, which uses edge e, and one path using the edge (v_1, v'), and O can contain at most one path with lca of smaller level intersecting the path accepted by the algorithm. Hence, the invariants are satisfied.

Case 1.2.2.7: P_v contains paths running from v_1 to some $c' \neq c$, but all these paths run to the same c', and P_v contains paths running from some $c'' \neq c$ to v_2 for at least two different c''. (The case with multiple equivalence classes of paths coming from v_1 and a single equivalence class of paths running to v_2 is analogous.) See Figure 5.19. Let P'_v be the set of all paths in P_v that do not intersect the undetermined path. Every path in P'_v must intersect at least one of the edges (v, c), (c, v), or (v, c'), because otherwise P_v would contain three edge-disjoint paths.

Case 1.2.2.7.1: $P'_v = \emptyset$. The algorithm accepts an arbitrary path from P_v running from v_1 to c', and it makes the paths in P_v running from some c'' to v_2 a group of deferred paths with reserved edge (v, c). We have $a_v = 1$ and $d_v = 1$, and it suffices to remove at most three paths from O to obtain a valid set O': if O contains at most one path from $P_v \cup U_v$, removing this path and at most two paths with lca of smaller level is sufficient; if O contains two paths from $P_v \cup U_v$, these paths must also use the edges (v, c), (c, v), and (v, c'),

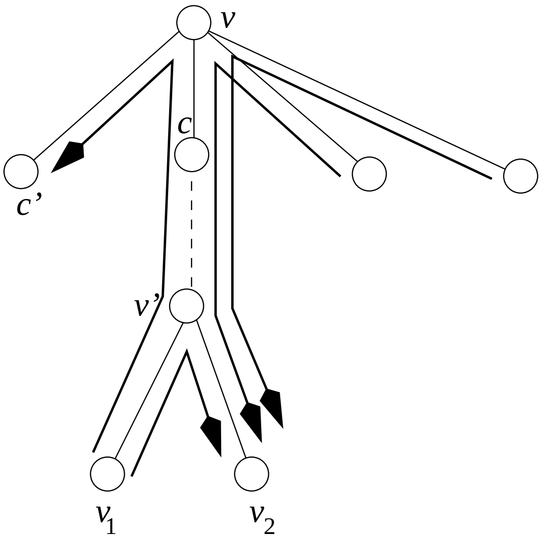


Figure 5.19: Case 1.2.2.7: P_v contains one equivalence class of paths coming from v_1, but several equivalence classes of paths running to v_2

and O cannot contain any path with lca of smaller level intersecting the path accepted by the algorithm or the newly reserved edge. The invariants are satisfied.

Case 1.2.2.7.2: P'_v has one equivalence class of paths. The algorithm accepts an arbitrary path from P'_v and the undetermined path. We have $a_v = 2$. It suffices to remove at most three paths from O to obtain a valid set O': if O contains at most one path from $P_v \cup U_v$, removing this path and at most two paths with lca of smaller level is sufficient; if O contains two paths from $P_v \cup U_v$, it can contain at most one path with lca of smaller level intersecting a path accepted by the algorithm. (To see the latter, note that either O contains the undetermined path, in which case it must also contain a path from P'_v, or it does not contain the undetermined path, in which case it must either contain a path running from v_1 to c' and a path running from some c'' to v_2, thus blocking three of the four top edges of the paths accepted by the algorithm, or it must contain a path from P'_v and a path from $P_v \setminus P'_v$, again blocking three of the four top edges of the paths accepted by the algorithm.) The invariants are satisfied.

Case 1.2.2.7.3: P'_v has more than one equivalence class of paths, and the edge e shared by all these paths is one of the edges (c, v), (v, c), or (v, c'). The algorithm accepts the undetermined path and makes P'_v a group of deferred paths with reserved edge e. We have $a_v = 1$ and $d_v = 1$. It suffices to

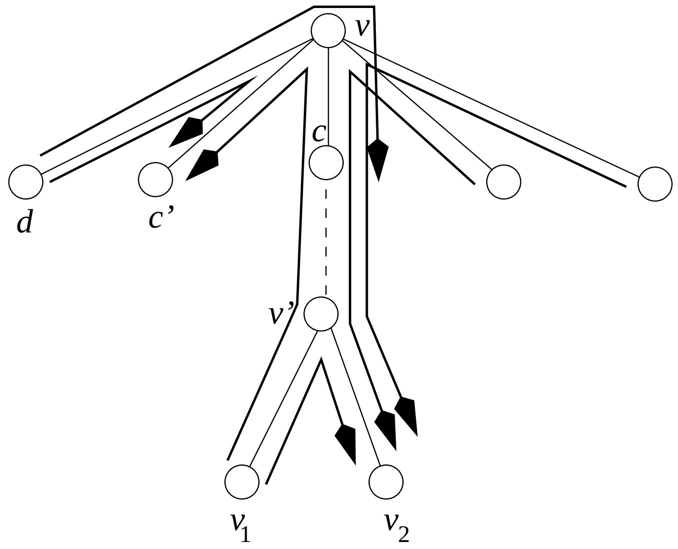


Figure 5.20: Case 1.2.2.7.4: P'_v contains several equivalence classes of paths sharing the edge (d, v)

remove at most three paths from O to obtain a valid set O': if O contains at most one path from $P_v \cup U_v$, removing this path and at most two paths with lca of smaller level is sufficient; if O contains two paths from $P_v \cup U_v$, it can contain at most one path with lca of smaller level intersecting the path accepted by the algorithm or the newly reserved edge. (To see the latter, note that either O contains the undetermined path, in which case it must also contain a path from P'_v using the newly reserved edge, or it does not contain the undetermined path, in which case it must either contain a path running from v_1 to c' and a path running from some c'' to v_2, thus blocking both top edges of the path accepted by the algorithm and the reserved edge of the new group of deferred paths, or it must contain a path from P'_v and a path from $P_v \setminus P'_v$, thus blocking the reserved edge and one of the two top edges of the path accepted by the algorithm.) The invariants are satisfied.

Case 1.2.2.7.4: P'_v has more than one equivalence class of paths, and the edge e shared by all these paths is an edge (d, v) for some $d \neq c, c'$. (Note that it is not possible that $e = (c', v)$ or $e = (v, d)$ for some $d \neq c', c$, because we would get three edge-disjoint paths in P_v in this case.) As every path in P'_v must intersect either the path running from v_1 to c' or all paths running from some c'' to v_2, the situation must be as shown in Figure 5.20: P'_v contains exactly two equivalence classes of paths, one class of paths running to c' and one class of paths running to c. The algorithm accepts an arbitrary path running from d to c' and the undetermined path. Hence, we have $a_v = 2$.

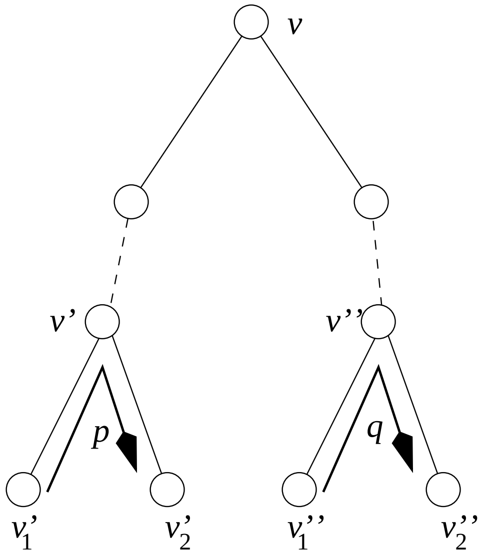


Figure 5.21: Case 1.2.3: v has two children with undetermined paths in their subtrees

Observe that any combination of two edge-disjoint paths from $P_v \cup U_v$ blocks at least three of the four top edges of the paths accepted by the algorithm. Hence, if O contains two paths from $P_v \cup U_v$, it can contain at most one path with lca of smaller level intersecting a path accepted by the algorithm, and it suffices to remove at most three paths from O to obtain O'. If O contains at most one path from $P_v \cup U_v$, removing this path and at most two paths with lca of smaller level is sufficient. The invariants are satisfied.

Case 1.2.3: $k = 2$, $\ell = m = 0$. Two children of v have undetermined paths in their subtrees. Denote the undetermined paths by p and q. See Figure 5.21. As $s = 2$, every path in P_v must intersect at least one undetermined path. In addition, if there are two paths in P_v that intersect one undetermined path in different top edges, at least one of them must also intersect the other undetermined path.

Case 1.2.3.1: $P_v = \emptyset$. The algorithm accepts both undetermined paths. Hence, we have $a_v = 2$, and we claim that it suffices to delete at most three paths from O in order to obtain a valid set O'. If O contains both undetermined paths, it suffices to remove these two paths from O. If O contains only one of the two undetermined paths, it may contain at most two paths with lca of smaller level intersecting the other undetermined path, and it suffices to remove at most three paths from O to obtain O'. If O does not contain any of the two undetermined paths, O can contain at most

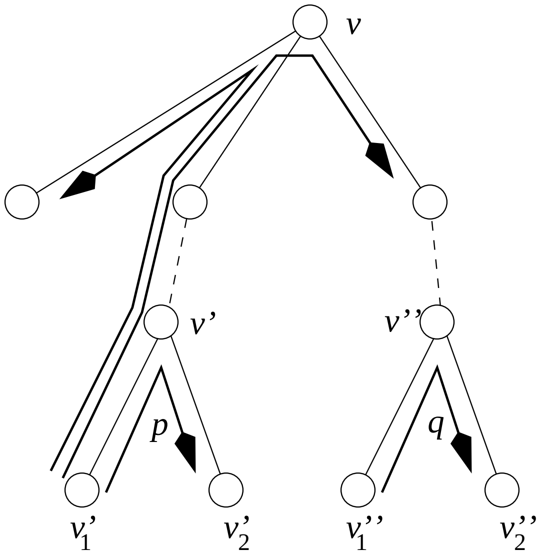


Figure 5.22: Case 1.2.3.2: All paths in P_v intersect the same undetermined path

two paths with lca of smaller level intersecting the paths accepted by the algorithm, and only these two paths must be deleted from O in order to obtain a valid set O'.

Case 1.2.3.2: $P_v \neq \emptyset$, every path in P_v intersects the same undetermined path, say p, and no path in P_v intersects the other undetermined path. Note that all paths in P_v must intersect p in the same top edge, because otherwise we would have $s = 3$. See Figure 5.22. The algorithm accepts both undetermined paths. Hence, we have $a_v = 2$, and we claim that it suffices to delete at most three paths from O in order to obtain a valid set O': If O contains both undetermined paths, it cannot contain any path with lca of smaller level intersecting a path accepted by the algorithm, and it suffices to remove the undetermined paths from O; if O contains a path from P_v intersecting p and the other undetermined path, it can contain at most one path with lca of smaller level intersecting p, and it suffices to remove these at most three paths; if O contains at most one path from $P_v \cup U_v$, it can contain at most two paths with lca of smaller level intersecting a path accepted by the algorithm, and again it suffices to remove these at most three paths. The invariants are satisfied.

Case 1.2.3.3: $P_v \neq \emptyset$, at least one path in P_v intersects the undetermined path p, and at least one path intersects the other undetermined path q.

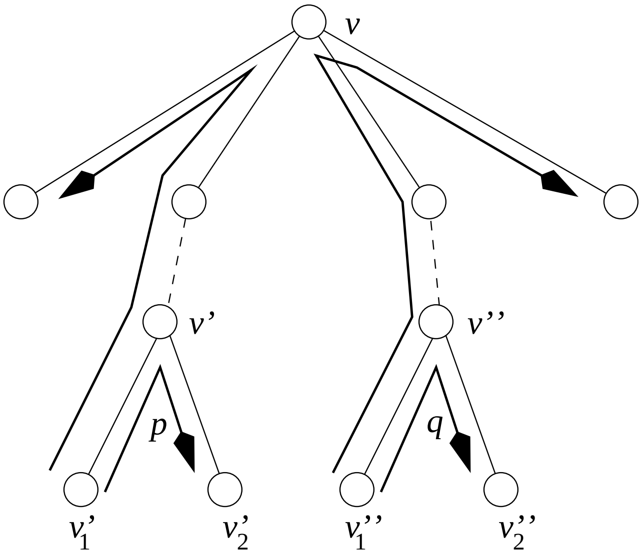


Figure 5.23: Case 1.2.3.3.1 (a): Both intersection edges are directed towards the root

Case 1.2.3.3.1: All paths from P_v intersecting p intersect p in the same edge e_1, and all paths from P_v intersecting q intersect q in the same edge e_2.

Case (a): e_1 and e_2 are both directed towards the root of the tree. (The case that e_1 and e_2 are both directed to the leaves is symmetrical.) See Figure 5.23. The algorithm accepts both undetermined paths. If O contains two paths from $P_v \cup U_v$, these paths block the edges e_1 and e_2, and O can contain at most one path with lca of smaller level intersecting a path accepted by the algorithm, because that path must use the edge $(p(v), v)$. Hence, it suffices to remove at most three paths from O in order to obtain O'. If O contains at most one path from $P_v \cup U_v$, it can contain at most two paths with lca of smaller level intersecting a path accepted by the algorithm, and again it suffices to remove at most three paths from O in order to obtain O'.

Case (b): One of e_1 and e_2, say e_1, is directed towards the root, and e_2 is directed towards the leaves. See Figure 5.24. If P_v contains two edge-disjoint paths p_1 and p_2 (this is the case in Figure 5.24), the algorithm makes p, q, p_1 and p_2 a group of 2-exclusive paths consisting of a pair of independent groups of exclusive paths, and rejects all other paths from P_v. The edges e_1 and e_2 are marked fixed. If O contains two paths from the new group of 2-exclusive paths already, let $O' = O$. Otherwise, it is possible to replace paths in O by paths from the new group of 2-exclusive paths to obtain O'. In any case, $|O|$ does not decrease, and the invariants are satisfied.

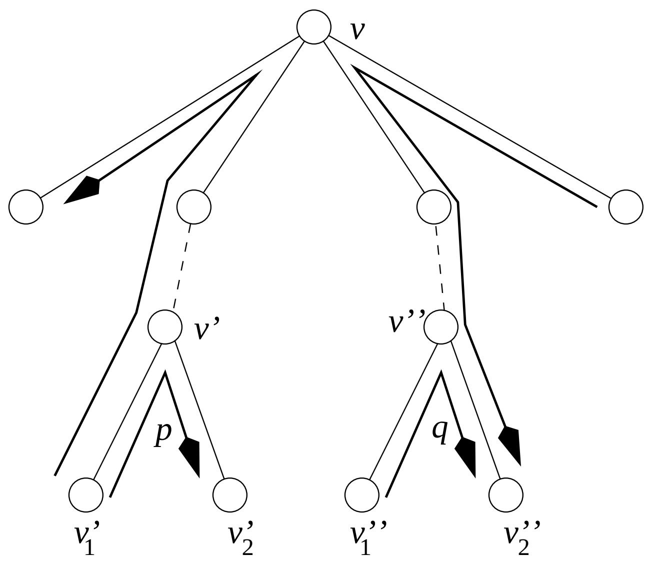


Figure 5.24: Case 1.2.3.3.1 (b): The intersection edges have different directions

If P_v does not contain two edge-disjoint paths, the algorithm accepts both undetermined paths. If O contains both undetermined paths, it suffices to remove these two paths to obtain a valid set O'. If O contains one undetermined path and one path from P_v, these paths block three out of four top edges blocked by the paths accepted by the algorithm, and O can contain at most one path with lca of smaller level intersecting a path accepted by the algorithm. It suffices to remove at most three paths from O. If O contains at most one path from $P_v \cup U_v$, again it suffices to remove at most three paths from O to obtain O'.

Case 1.2.3.3.2: There are paths $p_1, p_2 \in P_v$ such that p_1 intersects p in an edge directed towards the root, p_2 intersects p in an edge directed towards the leaves, and either p_1 or p_2 does not intersect q. (The case with p replaced by q is symmetrical.) Recall that at least one of p_1 and p_2 must also intersect q. Without loss of generality, assume that p_1 intersects q and p_2 does not. See Figure 5.25.

Let v' be the lca of p, and let v'' be the lca of q. Let the top edges of p be (v'_1, v') and (v', v'_2). Let the top edges of q be (v''_1, v'') and (v'', v''_2). Let c' be the child of v that is an ancestor of v', possibly $c' = v'$, and let c'' be the child of v that is an ancestor of v'', possibly $v'' = c''$. Note that all paths in P_v that run from v'_1 to v must intersect q, because otherwise such a path could be combined with p_2 and q to obtain three edge-disjoint paths, contradicting $s = 2$.

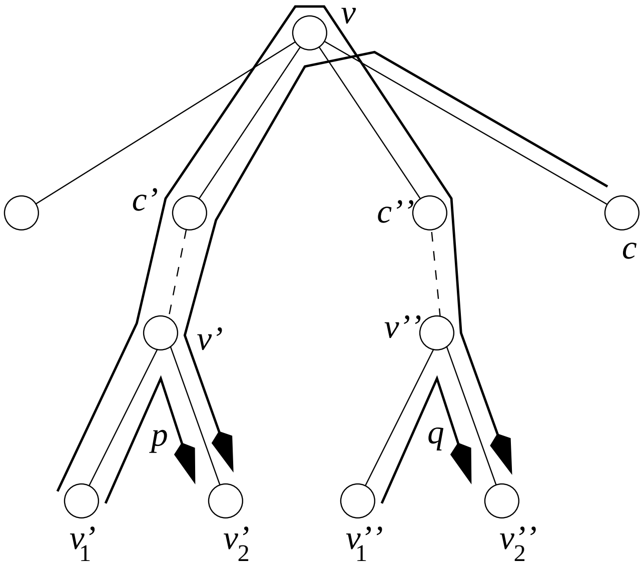


Figure 5.25: Case 1.2.3.3.2 (a): Paths from P_v intersect p in different directions

The algorithm accepts both undetermined paths p and q. In order to maintain the invariants, we must show that O contains at most three paths intersecting p or q. If O contains only one path from $P_v \cup U_v$, this follows because O can contain at most two paths with lca of smaller level intersecting p or q. If O contains two paths from $P_v \cup U_v$, we claim that O can contain at most one path with lca of smaller level intersecting p or q. We distinguish two cases.

Case (a): There is a path in P_v running from a child $c \neq c', c''$ of v to v and intersecting p in the edge (v', v_2'). Note that this is the case in Figure 5.25. Observe that any path in P_v that runs from v_1'' to v must use the edge (v, c'), because otherwise we would get three edge-disjoint paths in $P_v \cup U_v$.

If P_v does not contain any path running from v_1'' to v, any selection of two edge-disjoint paths from $P_v \cup U_v$ blocks the edges (v', v_2') and (v'', v_2''). Hence, if O contains two paths from $P_v \cup U_v$, it can contain at most one path with lca of smaller level intersecting p or q, because such a path must use the edge $(v, p(v))$.

If P_v contains a path running from v_1'' to c' not intersecting p, note that P_v cannot contain any path running from v to v_2'' not intersecting p (otherwise, we would get three edge-disjoint paths in $P_v \cup U_v$). In this case, any combination of two edge-disjoint paths from $P_v \cup U_v$ blocks at least three out of the four top edges of p and q, and O can contain at most one path with

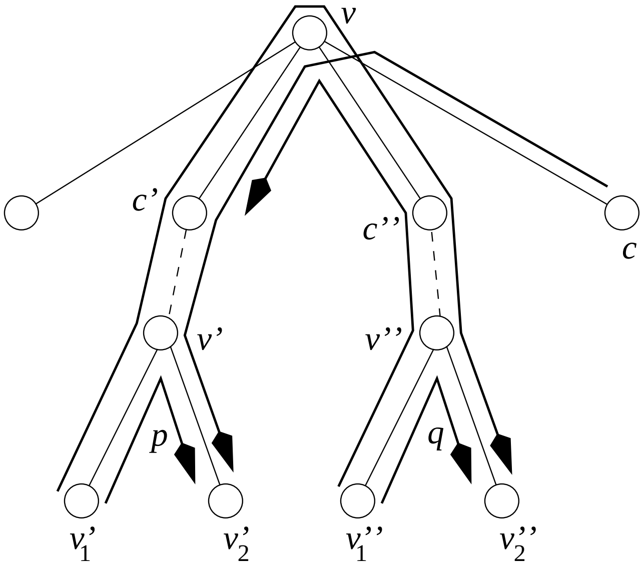


Figure 5.26: Case 1.2.3.3.2 (a) cont.: P_v contains a path from v_1'' to c' not intersecting p

lca of smaller level intersecting p or q. See Figure 5.26.

If P_v contains at least one path running from v_1'' to c', and if all such paths intersect p, then P_v may contain an arbitrary number of paths running from arbitrary children d of v to v_2''. See Figure 5.27. All paths in $P_v \cup U_v$ use edge (v', v_2') or edge (v'', v_2''). Hence, if O contains two paths from $P_v \cup U_v$, these paths block (v', v_2') and (v'', v_2''), and O can contain at most one path with lca of smaller level intersecting p or q, because such a path must use the edge $(v, p(v))$.

Case (b): All paths in P_v intersecting p in the edge (v', v_2') run from c'' to v_2'. See Figure 5.28.

If there is a path in P_v intersecting q in the edge (v'', v_2'') but not intersecting p (see Figure 5.29), observe that any path running from v_1'' to v must intersect p, because otherwise we would get three edge-disjoint paths in $P_v \cup U_v$. Hence, all paths in $P_v \cup U_v$ use edge (v', v_2') or edge (v'', v_2''). If O contains two paths from $P_v \cup U_v$, these paths block (v', v_2') and (v'', v_2''), and O can contain at most one path with lca of smaller level intersecting p or q, because such a path must use the edge $(v, p(v))$.

If all paths in P_v intersecting q in the edge (v'', v_2'') intersect p, then P_v may contain an arbitrary number of paths running from v_1'' to arbitrary children d of v. See Figure 5.30. Observe that any selection of two edge-disjoint paths from $P_v \cup U_v$ blocks three of the four top edges of p and q. Hence, if O

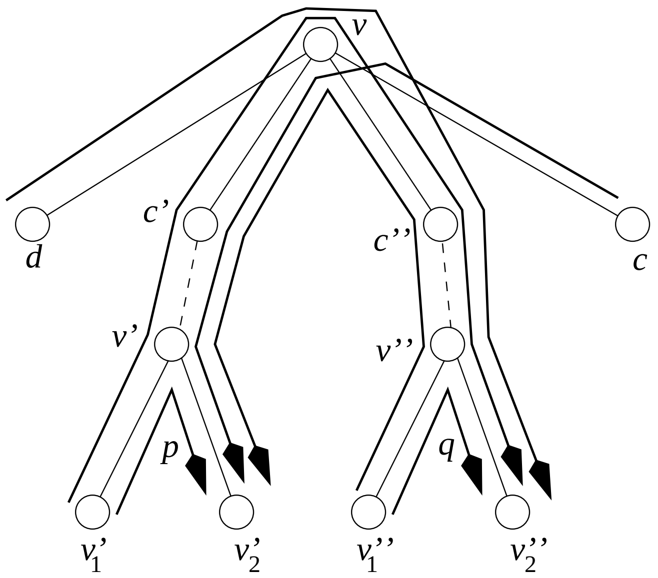


Figure 5.27: Case 1.2.3.3.2 (a) cont.: All paths in P_v running from v_1'' to v intersect p

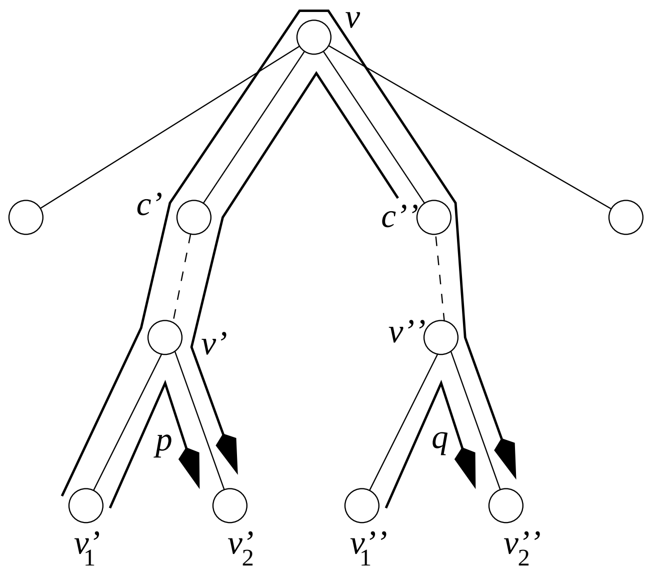


Figure 5.28: Case 1.2.3.3.2 (b): All paths in P_v intersecting p in (v', v_2') run from c'' to v_2'

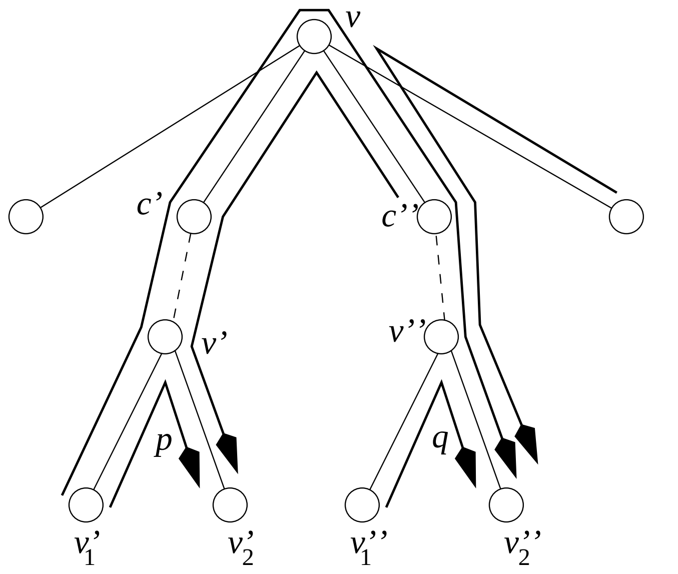


Figure 5.29: Case 1.2.3.3.2 (b) cont.: P_v contains a path running from v to v_2'' not intersecting p

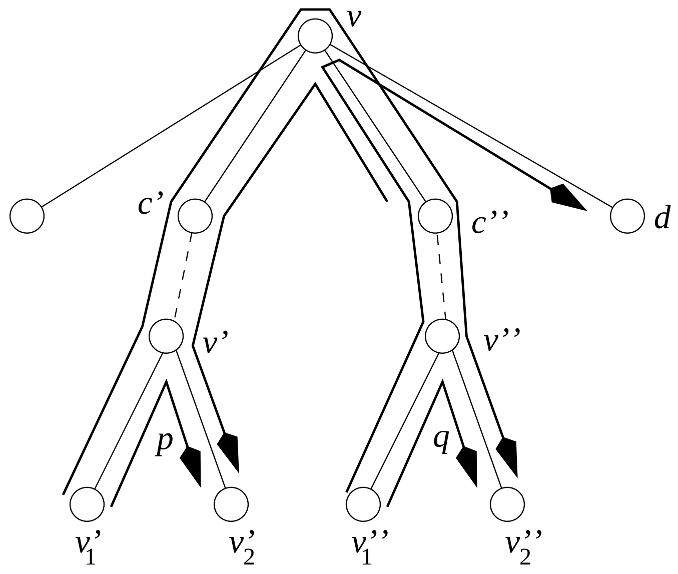


Figure 5.30: Case 1.2.3.3.2 (b) cont.: All paths in P_v running from v to v_2'' intersect p

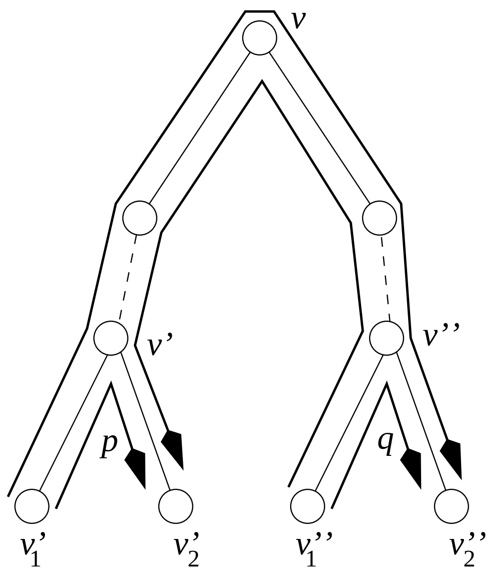


Figure 5.31: Case 1.2.3.3.3: All paths from P_v intersect p and q

contains two paths from $P_v \cup U_v$, it can contain at most one path with lca of smaller level intersecting p or q.

Case 1.2.3.3.3: All paths in P_v intersect both p and q, and there are two edge-disjoint paths in P_v. See Figure 5.31. The algorithm accepts p and q. We have $a_v = 2$. If O contains two edge-disjoint paths from $P_v \cup U_v$, these two paths block at least as many paths with lca of smaller level as p and q, and O cannot contain any path with lca of smaller level intersecting p or q; it suffices to remove two paths from O. If O contains at most one path from $P_v \cup U_v$, it can contain at most two paths with lca of smaller level intersecting p or q, and it suffices to remove at most three paths from O. The invariants are satisfied.

Case 1.2.4: $\ell = 1$, $k = m = 0$. There is one child c of v that has a group of exclusive paths in its subtree. We distinguish further cases regarding the maximum number of edge-disjoint paths in P_v.

Case 1.2.4.1: $P_v = \emptyset$. In this case, $P_v \cup X_v$ cannot contain two edge-disjoint paths. Hence, this case cannot occur for $s = 2$.

Case 1.2.4.2: There are two edge-disjoint paths p_1 and p_2 in P_v. As $s = 2$, p_1 and p_2 must intersect the exclusive paths in a way that blocks all of them from being accepted. See Figure 5.32. Denote the higher and the lower path in the group of exclusive paths by p and q, respectively. Assume without loss

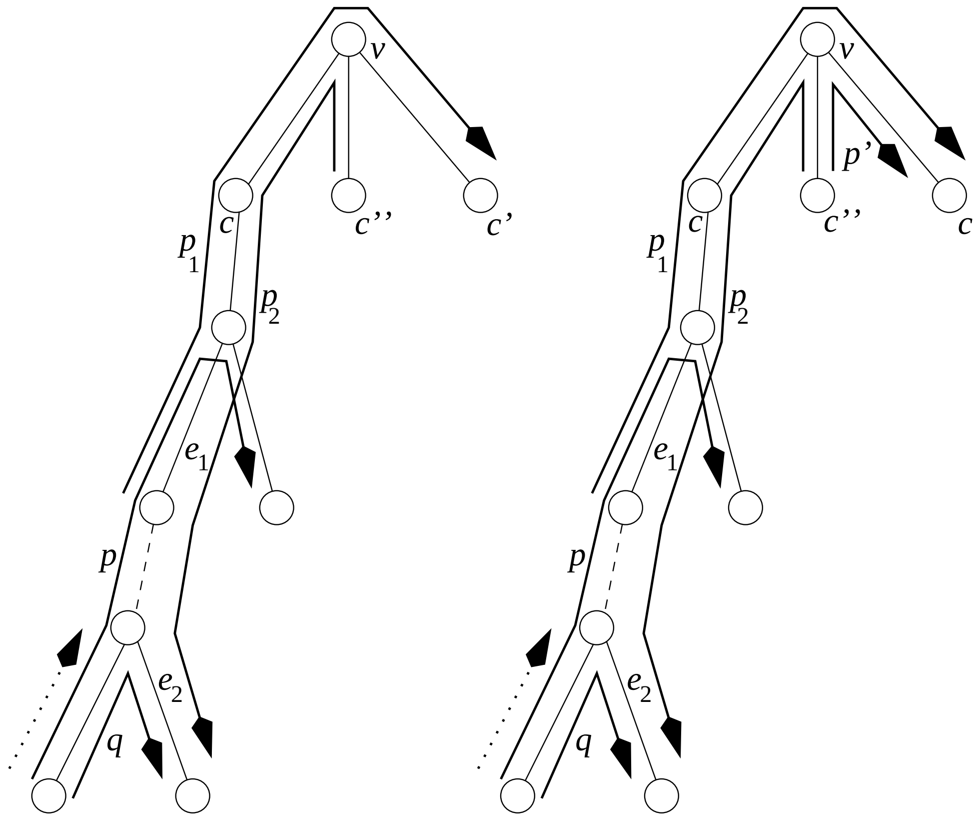


Figure 5.32: Case 1.2.4.2: P_v contains two edge-disjoint paths that block the exclusive paths

of generality that the fixed edge of the group of exclusive paths is directed towards the root of the tree (as shown in Figure 5.32). Let p_1 intersect p, and let p_2 intersect q. Let $c' \neq c$ be the child of v such that p_1 runs from c to c', and let $c'' \neq c$ be the child of v such that p_2 runs from c'' to c. Note that $c' = c''$ is possible. Let the top edge of p intersected by p_1 be e_1, and let the top edge of q intersected by p_2 be e_2. Observe that every path $p' \in P_v$ must either intersect edge e_1, or intersect edge e_2, or intersect both p_1 and p_2. (The latter case is possible only if $c' \neq c''$ and if all paths in P_v that intersect e_1 run from c to c' and all paths in P_v that intersect e_2 run from c'' to c; in that case, p' must run from c'' to c', as shown on the right-hand side of Figure 5.32.) Otherwise, $P_v \cup X_v$ would contain more than two edge-disjoint paths.

Case (a): All paths in P_v that intersect e_1 run from c to c', and all paths in P_v that intersect e_2 run from c'' to c.

First, assume that all paths in P_v intersect either e_1 or e_2. Note that there are exactly two equivalence classes of paths in P_v in this case. See Figure 5.32 (left-hand side). The algorithm uses the group of exclusive paths plus one representative from each of the two equivalence classes of paths in P_v to form a group of 2-exclusive paths (cf. Figure 5.7 on page 118). All other paths in P_v are rejected. The fixed edge of the group of exclusive paths is no longer marked fixed, instead the edges e_1 and e_2 are marked fixed (compare the dotted arrows in Figure 5.7). We must manipulate the set O of paths in order to satisfy the invariants. Note that O cannot contain two paths from P_v as O contains one of the exclusive paths due to condition (b) of Invariant B. If O contains one path from P_v, replace that path by the path in the same equivalence class picked by the algorithm as the representative of that class. If O does not contain any path from P_v, O can also be modified such that it contains two paths from the new group of 2-exclusive paths. To see this, note that O can contain at most one path with lca of smaller level intersecting a newly fixed edge; if it contained two such paths, these paths would block both exclusive paths in X_v, a contradiction to condition (b) of Invariant B. Therefore, it suffices to remove at most one path with lca of smaller level from O to achieve a situation in which none of the two newly fixed edges is intersected by a path in O with lca of smaller level and in which at most one path in O with lca of smaller level touches the subtree rooted at v at all. After removing from O also the exclusive path in X_v that was contained in O, we can add two paths from the new group of 2-exclusive paths to O due to Property (2E). $|O|$ does not decrease, and the invariants are satisfied.

Now assume that there is a path p' in P_v that intersects neither e_1 nor e_2. As noted above, we must have $c' \neq c''$ in this case, and p' must run from c'' to c'. See Figure 5.32 (right-hand side). The algorithm accepts the lower path from the group of exclusive paths and the path p', and it rejects all other paths in $P_v \cup X_v$. No edge is marked fixed anymore. We have $a_v = 2$. If O contains only one path from $P_v \cup X_v$, removing this path and at most two additional paths with lca of smaller level is sufficient. Furthermore, note that any combination of two edge-disjoint paths from $P_v \cup X_v$ blocks at least three of the four top edges of the paths accepted by the algorithm. Hence, if O contains two paths from $P_v \cup X_v$, it can contain at most one path with lca of smaller level intersecting the paths accepted by the algorithm but not intersecting the two paths from $P_v \cup X_v$ contained in O. In any case, it suffices to delete at most three paths from O to obtain a valid set O'.

Case (b): There are at least two equivalence classes of paths in P_v intersecting the higher path of the group of exclusive paths. The algorithm accepts the lower path of the group of exclusive paths and makes the paths in P_v intersecting the higher path a group of deferred paths. All other paths in $P_v \cup X_v$ are rejected, and no edge is marked fixed anymore. The reserved edge of the group of deferred paths is the top edge shared by all these paths. If O contains two paths from $P_v \cup X_v$, note that one of the two paths must be from X_v (due to condition (b) of Invariant B) and that these two paths also block the top edges of the lower path of the group of exclusive paths. Hence, O cannot contain any path with lca of smaller level intersecting the lower path, and it can contain at most one path with lca of smaller level intersecting the reserved edge of the newly deferred paths. It suffices to remove at most three paths from O to obtain O'. If O contains only one path from $P_v \cup X_v$, it suffices to remove at most three paths from O to obtain O', because O can contain at most two paths with lca of smaller level intersecting the path accepted by the algorithm or the reserved edge of the newly deferred paths. The invariants are satisfied.

Case (c): There is only one equivalence class of paths in P_v intersecting the higher path of the group of exclusive paths, and there are at least two equivalence classes of paths in P_v intersecting the lower path of the group of exclusive paths. The algorithm accepts the higher path of the group of exclusive paths and makes the paths in P_v intersecting the lower path a group of deferred paths. All other paths in $P_v \cup X_v$ are rejected, and no edge is marked fixed anymore. The reserved edge of the group of deferred paths is the top edge shared by all these paths. If O contains two paths from $P_v \cup X_v$, note that one of the two paths must be from X_v (due to condition (b) of Invariant B) and that these two paths also block edge e_1. Hence, O cannot contain any path with lca of smaller level intersecting e_1, and it can contain at most one path with lca of smaller level intersecting the reserved edge of the newly deferred paths or the top edge of the higher path that is directed towards the leaves, because all such paths must use the edge $(p(v), v)$. It suffices to remove at most three paths from O to obtain O'. If O contains only one path from $P_v \cup X_v$, it suffices to remove at most three paths from O to obtain O', because O can contain at most two paths with lca of smaller level intersecting the path accepted by the algorithm or the reserved edge of the newly deferred paths. The invariants are satisfied.

Case 1.2.4.3: There is only one equivalence class of paths in P_v. If there is a path p in P_v that does not intersect the lower path of the group of exclusive paths, the algorithm accepts p and the lower path of the group of exclusive paths. If all paths in P_v intersect the lower path of the group of exclusive paths, the algorithm picks one path p in P_v arbitrarily and accepts p and the higher exclusive path. All other paths in $P_v \cup X_v$ are rejected, and no edge is marked fixed anymore. Hence, we have $a_v = 2$. It suffices to remove at most three paths from O in order to obtain a valid set O': if O contains two paths from $P_v \cup X_v$, it can contain at most one path with lca of smaller level that intersects a path accepted by the algorithm; if O contains only one path from $P_v \cup X_v$, removing this path and at most two paths with lca of smaller level is sufficient. Hence, the invariants are satisfied.

Case 1.2.4.4: All paths in P_v share an edge e (called *intersection edge*) incident to v, and there are at least two equivalence classes of paths in P_v. Recall that c is the child of v whose subtree contains the group of exclusive paths. Let v' be the lca of the higher exclusive path. Assume that the fixed edge of the group of exclusive paths is directed towards the root. (Otherwise, analogous arguments apply.) We distinguish two cases regarding the position of the intersection edge e with respect to c.

Case (a): $e = (c, v)$, or $e = (v, c')$ for some $c' \neq c$. The algorithm makes P_v a group of deferred paths with reserved edge e, accepts the lower path of the group of exclusive paths (note that no path in P_v intersects the lower path in this case), and rejects the other path in X_v. No edge is marked fixed anymore. Hence, we have $a_v = 1$ and $d_v = 1$. Furthermore, at most three paths in O may intersect the path accepted by the algorithm or the reserved edge of the deferred paths: one path from the group of exclusive paths, at most one path using the edge e, and at most one path using the edge $(p(v), v)$. Hence, the invariants are satisfied.

Case (b): $e = (v, c)$, or $e = (c', v)$ for some $c' \neq c$. The algorithm makes P_v a group of deferred paths with reserved edge e, and it makes the group of exclusive paths a second group of deferred paths whose reserved edges are the formerly fixed edge plus all further edges in that direction on the path from the fixed edge to v'. Hence, we have $a_v = 0$ and $d_v = 2$. Furthermore, at most three paths in O may intersect reserved edges of the new groups of deferred paths: one path from the group of exclusive paths, at most one path using the edge e, and at most one path using the edge $(v, p(v))$. Hence, the invariants are satisfied.

Case 1.2.5: $\ell = 2$, $k = m = 0$. There are two children of v whose subtrees contain a group of exclusive paths. $s = 2$ implies $P_v = \emptyset$ in this case, as any path from P_v could be combined with one exclusive path from each subtree to obtain a set of three edge-disjoint paths.

If the fixed edges of both groups of exclusive paths point in the same direction (i.e., are both directed to the root or to the leaves), the algorithm accepts the lower paths of both groups of exclusive paths. The higher paths are rejected, and no edge is marked fixed anymore. We have $a_v = 2$, and at most three paths must be removed from O to obtain a valid set O': the

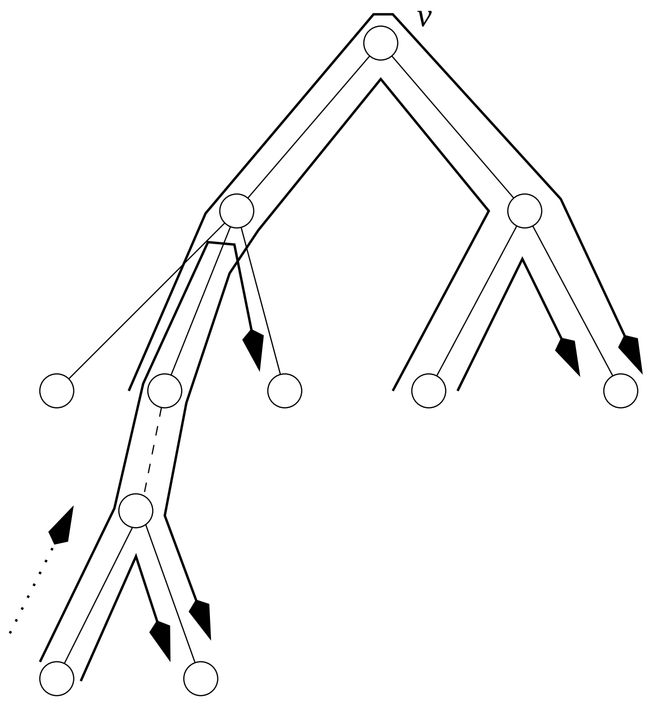


Figure 5.33: Case 1.2.6.1: P_v contains two edge-disjoint paths

two paths from the groups of exclusive paths that are contained in O, and at most one path with an lca of smaller level using the edge between v and $p(v)$ whose direction is opposite to the direction of the formerly fixed edges.

If the fixed edges of the groups of exclusive paths point in different directions (i.e., one is directed towards the root and one towards the leaves), the groups represent a pair of independent groups of exclusive paths, and the algorithm can create a new group of 2-exclusive paths. Note that O contains two paths from the new group of 2-exclusive paths already, because it contained one path from each of the two groups of exclusive paths in X_v due to condition (b) of Invariant B. Therefore, O does not change, and the invariants are still satisfied.

Case 1.2.6: $k = \ell = 1$, $m = 0$. There is one child of v that has a group of exclusive paths in its subtree, and one child of v that has an undetermined path in its subtree. All paths in P_v must intersect the undetermined path, because otherwise a path from P_v could be combined with the undetermined path and an exclusive path to obtain a set of three edge-disjoint paths.

Case 1.2.6.1: There are two edge-disjoint paths in P_v. In this case, the situation must be as shown in Figure 5.33: the two edge-disjoint paths from

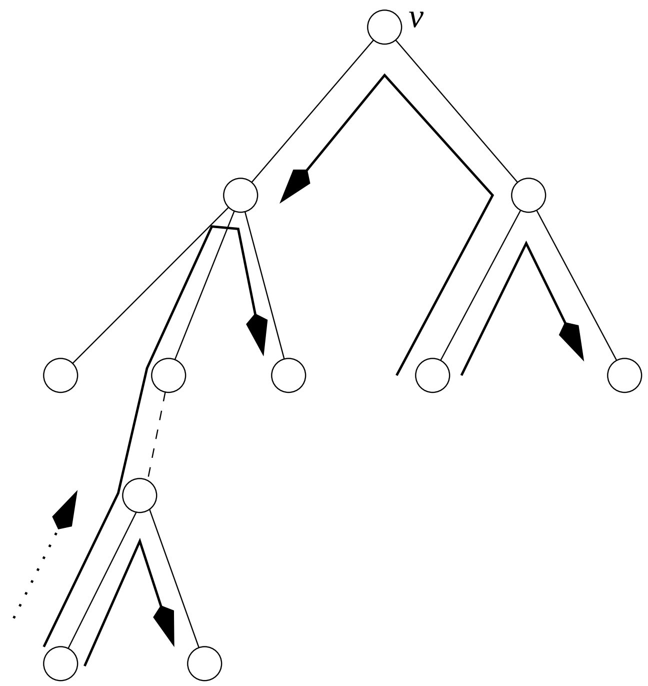


Figure 5.34: Case 1.2.6.2 (a): The fixed edge and the intersection edge have the same direction

P_v must intersect the group of exclusive paths in a way that blocks all exclusive paths from being accepted, and there cannot be any other kinds of paths in P_v.

The algorithm accepts the lower path from the group of exclusive paths and the undetermined path, and it rejects all other paths in $P_v \cup X_v$. No edge is marked fixed anymore. We have $a_v = 2$. If O contains only one path from $P_v \cup U_v \cup X_v$, removing this path and at most two additional paths with lca of smaller level is sufficient. Furthermore, note that any combination of two edge-disjoint paths from $P_v \cup U_v \cup X_v$ blocks at least three of the four top edges of the paths accepted by the algorithm. Hence, if O contains two paths from $P_v \cup U_v \cup X_v$, it can contain at most one path with lca of smaller level intersecting the paths accepted by the algorithm but not intersecting the two paths from $P_v \cup U_v \cup X_v$ contained in O. In any case, it suffices to delete at most three paths from O to obtain a valid set O'.

Case 1.2.6.2: All paths in P_v intersect the same edge e (called *intersection edge*) of the undetermined path.

Case (a): The direction of e is the same as that of the fixed edge of the group of exclusive paths (see Figure 5.34). The algorithm accepts the

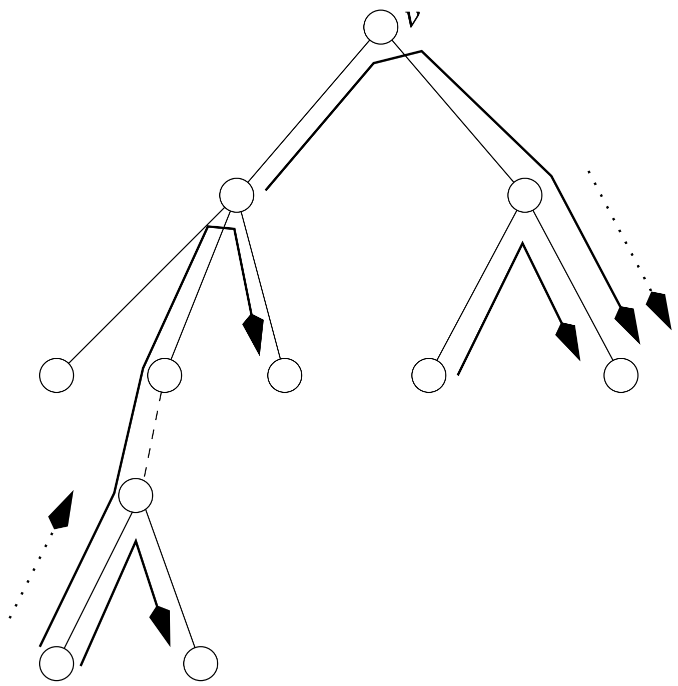


Figure 5.35: Case 1.2.6.2 (b): The fixed edge and the intersection edge have different directions, and there is a path in P_v not intersecting the higher exclusive path

undetermined path and the lower path from the group of exclusive paths. All other paths in $P_v \cup X_v$ are rejected, and no edge is marked fixed anymore. Hence, we have $a_v = 2$. If O contains only one path from $P_v \cup U_v \cup X_v$, it suffices to delete three paths from O to obtain a valid set O'. If O contains two paths from $P_v \cup U_v \cup X_v$, these paths also use the fixed edge and the intersection edge, and at most one further path from O can be blocked by the paths accepted by the algorithm. Again, it suffices to delete three paths from O to obtain a valid set O'.

Case (b): The direction of e is different from that of the fixed edge, and there is a path p in P_v that does not intersect the higher exclusive path (see Figure 5.35). The algorithm uses X_v, p and the undetermined path together to form a new group of 2-exclusive paths consisting of a pair of independent groups of exclusive paths. All other paths is P_v are rejected by the algorithm. In addition to the fixed edge of the old group of exclusive paths, the intersection edge e is marked fixed. The set O must be manipulated in order to satisfy condition (b) of Invariant B. Note that O contains one path from X_v due to condition (b) of Invariant B. If O contains the undetermined path or the path p, let $O' = O$. If O contains one path $\neq p$ from P_v, replace this path

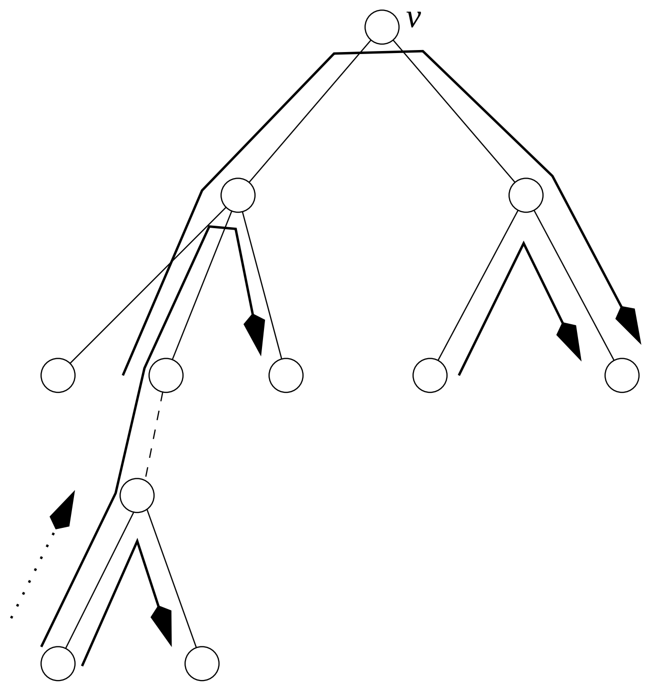


Figure 5.36: Case 1.2.6.2 (c): The fixed edge and the intersection edge have different directions, and all paths in P_v intersect the higher exclusive path

either by p or by the undetermined path (one of these must be possible). If O does not contain a path from $P_v \cup U_v$, but contains a path p' using the edge between v and $p(v)$ in the direction given by the intersection edge, replace p' either by p or by the undetermined path (one of the two must be possible). If O does not contain a path from $P_v \cup U_v$ and no path using the edge between v and $p(v)$ in the direction given by the intersection edge, add either p or the undetermined path to O. In any case, the invariants are satisfied. In particular, $|O|$ does not decrease.

Case (c): The direction of e is different from that of the fixed edge, and all paths in P_v intersect the higher exclusive path (see Figure 5.36). The algorithm accepts the undetermined path and the lower path from the group of exclusive paths, and it rejects all other paths from $P_v \cup X_v$. No edge is marked fixed anymore. We have $a_v = 2$. If O contains only one path from $P_v \cup U_v \cup X_v$, it is obviously enough to remove at most three paths from O in order to obtain a valid set O'. If O contains two paths from $P_v \cup U_v \cup X_v$, it must contain at least one of the two paths accepted by the algorithm, and the other path in O uses a top edge of the other path accepted by the algorithm. Hence, O can contain at most one further path with lca of smaller level intersecting the paths accepted by the algorithm. Again, it suffices to remove at most three paths from O in order to obtain O'.

Case 1.2.7: $m = 1$, $k = \ell = 0$. There is a subtree rooted at a child of v that contains a group of 2-exclusive paths. Therefore, we must have $P_v = \emptyset$, because any path in P_v could be combined with two paths from X_v to form a set of three edge-disjoint paths. Hence, the algorithm does nothing at node v and leaves the group of 2-exclusive paths in its unresolved state.

Case 1.3: $s \geq 3$. The algorithm accepts the s paths and rejects all other paths from $P_v \cup U_v \cup X_v$. No edge in this subtree is marked fixed anymore. As s is the maximum number of edge-disjoint paths in $P_v \cup U_v \cup X_v$, O can contain at most s paths from $P_v \cup U_v \cup X_v$. Furthermore, O can contain at most two paths from F using the edges $(v, p(v))$ or $(p(v), v)$, and these are the only two further paths in O that could possibly be blocked by the s paths accepted by the algorithm. Hence, a valid set O' can be obtained from O by deleting at most $s + 2$ paths. As $s + 2 \leq (5/3)s$, the invariants are maintained.

Case 2: $k+\ell+m > \max\{3, 2/\varepsilon\}$. Although each child of v can contain at most one undetermined path, one group of exclusive paths, or one group of 2-exclusive paths, there can nevertheless be a huge number of such unresolved paths or groups of paths in subtrees rooted at children of v, because there is no bound on the number of children of a node in T. In this case, the algorithm cannot try out all possibilities of accepting or rejecting undetermined paths or exclusive paths in polynomial time. Instead, it calculates only four candidate sets of edge-disjoint paths from $P_v \cup U_v \cup X_v$ and chooses the largest of them.

For obtaining two of the four sets, we employ a method of removing r paths from an arbitrary set S of edge-disjoint paths in P_v such that $\ell + 2m$ exclusive paths from X_v can be accepted in addition to the paths remaining in S. The resulting set of edge-disjoint paths in $S \cup X_v$ has cardinality $|S|+\ell+2m-r$. The details of the method and a proof that $r \leq (|S|+\ell+m)/3$ will be presented shortly in Lemma 5.3.2. With this tool we are ready to describe the candidate sets S_1, S_2, S_3, and S_4.

1. S_1 is obtained as the union of all k undetermined paths and a maximum number s_1 of edge-disjoint paths from P_v not intersecting any undetermined path, and as many additional edge-disjoint paths from the $\ell+m$ subtrees with exclusive paths as possible. We have $|S_1| \geq k + s_1 + m$, because S_1 contains k undetermined paths and at least m paths from groups of 2-exclusive paths in X_v due to Property (2E).

2. S_2 is obtained from S_1 by removing r of the s_1 paths in $S_1 \cap P_v$ from S_1 such that $\ell + 2m$ exclusive paths can be accepted. S_2 contains $\ell + 2m$ exclusive paths, and according to Lemma 5.3.2 only $r \leq (s_1 + \ell + m)/3$ of the s_1 paths in $S_1 \cap P_v$ were removed to obtain S_2. As S_2 still contains the k undetermined paths, we have $|S_2| \geq k + m + (2/3)(s_1 + \ell + m)$. In addition, we have $|S_2| \geq k + \ell + 2m \geq \max\{3, 2/\varepsilon\}$, because S_2 contains all undetermined paths from U_v and $\ell + 2m$ exclusive paths.

3. S_3 is obtained by taking a maximum number s_3 of edge-disjoint paths from P_v and as many additional edge-disjoint paths from the $\ell + m$ subtrees with exclusive paths and the k subtrees with undetermined paths as possible. We have $|S_3| \geq s_3 + m$, because S_3 contains at least m paths from groups of 2-exclusive paths in X_v due to Property (2E).

4. S_4 is obtained from S_3 by removing r of the s_3 paths in $S_3 \cap P_v$ from S_3 until $\ell + 2m$ exclusive paths can be accepted, in the same way as S_2 is obtained from S_1. Since $r \leq (s_3 + \ell + m)/3$ according to Lemma 5.3.2, we have $|S_4| \geq m + (2/3)(s_3 + \ell + m)$.

The algorithm accepts the paths in that set S_i with maximum cardinality and rejects all other paths from $P_v \cup U_v \cup X_v$. Recall that a_v denotes the number of paths accepted by the algorithm at node v, and note that $a_v \geq \max\{3, 2/\varepsilon\}$ follows from $|S_2| \geq \max\{3, 2/\varepsilon\}$.

We claim that the number of paths in $O_v = O \cap (P_v \cup U_v \cup X_v)$ satisfies

$$|O_v| \leq s_1 + (s_3 - s_1)/2 + k + \ell + 2m. \tag{5.1}$$

Let b' be the number of paths from P_v that are contained in O_v and that intersect at least one of the k undetermined paths. Observe that O_v can contain at most $k - b'/2$ undetermined paths. Note that the maximum number of edge-disjoint paths from P_v is s_3 and the maximum number of edge-disjoint paths from P_v not intersecting undetermined paths is s_1. If $b' \geq s_3 - s_1$, we have

$$|O_v| \leq s_3 + k - b'/2 + \ell + 2m \leq s_1 + (s_3 - s_1)/2 + k + \ell + 2m.$$

If $b' < s_3 - s_1$, we have

$$|O_v| \leq s_1 + b' + k - b'/2 + \ell + 2m \leq s_1 + (s_3 - s_1)/2 + k + \ell + 2m.$$

With this upper bound on $|O_v|$ and the lower bounds on the cardinalities of the four sets S_i, we can now prove that at least one of the sets S_i satisfies $|O_v| + 2 \leq (5/3 + \varepsilon)|S_i|$. Since it suffices to delete at most $|O_v| + 2$ paths from O in order to obtain a valid set O', this implies that the invariants are maintained.

Case 2.1: $s_3 - s_1 > (3/2)k$. Let α be such that $s_3 - s_1 = \alpha k$. Note that $\alpha > 3/2$.

Case 2.1.1: $\ell + 2m \leq (\alpha/2)k$. From (5.1) we get $|O_v| \leq s_1 + (1+\alpha)k$, and we have $a_v \geq |S_3| \geq s_3 + m \geq s_1 + \alpha k$. We obtain $|O_v| + 2 \leq a_v(1+\alpha)/\alpha + 2 \leq (5/3)a_v + 2 \leq (5/3+\varepsilon)a_v$, where the last inequality follows from $a_v \geq \max\{3, 2/\varepsilon\}$, and the invariants are satisfied.

Case 2.1.2: $\ell + 2m \geq (\alpha/2)k$. From (5.1) we get $|O_v| \leq s_1 + (1+\alpha/2)k + \ell + 2m$, and we have $a_v \geq |S_4| \geq (2/3)s_1 + (2\alpha/3)k + (2/3)\ell + (5/3)m$. We can bound the ratio between $|O_v| + 2$ and a_v as follows:

$$\begin{aligned}
\frac{|O_v|+2}{a_v} &\leq \frac{s_1 + (1+\alpha/2)k + \ell + 2m}{(2/3)s_1 + (2\alpha/3)k + (2/3)\ell + (5/3)m} + \frac{2}{a_v} \\
&\leq \frac{3}{2} + \frac{(1-\alpha/2)k}{(2/3)s_1 + (2\alpha/3)k + (2/3)\ell + (5/3)m} + \varepsilon \\
&\leq \frac{3}{2} + \frac{(1-\alpha/2)k}{(2\alpha/3)k + (\alpha/3)k} + \varepsilon \\
&\leq 1 + \frac{1}{\alpha} + \varepsilon \leq \frac{5}{3} + \varepsilon
\end{aligned}$$

Consequently, the invariants are satisfied.

Case 2.2: $(4/3)k < s_3 - s_1 \leq (3/2)k$. Let α be such that $s_3 - s_1 = \alpha k$. Note that $4/3 < \alpha \leq 3/2$.

Case 2.2.1: $\ell + 2m \leq (3/2)(\alpha - 1)k$. From (5.1) we get $|O_v| \leq s_1 + (2\alpha - 1/2)k$, and we have $a_v \geq |S_3| \geq s_3 + m \geq s_1 + \alpha k$. We obtain $|O_v| + 2 \leq a_v(2\alpha - 1/2)/\alpha + 2 \leq a_v(2 - (1/2\alpha)) + 2 \leq (5/3)a_v + 2 \leq (5/3+\varepsilon)a_v$, where the last inequality follows from $a_v \geq \max\{3, 2/\varepsilon\}$, and the invariants are satisfied.

Case 2.2.2: $\ell + 2m \geq (3/2)(\alpha - 1)k$. From (5.1) we get $|O_v| \leq s_1 + (1 + \alpha/2)k + \ell + 2m$, and we have $a_v \geq |S_2| \geq (2/3)s_1 + k + (2/3)\ell + (5/3)m$. We can bound the ratio between $|O_v| + 2$ and a_v as follows:

$$\begin{aligned}
\frac{|O_v|+2}{a_v} &\leq \frac{s_1 + (1+\alpha/2)k + \ell + 2m}{(2/3)s_1 + k + (2/3)\ell + (5/3)m} + \frac{2}{a_v} \\
&\leq \frac{3}{2} + \frac{(\alpha-1)k/2}{(2/3)s_1 + k + (2/3)\ell + (5/3)m} + \varepsilon \\
&\leq \frac{3}{2} + \frac{(\alpha-1)k/2}{k + (\alpha-1)k} + \varepsilon
\end{aligned}$$

$$\leq \ 2 - \frac{1}{2\alpha} + \varepsilon \leq \frac{5}{3} + \varepsilon$$

Consequently, the invariants are satisfied.

Case 2.3: $s_3 - s_1 \leq (4/3)k$. From (5.1) we get $|O_v| \leq s_1 + (2/3)k + k + \ell + 2m \leq s_1 + (5/3)k + \ell + 2m$, and we have $a_v \geq |S_2| \geq (2/3)s_1 + k + (2/3)\ell + (5/3)m$. We obtain $|O_v| + 2 \leq (5/3)a_v + 2 \leq (5/3 + \varepsilon)a_v$, where the last inequality follows from $a_v \geq \max\{3, 2/\varepsilon\}$, and the invariants are satisfied.

We have shown that $|O_v| + 2 \leq (5/3 + \varepsilon)a_v$ in all subcases of Case 2. To complete the description of Case 2, we still have to explain the method for removing paths from S_1 and S_3 in order to obtain S_2 and S_4, respectively. The method takes an arbitrary set S of edge-disjoint paths in P_v and removes paths from S to obtain a set S' such that every subtree with exclusive paths is touched by at most one path in S'. The motivation for this is that S can cause all paths from a group of exclusive paths to be blocked only if two paths from S intersect the corresponding subtree (Property (E)). Similarly, if only one path from a group of 2-exclusive paths can be accepted, S must contain two paths from P_v that intersect the corresponding subtree (Property (2E)).

The method proceeds as follows. Consider a graph G with the paths in S as its vertices and an edge between two paths if they touch the same child of v. G has maximum degree two and consists of a collection of chains and cycles. Note that every edge of G corresponds to a child of v that is touched by two paths in S. We are interested in the maximal parts of chains and cycles that consist entirely of edges corresponding to children of v that are the roots of subtrees with exclusive paths. There are the following possibilities for such parts:

(i) a cycle such that all paths on the cycle have both endpoints in a subtree with exclusive paths

(ii) a chain such that the paths at both ends have only one endpoint in a subtree with exclusive paths, while the internal paths have both endpoints in subtrees with exclusive paths

(iii) a chain such that the path at one end has only one endpoint in a subtree with exclusive paths, while all other paths have both endpoints in a subtree with exclusive paths

(iv) a chain such that all its paths have both endpoints in a subtree with exclusive paths

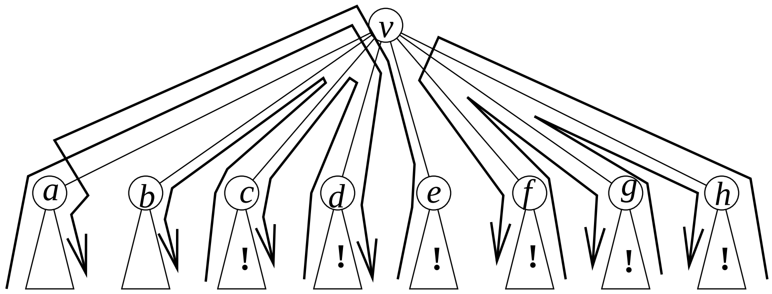


Figure 5.37: Set of edge-disjoint paths in P_v

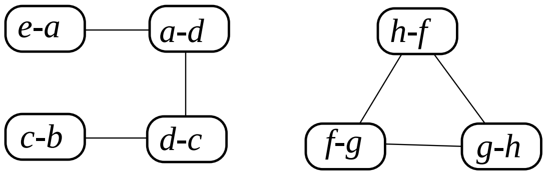


Figure 5.38: Graph G representing the structure of the paths

Note that every such maximal part of a cycle or chain consists of at least two paths, because it must contain at least one edge. The method for removing paths proceeds as follows. Cycles of even length and chains are handled by removing every other path from S, starting with the second path for chains. Cycles of odd length are handled by removing two consecutive paths in one place and every other path from the rest of the cycle.

As an example, consider the configuration depicted in Figure 5.37. The node v has eight children, a to h, and six of them (c to h) are the root of a subtree with exclusive paths (indicated by an exclamation mark). A set S of edge-disjoint paths in P_v is sketched. The graph G obtained from this set is shown in Figure 5.38, and the label of a vertex in G is u-w if the corresponding path runs from u to w in T (i.e., if the corresponding path begins in the subtree rooted at u and ends in the subtree rooted at w). With respect to (i)–(iv) above, G contains: a cycle of type (i) with length three, containing the paths f-g, g-h, and h-f, and a chain of type (ii) with length three, containing the paths a-d, d-c, and c-b. According to the rules given above, three paths would be removed from S: two paths, say f-g and g-h, from the cycle, and the path d-c from the chain of length three. After these three paths are removed from S, S doesn't contain any pair of paths with endpoints in the same subtree with exclusive paths, as required.

It is easy to see that this process always ensures that in the end S contains, for each subtree with exclusive paths, at most one path with an endpoint in that subtree. Hence, due to Properties (E) and (2E), S can be filled up with edge-disjoint exclusive paths until it contains all $\ell + 2m$ exclusive paths. Let r be the number of paths removed from S during this process. The resulting set of edge-disjoint paths has cardinality $|S| + \ell + 2m - r$.

Lemma 5.3.2 *Let v be a node with $\ell + m$ children with exclusive paths. Let $S \subseteq P_v$ be a set of edge-disjoint paths with lca v. Let $S' \subseteq S$ be the set of paths obtained from S by removing paths according to the method described above. Let $|S| = s$ and $|S \setminus S'| = r$. Then $r \leq (s + \ell + m)/3$.*

Proof: Let a be the number of cycles of type (i), and let a_i for $1 \leq i \leq a$ be the length (number of paths) of the i-th cycle. Let b be the number of chains of type (ii), and let b_i for $1 \leq i \leq b$ be the length of the i-th such chain. Let c be the number of chains of type (iii), and let c_i for $1 \leq i \leq c$ be the length of the i-th such chain. Let d be the number of chains of type (iv), and let d_i for $1 \leq i \leq d$ be the length of the i-th such chain. Note that $a_i, b_i, c_i, d_i \geq 2$ for all i. As the number of paths contained in the union of all these chains and cycles is at most s, we have

$$\sum_{i=1}^{a} a_i + \sum_{i=1}^{b} b_i + \sum_{i=1}^{c} c_i + \sum_{i=1}^{d} d_i \leq s. \tag{5.2}$$

Considering the number of children with exclusive paths covered by each chain or cycle, we obtain

$$\sum_{i=1}^{a} a_i + \sum_{i=1}^{b} (b_i - 1) + \sum_{i=1}^{c} c_i + \sum_{i=1}^{d} (d_i + 1) \leq \ell + m. \tag{5.3}$$

Using $b_i - 1 \geq b_i/2$, we use (5.3) to obtain

$$\sum_{i=1}^{a} a_i + \sum_{i=1}^{b} \frac{b_i}{2} + \sum_{i=1}^{c} \frac{c_i}{2} + \sum_{i=1}^{d} \frac{d_i}{2} \leq \ell + m. \tag{5.4}$$

Adding up (5.2) and (5.4), we obtain

$$2\sum_{i=1}^{a} a_i + \frac{3}{2}\left(\sum_{i=1}^{b} b_i + \sum_{i=1}^{c} c_i + \sum_{i=1}^{d} d_i\right) \leq s + \ell + m. \tag{5.5}$$

The number of paths removed from S to obtain S' is

$$r = \sum_{i=1}^{a} \left\lceil \frac{a_i}{2} \right\rceil + \sum_{i=1}^{b} \left\lfloor \frac{b_i}{2} \right\rfloor + \sum_{i=1}^{c} \left\lfloor \frac{c_i}{2} \right\rfloor + \sum_{i=1}^{d} \left\lfloor \frac{d_i}{2} \right\rfloor. \tag{5.6}$$

With $\lceil a_i/2 \rceil \leq (2/3)a_i$ for $a_i \geq 2$ and using (5.5), we obtain from (5.6):

$$r \leq \frac{2}{3}\sum_{i=1}^{a} a_i + \frac{1}{2}\left(\sum_{i=1}^{b} b_i + \sum_{i=1}^{c} c_i + \sum_{i=1}^{d} d_i\right) \leq \frac{s+\ell+m}{3}\,. \qquad \square$$

In the example displayed in Figure 5.37, we had $s = 7$ and $\ell + m = 6$, and it was sufficient to remove $r = 3$ paths. Indeed, $r \leq (s+\ell+m)/3 = 4\frac{1}{3}$.

By now we have specified completely how our approximation algorithm proceeds during the first and second pass. We have shown that Invariants A and B are indeed maintained; this implies that our algorithm is a $(5/3+\varepsilon)$-approximation algorithm for MaxPC with $W = 1$ (i.e., for the maximum edge-disjoint paths problem) in bidirected trees. Its running-time is polynomial in the size of the input for fixed $\varepsilon > 0$, but exponential in $1/\varepsilon$. More precisely, the running-time can be bounded by $O(4^{2/\varepsilon} \cdot q(n, L))$ for a set P of paths with maximum load L in a bidirected tree with n nodes, where q is a polynomial function. As a consequence, one can choose $\varepsilon = 1/\Theta(\log(nL))$ and still achieve running-time polynomial in the size of the input. With this modification, the algorithm achieves approximation ratio $5/3+1/\Theta(\log(nL))$ and, therefore, asymptotic approximation ratio $5/3$.

5.4 Approximating MaxPC for Arbitrary *W*

In order to obtain approximation algorithms for MaxPC with an arbitrary number W of available wavelengths, we employ a technique from [AAF$^+$96] that allows reducing the problem with W wavelengths to the problem with one wavelength with only a small increase in the approximation ratio.[1] The technique works for MaxPC in arbitrary graphs G; we discuss it here only for trees. Let an instance of MaxPC be given by a bidirected tree $T = (V, E)$, a set P of paths in T, and a number W of wavelengths. An approximation algorithm A for arbitrary number W of wavelengths is obtained from an approximation algorithm A_1 for one wavelength (i.e., for the maximum edge-disjoint paths problem) by running W copies of A_1, giving as input to the i-th copy the bidirected tree T and the set of paths that have not been accepted by the first $i-1$ copies of A_1 (see Figure 5.39). The output of A is the union of the W sets of paths output by the copies of A_1, and the paths in the i-th set are assigned color i.

[1] The author is grateful to Stefano Leonardi for pointing out the reduction from MaxPC with arbitrary number of wavelengths to MaxPC with one wavelength and to Adi Rosén for informing him about the improved analysis for the ratio obtained by this reduction in the case of identical networks for all wavelengths.

Algorithm *A*
Input: bidirected tree T, set P of paths, number W of wavelengths
Output: subsets $P_1,\ldots,P_W$ of P such that each P_i is edge-disjoint
begin
for $i = 1$ **to** W **do**
  begin
  $P_i \leftarrow A_1(T, P)$;
  $P \leftarrow P \setminus P_i$;
  end
end

Figure 5.39: Reduction from many wavelengths to one wavelength

In [AAF+96] it is shown that the algorithm A obtained using this technique has approximation ratio at most $\rho+1$ if A_1 has approximation ratio ρ, even if different wavelengths are associated with different network topologies. For identical networks, which we have in our application, the approximation ratio achieved by A can even be bounded by

$$\frac{1}{1-(1-\frac{1}{\rho W})^W}$$

which is smaller than

$$\frac{1}{1-\mathrm{e}^{-1/\rho}}$$

for all W. This bound is mentioned in a preliminary draft of the journal version of [AAF+96], which was kindly supplied to the author by Adi Rosén. The bound can be proved easily by using the fact that if A has selected p_k paths after running k copies of A_1, there is still a set of at least $(|P^*| - p_k)/W$ edge-disjoint paths among the remaining paths (this follows from a pigeonhole argument), and the next copy of A_1 accepts at least a $1/\rho$ fraction of them. As observed in the journal version of [AAF+96], this proof can be viewed as an adaptation of a similar proof to be found in [CFN77].

Since we have obtained an exact algorithm for MaxPC with $W = 1$ in bidirected trees of bounded degree and $(5/3+\varepsilon)$-approximation algorithms for MaxPC with $W = 1$ in arbitrary bidirected trees, we can employ the above technique and obtain approximation algorithms with ratio $1/(1-1/e) \approx 1.58$ for arbitrary W in bidirected trees of bounded degree and with ratio ≈ 2.22 for arbitrary W in arbitrary bidirected trees.

Chapter 6

Analysis of List-Scheduling Variants

The previous chapters have studied path coloring problems that can be viewed as restricted call-scheduling problems with unit bandwidth requirements and unit duration. In this chapter, we investigate algorithms for the general call-scheduling problem with arbitrary bandwidth requirements and (possibly unknown) duration. This problem arises in communication networks with bandwidth reservation, e.g., ATM networks (see Section 1.1.2).

The communication network is represented by a graph $G = (V, E)$ with edge capacities. We focus on the case of undirected tree networks with unit edge capacities. A call $c = (u_c, v_c, b_c, d_c)$ is specified by its communication endpoints $u_c \in V$ and $v_c \in V$, its bandwidth requirement b_c (an arbitrary rational number satisfying $0 < b_c \leq 1$), and its duration $d_c \in \mathbb{N}$. For a call c in a tree network $T = (V, E)$ we denote by P_c the (unique) undirected path from u_c to v_c in T. Recall that the load $L(e)$ of an edge e is the sum of $b_c d_c$ over all calls c with $e \in P_c$, and that $L(e)$ is a lower bound on the optimal schedule length.

Usually we denote calls by c (for call) or by r (for request), and we denote sets of calls by C or R. If the calls are arranged in a list, we denote this list by L. (The maximum load, which was also denoted by L in previous chapters, will not appear in this chapter; therefore, no confusion should arise from this.) An instance of the call-scheduling problem is given by a tree network $T = (V, E)$ and a set R of calls. We want to find a feasible schedule, i.e., an assignment of a starting time $t_c \in \mathbb{N}_0$ to every call $c \in R$ such that the sum of bandwidth requirements of calls that are active at time t and use the same edge e is at most 1, for all $t \in \mathbb{N}_0$ and all $e \in E$. A call c is active in the time interval $[t_c, t_c + d_c)$. The time interval $[t, t+1)$ for some $t \in \mathbb{N}_0$ is called *time step* t. The length $|S|$ of a schedule S is defined as $\max_{c \in R} t_c + d_c$

and is also called its makespan. The goal of the call scheduling problem is to find a feasible schedule of minimum makespan. We denote by $OPT(R)$ (or simply by OPT if R is clear from the context) the minimum makespan for R, and by $A(R)$ the length of the schedule produced by algorithm A.

Since it is $\mathcal{NP}$-hard to compute a minimum makespan schedule for calls with arbitrary bandwidth requirements in any network containing at least one edge (see Section 3.3), we analyze the performance of polynomial-time approximation algorithms. Recall that a call-scheduling algorithm A has approximation ratio ρ if $A(R)/OPT(R) \leq \rho$ for all sets R of calls and asymptotic approximation ratio ρ if $\limsup_{OPT(R)\to\infty} A(R)/OPT(R) \leq \rho$. For a particular instance R, we say that A has approximation ratio ρ *on that instance* if $A(R)/OPT(R) \leq \rho$.

We will consider the case that all calls are available at time 0 and complete information about them (including bandwidth requirements and call duration) is given to the call-scheduling algorithm in advance, i.e., we consider off-line algorithms. Some of our algorithms do not require advance knowledge of call duration, however, and we refer to these as *batch-style on-line algorithms*, although they still require that all calls are given to the algorithm in advance.

In this chapter, we will analyze the performance of variants of the classical List-Scheduling (LS) algorithm [Gra69] for call scheduling with arbitrary bandwidth requirements. These variants will be introduced in Section 6.1. First, we deal with call scheduling in stars (Section 6.2), then in arbitrary trees (Section 6.3). Among other results, we will show that $DBLS(L) \leq \lceil (8/3)\, OPT(L) \rceil$ for calls with unit duration in stars, that $LS(L) \leq 5 \cdot OPT(L)$ for calls with arbitrary duration in stars, that $LLS(L) \leq 6 \cdot OPT(L)$ for calls with unit duration in trees, and that $LSL(L) \leq 5 \log n \cdot OPT(L)$ for calls with arbitrary duration in trees with n nodes. We will also give lower bounds on the approximation ratios achieved by these algorithms in the worst case. $DBLS$, LLS, and LSL are variants of LS that will be defined shortly. LS and LSL are batch-style on-line algorithms.

6.1 Basic Algorithms

This section reviews some classical algorithms that are well-known from the multiprocessor scheduling world or from bin-packing. In subsequent sections, we will analyze the performance of these algorithms and modified versions if applied to the call-scheduling problem.

Bin-packing is the problem of packing a given set of items of variable sizes into bins of a given capacity such that the sum of the sizes of items packed into

Algorithm: List-Scheduling (*LS*)

Input: tree T and set R of calls, arranged in a list L

begin
$t \leftarrow 0$;
while not all calls have been scheduled **do**
    begin
    if $\exists$ call r in L such that bandwidth b_r
      is available at time t along path P_r in T
    then
        establish the first (in L) such call r at time t;
        remove r from L;
    else
        $t \leftarrow$ earliest time when one of the active calls finishes;
    fi
    end
end

Figure 6.1: Algorithm List-Scheduling for calls in a tree

the same bin does not exceed the capacity of that bin; the goal is to minimize the number of non-empty bins. This can be viewed as call scheduling for calls with arbitrary bandwidth requirements and unit duration in a network consisting of two nodes connected by a single edge: calls correspond to items, and time steps correspond to bins. Therefore, call scheduling in stars or trees is a generalization of bin-packing, and many techniques and results for bin-packing are relevant for call scheduling as well. Please refer to the chapter by Coffman, Garey, and Johnson [CGJ97] (in a book edited by Hochbaum) for an introduction to bin-packing and an overview of known approximation algorithms; the chapter by Graham [Gra76] (in a book edited by Coffman) is another excellent reference for bin-packing.

The call-scheduling algorithms introduced in this section assume that the set of calls is presented to them in a list L.

6.1.1 List-Scheduling

One of the earliest heuristics for the solution of scheduling problems was the List-Scheduling (*LS*) algorithm introduced by Graham [Gra69]. This algorithm, if applied to the call-scheduling problem in tree networks, is shown in Figure 6.1. We refer to schedules produced by *LS* as *list-schedules*.

Algorithm: First-Fit (*FF*)

Input: tree T and set R of calls, arranged in a list L

begin
while list L is not empty **do**
    begin
    $r \leftarrow$ first call in L;
    $t_r \leftarrow$ smallest value in $\mathbb{N}_0$ such that all edges of P_r have bandwidth b_r available in d_r consecutive time steps after t_r;
    schedule call r at time t_r;
    delete r from L;
    end
end

Figure 6.2: Algorithm First-Fit for calls in a tree

One important property of list-schedules is that if a call r is established at time t_r, it follows that at any time prior to t_r at least one of the edges on path P_r did not have bandwidth b_r available. We will use this property later as a tool to prove bounds on the approximation ratio of List-Scheduling. Note that this property holds only because there are no precedence constraints for the calls.

The length of the schedule produced by List-Scheduling can depend on the order in which the calls are arranged in the list L. Hence, sorting the calls according to certain criteria may improve the approximation ratio achieved by List-Scheduling. Therefore, we also consider the List-Scheduling variant *DBLS* (Decreasing-Bandwidth List-Scheduling), which sorts the calls according to non-increasing bandwidth requirements as a first step.

6.1.2 First-Fit

The First-Fit (*FF*) strategy has been thoroughly analyzed in the context of bin-packing [JDU+74]. Adapted to the call-scheduling scenario, First-Fit is displayed in Figure 6.2. It is easy to see that First-Fit and List-Scheduling produce the same schedule in the case of call scheduling with unit call duration. Therefore, we can derive bounds on the approximation ratio of variants of *LS* for calls with unit duration by analyzing the corresponding variant of *FF*, and vice versa.

As with List-Scheduling, schedules produced by First-Fit have a property that can be exploited for analyzing its performance: if a call r is scheduled at time step t_r, there is no time interval of d_r time steps prior to t_r such that all edges of P_r have at least b_r bandwidth available during the whole interval. In addition, at least one of the edges of P_r must have less than b_r bandwidth available (i.e., the sum of bandwidth requirements of active calls using the edge must exceed $1 - b_r$) just before t_r.

In the bin-packing scenario, it is well-known that the performance of First-Fit varies considerably when the items are arranged in a certain order before the algorithm is applied to the list. In particular, First-Fit-Decreasing (sorting the items by non-increasing size) is known to have an asymptotic approximation ratio of 11/9 as compared to 17/10 for standard First-Fit [JDU$^+$74]. Hence, we will investigate the performance of such a variant of the First-Fit algorithm also for call-scheduling problems. First-Fit-Decreasing-Bandwidth (*FFDB*) is the variant where the calls are sorted by non-increasing bandwidth requirements. Observe that *FFDB* and *DBLS* produce the same schedule for calls with unit duration.

6.2 Approximation Results for Stars

Stars occur as the subgraphs of trees that are induced by an arbitrary node of the tree and its neighbours. Hence, we encounter call-scheduling problems in stars as subproblems when we want to schedule calls in trees.

Note that there are two kinds of calls in a star T. First, there are calls that connect the central node to one of the other nodes. Second, there are calls that connect two nodes that are both adjacent to the central node. We refer to these calls as 1-*calls* and 2-*calls*, respectively.

The next two lemmas show that we can make some simplifying assumptions about worst-case examples for *LS* and its variants. These assumptions will be helpful in proving bounds on the approximation ratio of these call-scheduling algorithms.

Lemma 6.2.1 *Let A be an algorithm for call scheduling in stars that works by applying List-Scheduling to the input list of calls, possibly after sorting the list based on the bandwidth requirements or duration of the calls. If there is an input list L of calls in a star T such that $A(L) = \alpha OPT(L)$ for some $\alpha \geq 1$, then there is also an input list L' in a star T' such that $OPT(L') = OPT(L)$, $A(L') = \alpha OPT(L')$, and all calls in L' are 2-calls, i.e., they use exactly two edges of T'. Moreover, $|L| = |L'|$ and the bandwidth requirement and duration of the i-th call in L are equal to the bandwidth requirement and duration of the i-th call in L', for $1 \leq i \leq |L|$.*

Proof: We show how to construct T' and L'. Initially, let $T' = T$ and let $L' = L$. Then, for each 1-call c in L' that uses an edge e of T', add a new node u_c and a new edge e_c joining u_c and the central node to T', and replace c in L' by a 2-call c' with the same bandwidth requirement and duration that uses the edges e and e_c. It is easy to see that neither the optimal schedule length nor the schedule length produced by algorithm A is affected by this construction. □

This lemma allows us to assume that all calls in a worst-case example for *LS* or *DBLS* are 2-calls. In the case of unit call duration, we can make the same assumption on worst-case instances for *FF* or *FFDB*. The next lemma allows us to make an additional assumption regarding the structure of the schedule. We say that two list-schedules S_1 and S_2 for calls in a star are *disjoint* if the set of edges used by calls in S_1 is disjoint from the set of edges used by calls in S_2. Simlarly, we say that a schedule produced by *LS* for a list L of calls in a star *can be partitioned into two disjoint schedules* if there are two disjoint sublists L_1 and L_2 of L such that every call in L is contained in either L_1 or L_2 and the list-schedule for L_1 is disjoint from the list-schedule for L_2. Finally, we say that *the calls established before time t in a list-schedule for L can be partitioned into two disjoint schedules* if the list-schedule for L' can be partitioned into two disjoint schedules, where L' is the sublist of L containing all calls established before time t by *LS*.

Lemma 6.2.2 *Let A be as in Lemma 6.2.1. If there is an input list L of calls in a star T such that $A(L) = \alpha OPT(L)$ for some $\alpha \geq 1$, then there is an input list L' in a star T' such that $OPT(L') = OPT(L)$, $A(L') = \alpha OPT(L')$, all calls in L' are 2-calls, and the following property holds:*

> *If A schedules a call $c \in L'$ at time t_c, and if c uses the edges e_1 and e_2 of T', then the calls established before time t_c in the schedule produced by A can be partitioned into two disjoint schedules such that the calls scheduled on e_1 belong to one of the two schedules and those scheduled on e_2 belong to the other schedule.*

Furthermore, the set of bandwidth requirements of calls in L that are established at a time t in the schedule produced by A for L is equal to the set of bandwidth requirements of calls in L' that are established at time t in the schedule produced by A for L', and the same holds for the duration of these calls.

Proof: First, obtain T' and L' as in Lemma 6.2.1 so as to ensure that L' contains only 2-calls and $A(L') = \alpha OPT(L')$. Then iterate the following construction. Consider the schedule produced by A for L'. Among all calls violating the property stated in the lemma, let c be one established first by A. Assume that c uses edges e_1 and e_2. Let L_c be the sublist of L' of all calls established by LS before time t_c. Let L'_c be an identical copy of L_c that uses disjoint edges incident to the central node newly added to T'. Replace c in L' by a call c' that uses e_1 and the edge e'_2 corresponding to e_2 in the set of edges used by calls in L'_c. Every call in L' that has not been established before time t_c and that uses edge e_2 (including c) is replaced by a call that uses e'_2 instead of e_2, but is identical otherwise. Finally, every call in L'_c is inserted into L' right after its corresponding call from L_c. This concludes the description of one iteration of the construction. The desired property still holds for all calls that are established before time t_c by A (i.e., for the calls from L_c and L'_c), and now it holds for c as well. Therefore, this construction terminates after a finite number of iterations and yields T' and L' such that the desired property holds for every call. □

Like Lemma 6.2.1, this lemma applies to *LS* and *DBLS* (always) and to *FF* and *FFDB* (only in case of unit call duration). Note that the construction in the proof of Lemma 6.2.2 can produce a set L' of calls in a star T' such that the cardinality of L' and the size of T' are exponential in the size of the original instance. Since we use the construction only as a tool for analyzing the worst-case approximation ratio of *LS* variants, this exponential blow-up does not cause any problem.

6.2.1 Unit Bandwidth Requirements

In this section we assume that $b_c = 1$ for all calls c. If we assume in addition that we have unit call duration, i.e., $d_c = 1$ for all calls c, then the call-scheduling problem is equivalent to path coloring and, therefore, already $\mathcal{NP}$-hard for stars (see Corollary 3.1.6 on page 46). A polynomial-time approximation algorithm with absolute approximation ratio 4/3 and asymptotic approximation ratio 11/10 for path coloring in stars (and trees) has been obtained in Theorem 4.1.1 using an edge-coloring algorithm due to Nishizeki and Kashiwagi [NK90]. The equivalence between call scheduling in stars and edge coloring is lost once we allow arbitrary bandwidth requirements or arbitrary call duration, however. In this section we will focus on the simple heuristics List-Scheduling and First-Fit, which work for these more general settings as well. Lemma 6.2.3 and Lemma 6.2.5 below can be obtained by adapting the result given by Coffman, Garey, Johnson, and Lapaugh in [CGJL85, Corollary 12.2] for file-transfer scheduling. We give short self-contained proofs here in order to illustrate techniques that will be used again in subsequent sections.

Lemma 6.2.3 (Coffman et al., 1985) *List-Scheduling has approximation ratio* 2 *for call scheduling with unit bandwidth requirements and arbitrary duration in stars.*

Proof: Let R be the given set of calls, and let r be one of the calls that finish last in the schedule produced by List-Scheduling, i.e., a call r with $LS(R) = t_r + d_r$. By Lemma 6.2.1 we can assume that r is a 2-call. Let the edges used by r be e_1 and e_2. The List-Scheduling property implies that in each of the time steps prior to t_r either e_1 or e_2 was occupied by another call. Hence, one of these edges, say e_1, was occupied during at least $\lceil t_r/2 \rceil$ time steps prior to t_r. Therefore, the load of e_1 is at least $\lceil t_r/2 \rceil + d_r$. This implies $OPT(R) \geq \lceil t_r/2 \rceil + d_r$. Consequently, $LS(R) = t_r + d_r \leq 2 \cdot \lceil t_r/2 \rceil + d_r \leq 2 \cdot OPT(R)$. □

As First-Fit produces the same schedule as List-Scheduling for calls with unit duration, we get the following corollary immediately.

Corollary 6.2.4 *First-Fit has approximation ratio* 2 *for call scheduling with unit bandwidth requirements and unit duration in stars.*

The upper bound 2 on the approximatio ratio of LS from Lemma 6.2.3 can be shown to be tight.

Lemma 6.2.5 (Coffman et al., 1985) *There are stars and lists of calls with unit bandwidth requirements and unit duration for which the schedule computed by List-Scheduling is longer than the optimal schedule by a factor arbitrarily close to* 2.

Proof: Let n be an arbitrary integer. Consider the star with $n+1$ nodes $v_1, v_2, \ldots, v_{n+1}$ adjacent to the central node v and $n+2$ nodes altogether. The edge $\{v, v_i\}$ is denoted by e_i for $1 \leq i \leq n+1$. Now consider the following sets of calls:

$$\begin{aligned} R_1 &= \{\, n-1 \text{ copies of } (v_i, v, 1, 1) \mid 2 \leq i \leq n+1 \,\} \\ R_2 &= \{\, (v_1, v_i, 1, 1) \mid 2 \leq i \leq n+1 \,\} \end{aligned}$$

Let L and L' be lists containing the calls in $R_1 \cup R_2$. Let the calls from R_1 precede those from R_2 in L, and let the calls from R_2 precede those from R_1 in L'. It is easy to see that $LS(L) = 2n-1$ and $LS(L') = OPT(R_1 \cup R_2) = n$. Hence, $LS(L)/OPT(L) = 2 - 1/n$. Figure 6.3 shows the two schedules for the case $n = 8$: the horizontal axis represents the time, there is a dotted rectangle for every edge e_i, and every call is represented by one or two squares drawn within the rectangles of the one or two edges it uses. The shaded squares correspond to calls from R_2. □

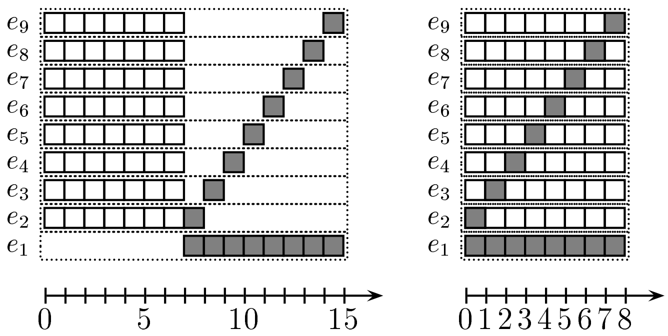


Figure 6.3: List-schedule and optimal schedule

Corollary 6.2.6 *There are stars and lists of calls with unit bandwidth requirements and unit duration for which the schedules computed by FF, FFDB and DBLS are longer than the optimal schedule by a factor arbitrarily close to* 2.

Proof: The schedule computed by *FF* is identical to the schedule computed by List-Scheduling because all calls have the same duration. Furthermore, since all calls have the same bandwidth requirement and the same duration, *FFDB* and *DBLS* will not change the ordering of the calls and, therefore, perform just as badly as standard First-Fit and List-Scheduling. □

6.2.2 Arbitrary Bandwidth, Unit Duration

In this section we assume that the duration of every call is 1, while bandwidth requirements can be arbitrary rational numbers b_r, $0 < b_r \leq 1$. While call r is active, it occupies b_r bandwidth on the one or two edges it uses, i.e., on all edges of P_r. Several active calls can share an edge if the sum of their bandwidth requirements is at most 1.

First-Fit and List-Scheduling

List-Scheduling will be shown to have approximation ratio at most 5 for calls with arbitrary bandwidth requirements and arbitrary duration in stars. Since we assume unit call duration in this section, we are able to establish a tighter bound: we will prove that the approximation ratio of *LS* is at most 4.875 in this case. As *LS* and *FF* produce the same schedule for calls with unit duration, this implies that the approximation ratio of *FF* is also at most 4.875.

Given a schedule S computed by *LS* for a list L of calls, it turns out that estimates on the approximation ratio of *LS* on that particular instance L

depend heavily on the smallest bandwidth requirement of a call that finishes last in S, i.e., at time $|S|$. The following lemmas make this relationship clearer. First, we consider the case that a call with bandwidth requirement at most $\frac{1}{3}$ finishes last in the list-schedule.

Lemma 6.2.7 *Let S be a list-schedule for a list L of calls with arbitrary bandwidth requirements and unit duration. If there is a call r with bandwidth requirement $b_r \leq \alpha$ for some $\alpha \leq \frac{1}{3}$ that finishes last in S, then*

$$|S| \leq \left\lceil \frac{2}{1-\alpha} OPT \right\rceil .$$

Proof: By Lemma 6.2.1 we can assume that r is a 2-call. Since r is blocked during the first t_r time steps, at least one of the two edges used by r has less than b_r bandwidth available during at least $\lceil t_r/2 \rceil$ time steps before t_r. Hence, the load on that edge is greater than $\lceil t_r/2 \rceil \cdot (1-\alpha)$. Since the load is a lower bound on OPT, we obtain $OPT > (t_r/2)(1-\alpha)$, which implies $t_r < (2/(1-\alpha)) \cdot OPT$ and, thus, $|S| = t_r + 1 \leq \lceil (2/(1-\alpha)) \cdot OPT \rceil$. □

Now we study the case that a call that finishes last in the list-schedule has bandwidth requirement at most $\frac{1}{2}$.

Lemma 6.2.8 *Let S be a list-schedule for a list L of calls with arbitrary bandwidth requirements and unit duration. If there is a call r with bandwidth requirement $b_r \leq \frac{1}{2}$ that finishes last in S, then $|S| \leq 3.875 \cdot OPT$.*

Proof: Let r be a call with the smallest bandwidth requirement b_r among all calls that finish last in S, i.e., among all calls that are scheduled at time $t = |S| - 1$. Lemma 6.2.1 allows us to assume without loss of generality that r is a 2-call. If $b_r \leq \frac{1}{3}$, it follows from Lemma 6.2.7 with $\alpha = \frac{1}{3}$ that

$$|S| \leq \left\lceil \frac{2}{1-\alpha} OPT \right\rceil \leq 3 \cdot OPT .$$

Therefore, assume that $\frac{1}{3} < b_r \leq \frac{1}{2}$. Consider all calls with bandwidth requirement at most $\frac{1}{3}$ that use at least one edge that is also used by r. If there is no such call, the call r is blocked in every time step prior to t_r on one of its two edges either by one call with bandwidth greater than $\frac{1}{2}$ or by two calls with bandwidth greater than $\frac{1}{3}$ each. Let the edges used by r be e_1 and e_2, and introduce the following variables (cf. Figure 6.4):

- a_1 = number of time steps before t_r during which r is blocked on e_1 by a single call, but not blocked on e_2

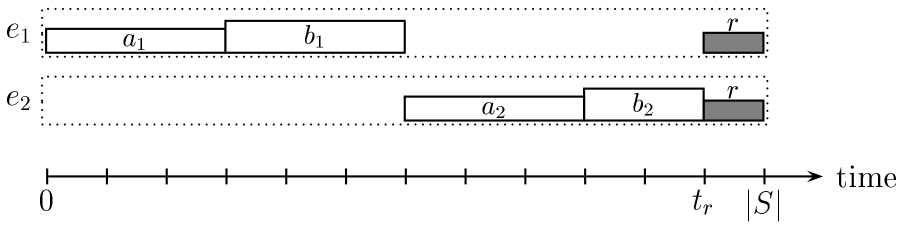


Figure 6.4: List-schedule S, $b_r \leq \frac{1}{2}$

- b_1 = number of time steps before t_r during which r is blocked on e_1 by a combination of at least two calls, but not blocked on e_2
- a_2 = number of time steps before t_r during which r is blocked on e_2 by a single call
- b_2 = number of time steps before t_r during which r is blocked on e_2 by a combination of at least two calls

Note that the time steps accounted for by these variables need not be consecutive; they are drawn as intervals in Figure 6.4 (and in all following figures) only for the sake of simplicity. The definitions made above imply:

$$\begin{aligned} |S| &= a_1 + b_1 + a_2 + b_2 + 1 & (6.1)\\ OPT &\geq a_1 + 1 & (6.2)\\ OPT &\geq a_2 + 1 & (6.3)\\ 2 \cdot OPT &\geq \frac{1}{2}(a_1 + a_2) + \frac{2}{3}(b_1 + b_2 + 1) & (6.4) \end{aligned}$$

Inequality (6.2) holds because a_1 calls with bandwidth requirements greater than $1 - b_r \geq \frac{1}{2}$ use edge e_1, and none of them can be combined with r. Inequality (6.3) holds for the same reason by considering e_2. Inequality (6.4) is derived by adding up the bandwidth requirements of calls on e_1 and e_2, and by taking into account $L(e_1) + L(e_2) \leq 2 \cdot OPT$. From (6.1) and (6.4) we can derive:

$$|S| = a_1 + a_2 + b_1 + b_2 + 1 \leq 4 \cdot OPT - \frac{b_1 + b_2}{3} \qquad (6.5)$$

If $b_1 + b_2 > \frac{3}{2}OPT$, (6.5) implies $|S| \leq 3.5 \cdot OPT$. If $b_1 + b_2 \leq \frac{3}{2}OPT$, inequalities (6.1), (6.2), and (6.3) imply $|S| \leq 3.5 \cdot OPT$.

If there are calls with bandwidth requirement at most $\frac{1}{3}$ that use at least one edge that is also used by r, let c be a call with the latest completion time $t_c + 1$ among all such calls. By Lemma 6.2.2 we may assume that c is a

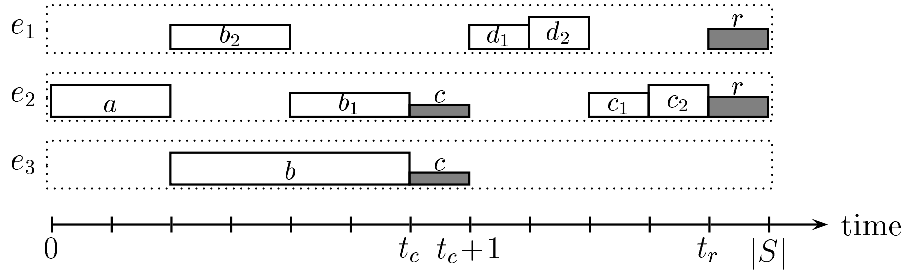


Figure 6.5: List-schedule S, $b_r \le \frac{1}{2}$, $b_c \le \frac{1}{3}$

2-call and that it does not use the same two edges as r. Let the edges used by r be e_1 and e_2, and let the edges used by c be e_2 and e_3. Introduce the following variables (cf. Figure 6.5):

- a = number of time steps during which c is blocked on e_2
- b = number of time steps during which c is blocked on e_3, but not on e_2
- b_1 = number of time steps prior to t_c during which r is blocked on e_2, but not c
- b_2 = number of time steps prior to t_c during which r is blocked on e_1, but not on e_2
- c_1 = number of time steps after t_c during which r is blocked on e_2 by a single call
- c_2 = number of time steps after t_c during which r is blocked on e_2 by a combination of at least two calls
- d_1 = number of time steps after t_c during which r is blocked on e_1 by a single call, but not blocked on e_2
- d_2 = number of time steps after t_c during which r is blocked on e_1 by a combination of at least two calls, but not blocked on e_2

Note that the time steps accounted for by these variables need not be consecutive. Using these definitions, it is clear that $|S| = a+b_1+b_2+c_1+c_2+d_1+d_2+2$. If $a+c_2+d_2 \le \frac{3}{8}OPT$, the easily observed inequalities $OPT > \frac{2}{3}(b_1+b_2)$ (follows from $b \ge b_1+b_2$ and the load on e_3), $OPT \ge c_1+1$, and $OPT \ge d_1+1$ imply $|S| \le 3.875 \cdot OPT$. If $a+c_2+d_2 > \frac{3}{8}OPT$, consider the sum of the loads on e_1 and e_2 (note that there is load greater than $\frac{1}{2}$ on e_1 or e_2 at time t_c because r is blocked, and that there is load greater than $\frac{1}{3}$ on e_1 and e_2 at time t_r because of r):

$$L(e_1) + L(e_2) > \frac{2}{3}(a + c_2 + d_2 + 1) + \frac{1}{2}(b_1 + b_2 + c_1 + d_1 + 1)$$

Since $L(e_1) + L(e_2) \leq 2 \cdot OPT$, we get $|S| < 4 \cdot OPT - \frac{1}{3}(a + c_2 + d_2 + 1) \leq 3.875 \cdot OPT$. □

Now we are ready to prove the announced general upper bound on the approximation ratio of LS and FF.

Theorem 6.2.9 *First-Fit and List-Scheduling have approximation ratio at most* 4.875 *for call scheduling with arbitrary bandwidth requirements and unit duration in stars.*

Proof: As FF and LS produce the same schedule for calls with unit duration, we consider only LS in the following. Let r be a call with the smallest bandwidth requirement b_r among all calls that finish last in S. If $b_r \leq \frac{1}{2}$, Lemma 6.2.8 implies that $|S| \leq 3.875 \cdot OPT$. Hence, we assume that $b_r > \frac{1}{2}$ from now on. Lemma 6.2.1 allows us to assume that r is a 2-call. Consider all calls c with bandwidth requirement $b_c \leq \frac{1}{2}$ that use at least one edge that is also used by r. If there is no such call, at least one of the edges used by r is occupied by a call with bandwidth requirement greater than $\frac{1}{2}$ in at least $\lceil t_r/2 \rceil + 1 \leq OPT$ time steps, and hence $|S| \leq 2 \cdot OPT$. Otherwise, let c be a call with the latest completion time $t_c + 1$ among all such calls. Lemma 6.2.2 allows us to assume that c is a 2-call that does not use the same edges as r. Let the edges used by r be e_1 and e_2, and let the edges used by c be e_2 and e_3.

Consider all calls $c' \neq c$ with bandwidth requirement $b_{c'} \leq \frac{1}{3}$ that use at least one edge that is also used by c and that are scheduled before t_c. For the case that there is no such call c' (note that $b_c \leq \frac{1}{3}$ is not excluded in this case), introduce the following variables (cf. Figure 6.6):

- a_1 = number of time steps during which c is blocked on e_2 by a single call (which must have bandwidth requirement greater than $\frac{1}{2}$ so it can block c)

- a_2 = number of time steps during which c is blocked on e_2 by a combination of at least two calls

- b_1 = number of time steps during which c is blocked on e_3 by a single call, but not blocked on e_2

- b_2 = number of time steps during which c is blocked on e_3 by a combination of at least two calls, but not blocked on e_2

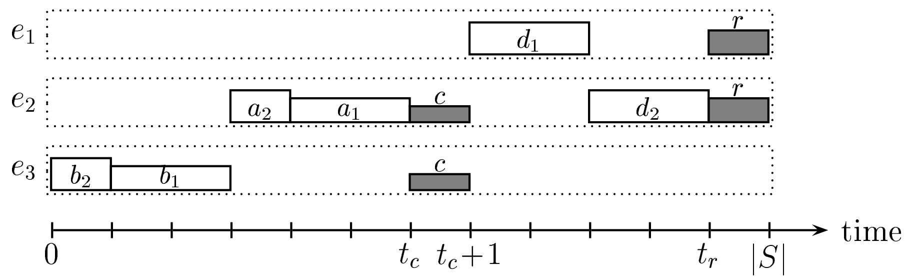


Figure 6.6: List-schedule S, $b_r > \frac{1}{2}$, $b_c \le \frac{1}{2}$

- d_1 = number of time steps after t_c during which r is blocked on e_1
- d_2 = number of time steps after t_c during which r is blocked on e_2, but not on e_1

With these definitions, it is clear that

$$|S| = a_1 + a_2 + b_1 + b_2 + d_1 + d_2 + 2$$

and that $d_1 + 1 \le OPT$, $a_1 + d_2 + 1 \le OPT$, and $b_1 \le OPT$. This implies $|S| \le 3 \cdot OPT + a_2 + b_2$. If $a_2 + b_2 \le \frac{3}{2}OPT$, we have $|S| \le 4.5 \cdot OPT$. If $a_2 + b_2 > \frac{3}{2}OPT$, we consider the load on e_2 and e_3 to obtain:

$$\frac{2}{3}(a_2 + b_2) + \frac{1}{2}(a_1 + b_1 + d_2 + 1) < 2 \cdot OPT$$

This can be transformed and combined with $d_1 + 1 \le OPT$ to give $|S| \le 5OPT - \frac{1}{3}(a_2 + b_2)$. Since $a_2 + b_2 > \frac{3}{2}OPT$, we obtain again that $|S| \le 4.5 \cdot OPT$. (In the case that $b_c \le \frac{1}{3}$, this can in fact be tightened to $|S| \le 4.25 \cdot OPT$.)

Now we deal with the case that there is at least one call $c' \neq c$ with $t_{c'} < t_c$ and with bandwidth requirement $b_{c'} \le \frac{1}{3}$ that uses at least one edge that is also used by c. Let c' be a call with latest completion time among all these calls. First, consider the case that the edge used by c and c' is e_2. Lemma 6.2.1 allows us to assume that c' is a 2-call. Let e_4 be the other edge used by c'. By Lemma 6.2.2 we can assume that $e_4 \notin \{e_1, e_2, e_3\}$. Introduce the following variables (cf. Figure 6.7):

- f = number of time steps during which c' is blocked on e_2
- b_1 = number of time steps before $t_{c'}$ during which c is blocked on e_2, but c' is not blocked on e_2

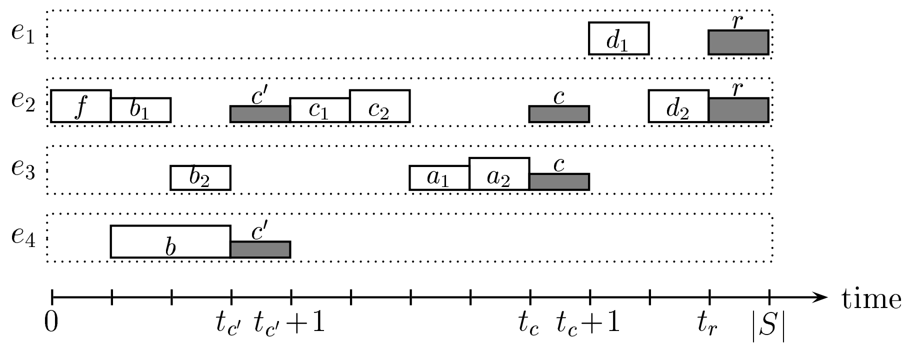


Figure 6.7: List-schedule S, $b_r > \frac{1}{2}$, $b_c \leq \frac{1}{2}$, $b_{c'} \leq \frac{1}{3}$, c and c' use e_2

- b_2 = number of time steps before $t_{c'}$ during which c is blocked on e_3, but not on e_2
- b = number of time steps during which c' is blocked on e_4, but not on e_2
- c_1 = number of time steps after $t_{c'}$ and before t_c during which c is blocked on e_2 by a single call
- c_2 = number of time steps after $t_{c'}$ and before t_c during which c is blocked on e_2 by a combination of at least two calls
- a_1 = number of time steps after $t_{c'}$ and before t_c during which c is blocked on e_3 by a single call, but not blocked on e_2
- a_2 = number of time steps after $t_{c'}$ and before t_c during which c is blocked on e_3 by a combination of at least two calls, but not blocked on e_2
- d_1 = number of time steps after t_c during which r is blocked on e_1, but not on e_2
- d_2 = number of time steps after t_c during which r is blocked on e_2

These definitions imply $b = b_1 + b_2$ and

$$|S| = f + b + c_1 + c_2 + a_1 + a_2 + d_1 + d_2 + 3\,.$$

Furthermore, note that $b \leq \frac{3}{2}OPT$ because of the load on e_4. Then observe that $d_1 + 1 \leq OPT$, $d_2 + c_1 + 1 \leq OPT$, and $a_1 + 1 \leq OPT$. Thus we get

$$|S| \leq 4.5 \cdot OPT + f + c_2 + a_2\,.$$

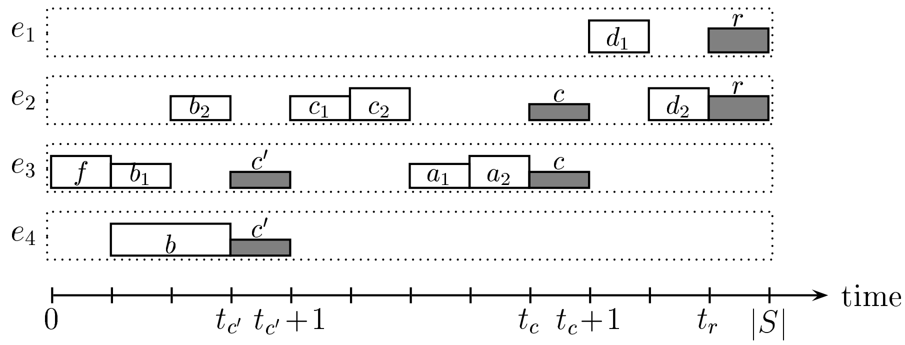


Figure 6.8: List-schedule S, $b_r > \frac{1}{2}$, $b_c \le \frac{1}{2}$, $b_{c'} \le \frac{1}{3}$, c and c' use e_3

If $f + c_2 + a_2 \le \frac{3}{8}OPT$, we have $|S| \le 4.875 \cdot OPT$. If $f + c_2 + a_2 > \frac{3}{8}OPT$, add the load on e_2 and e_3 to obtain:

$$\frac{1}{2}(b_1 + b_2 + c_1 + a_1 + d_2 + 2) + \frac{2}{3}(f + c_2 + a_2) < 2 \cdot OPT$$

Together with $d_1 + 1 \le OPT$ this implies $|S| \le 5 \cdot OPT - \frac{1}{3}(f + c_2 + a_2)$, and we obtain $|S| \le 4.875 \cdot OPT$ in this case as well.

Finally, the last remaining case is as follows: there are calls scheduled before t_c and with bandwidth requirement at most $\frac{1}{3}$ that use at least one edge that is also used by c, and a call c' with latest completion time among all these calls uses edge e_3. By Lemma 6.2.1 we can assume that c' is a 2-call. Let e_4 be the other edge used by c'. Lemma 6.2.2 allows us to assume that $e_4 \notin \{e_1, e_2, e_3\}$. Introduce the following variables (cf. Figure 6.8):

- f = number of time steps during which c' is blocked on e_3
- b_1 = number of time steps before $t_{c'}$ during which c is blocked on e_3, but c' is not blocked on e_3
- b_2 = number of time steps before $t_{c'}$ during which c is blocked on e_2, but not on e_3
- b = number of time steps during which c' is blocked on e_4, but not on e_3
- $c_1, c_2, a_1, a_2, d_1, d_2$ are defined as in the previous case

The same calculations as in the previous case show that $|S| \le 4.875 \cdot OPT$ in this final case as well. This establishes the theorem. □

Next, we try to obtain lower bounds on the approximation ratio of LS. First, we consider the case that a call with bandwidth requirement at most α

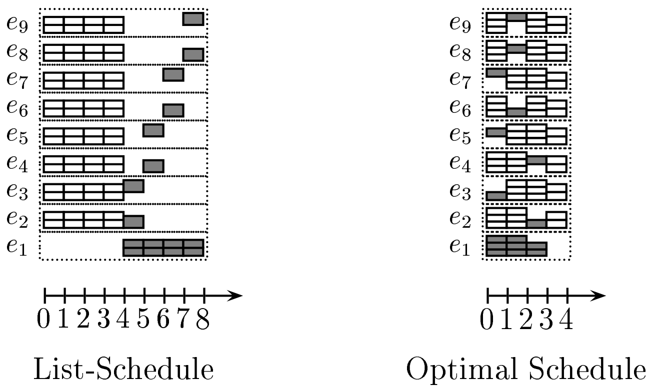


Figure 6.9: Schedules for $k' = 2$ and $\ell = 3$: $LS(L) = 8$, $OPT(L) = 4$

for some $\alpha \leq 1/3$ finishes last in the list-schedule. Lemma 6.2.7 showed that $LS(L) \leq \lceil (2/(1-\alpha)) \cdot OPT(L) \rceil$ in this case. We restrict α to values of the form $1/k$ for integers $k \geq 3$. Then Lemma 6.2.7 says that $LS(L) \leq \lceil (2k/(k-1)) \cdot OPT(L) \rceil$. Lemma 6.2.10 below proves that this bound is asymptotically tight.

In [Gra76, pp. 217–219], First-Fit bin-packing is analyzed under the restriction that all items have size at most α for some $\alpha \leq \frac{1}{2}$. With $k' = \lfloor 1/\alpha \rfloor$, it is shown that for any list of items with sizes at most α, $FF(L) \leq ((k'+1)/k')OPT + 2$ and that there are examples with $FF(L) \geq ((k'+1)/k')OPT - 1/k'$. We adapt this construction to prove the lemma.

Lemma 6.2.10 *For every $k \geq 3$, $k \in \mathbb{N}$, there are stars and lists L of calls with unit duration and bandwidth requirements smaller than $1/(k-1)$ such that a call with bandwidth requirement smaller than $1/k$ finishes last in the list-schedule for L and $LS(L)/OPT(L)$ is arbitrarily close to $2k/(k-1)$.*

Proof: Let $k' = k - 1$. Let ℓ be a positive integer such that k' divides $\ell(k'+1) - 1$. We construct a list L of calls with optimal schedule length $\ell + 1$ and list-schedule length $2(\ell(k'+1) - 1)/k'$. Let b_j^δ and $a_{1j}^\delta = \cdots = a_{k'j}^\delta$ be defined as

$$\begin{aligned} b_j^\delta &= \frac{1}{k'+1} - k'^{2j+1}\delta, \quad j = 0, 1, 2, \ldots, \ell - 1 \\ a_{ij}^\delta &= \frac{1}{k'+1} + k'^{2j}\delta, \qquad i = 1, \ldots, k',\ j = 1, 2, \ldots, \ell, \end{aligned}$$

where δ is chosen sufficiently small ($k'^{2\ell+1}\delta \ll 1$). A list of calls with exactly one call with bandwidth requirement b_j^δ for each $j = 0, 1, \ldots, \ell-1$ and one call with bandwidth requirement a_{ij}^δ for each $i = 1, 2, \ldots, k'$ and $j = 1, 2, \ldots, \ell$

Table 6.1: Values of $LS(L)/OPT(L)$ obtained by Lemma 6.2.10

k	3	4	5	6	7
$2k/(k-1)$	3	$2\frac{2}{3}$	2.5	2.4	$2\frac{1}{3}$

except the one for $i = k', j = 1$ is called a *δ-list* if the calls are ordered as follows: the a_{ij}^{δ}-calls appear in order of non-increasing bandwidth, the b_j^{δ}-calls appear in order of strictly increasing bandwidth, there are k' a_{ij}^{δ}-calls between every pair of successive b_j^{δ}-calls, and the call with bandwidth requirement $b_{\ell-1}^{\delta}$ is the second call in the list. Note that a δ-list contains $\ell(k'+1)-1$ calls.

Consider a δ-list L_δ such that all calls in L_δ are 1-calls using the same edge e. Since First-Fit bin-packing is equivalent to LS for calls with unit duration on one edge, [Gra76, pp. 217–219] implies $LS(L_\delta) = (\ell(k'+1)-1)/k'$ and $OPT(L_\delta) = \ell$. LS schedules exactly the first k' calls of the remaining list in every time step, and no time step has more than $1/(k'+1)-(k'^3-k'^2-k')\delta \le 1/(k'+1)-k'\delta$ bandwidth available on edge e in the resulting schedule. In addition, the call with bandwidth $b_0^{\delta} < 1/(k'+1) = 1/k$ is scheduled in the last time step. In an optimal schedule, the call with bandwidth b_j^{δ} can be scheduled together with the k' calls with bandwidth a_{ij}^{δ}, for $2 \le j \le \ell-1$; the calls with bandwidth b_0^{δ} and b_1^{δ} can be combined with the $k'-1$ calls with bandwidth a_{i1}^{δ}; the remaining calls are those with bandwidth $a_{i\ell}^{\delta}$, and they can be scheduled in one time step as well.

We use $\ell(k'+1)-1$ such δ-lists with 1-calls on separate edges (one edge for each δ-list). These δ-lists come first in the list L. At the end of L, we append one additional δ'-list $L_{\delta'}$, with δ' such that $k'^{2\ell-1}\delta' < k'\delta$. Let v be a vertex of the star that has not been used by any of the 1-calls. The calls in $L_{\delta'}$ all connect the vertex v to one of the vertices used by the $\ell(k'+1)-1$ δ-lists, such that no two calls in $L_{\delta'}$ connect v to the same vertex v'. Obviously, LS will schedule the calls in $L_{\delta'}$ in $(\ell(k'+1)-1)/k'$ successive time steps starting from $(\ell(k'+1)-1)/k'$. Hence, the list-schedule has length $2(\ell(k'+1)-1)/k'$. Furthermore, we have $OPT \le \ell+1$, because a schedule of length $\ell+1$ can be obtained by scheduling the calls from $L_{\delta'}$ in the first ℓ time steps and then scheduling the calls from each L_δ in ℓ out of the first $\ell+1$ time steps, skipping the time step in which a call from $L_{\delta'}$ uses the respective edge. Therefore, the approximation ratio of LS on these examples is at least:

$$\frac{2\frac{\ell(k'+1)-1}{k'}}{\ell+1} = \frac{2\ell(k'+1)-2}{(\ell+1)k'} \xrightarrow{\ell\to\infty} \frac{2(k'+1)}{k'}$$

By choosing ℓ large enough, the approximation ratio can be brought arbitrarily close to $2(k'+1)/k' = 2k/(k-1)$.

See Figure 6.9 for a sketch of the list-schedule and optimal schedule in the case $k' = 2$ and $\ell = 3$; the calls from the δ'-list are drawn as shaded rectangles. For this instance we have $LS(L) = 2 \cdot OPT$; choosing $\ell = 21$ instead of $\ell = 3$ would give $LS(L) \approx 2.8 \cdot OPT$, and $\ell = 99$ would give $LS(L) \approx 2.96 \cdot OPT$, for example. □

The lemma yields a family of examples L with $LS(L)/OPT(L)$ arbitrarily close to 3 if we set $k = 3$. For values of k ranging from 3 to 7, we obtain families of examples with $LS(L)/OPT(L)$ arbitrarily close to the values given in Table 6.1.

Now we consider the case that a call r with bandwidth requirement b_r satisfying $\frac{1}{3} < b_r \leq \frac{1}{2}$ finishes last in the list-schedule. Under this assumption, Lemma 6.2.8 tells us that the approximation ratio of LS is at most 3.875. Using a worst-case example for First-Fit bin-packing, we are able to construct call-scheduling instances for which the approximation ratio of LS is arbitrarily close to 3.7.

Lemma 6.2.11 *There are stars and lists of calls with arbitrary bandwidth requirements and unit duration such that the schedule computed by List-Scheduling or First-Fit is longer than the optimal schedule by a factor arbitrarily close to* 3.7. *The call scheduled last by List-Scheduling or First-Fit has bandwidth requirement* $\frac{1}{2}$.

Proof: Let ℓ be any positive integer divisible by 17. We describe how to construct a list L of calls with optimal schedule length $10\ell/17 + 1$ and list-schedule length $37\ell/17+1$. Following a well-known worst-case input to First-Fit bin-packing (cf. [Gra76, pp. 211-213]) with approximation ratio $\approx 17/10$, consider the following bandwidth requirements, dependent on a parameter δ ($\delta \ll 18^{-\ell/17}$):

$$
\begin{aligned}
a_{0,i} &= \frac{1}{6} + 33\delta_i & \qquad a_{1,i} &= \frac{1}{6} - 3\delta_i \\
a_{2,i} &= a_{3,i} = \frac{1}{6} - 7\delta_i & a_{4,i} &= \frac{1}{6} - 13\delta_i \\
a_{5,i} &= \frac{1}{6} + 9\delta_i & a_{6,i} &= a_{7,i} = a_{8,i} = a_{9,i} = \frac{1}{6} - 2\delta_i,
\end{aligned}
$$

where $1 \leq i \leq \ell/17$ and $\delta_i = \delta \cdot 18^{\ell/17-i}$. The $10\ell/17$ calls with these bandwidth requirements are followed by $10\ell/17$ calls with the following bandwidth requirements, $1 \leq i \leq \ell/17$:

$$
\begin{aligned}
b_{0,i} &= \frac{1}{3} + 46\delta_i & \qquad b_{1,i} &= \frac{1}{3} - 34\delta_i \\
b_{2,i} &= a_{3,i} = \frac{1}{3} + 6\delta_i & \qquad b_{4,i} &= \frac{1}{3} + 12\delta_i \\
b_{5,i} &= \frac{1}{3} - 10\delta_i & \qquad b_{6,i} &= b_{7,i} = b_{8,i} = b_{9,i} = \frac{1}{3} + \delta_i
\end{aligned}
$$

Finally, we have $10\ell/17$ calls with bandwidth requirement $\frac{1}{2} + \delta$. We call a list of $30\ell/17$ calls with such bandwidth requirements a 1.7-*list*. In the bin-packing scenario, it is known that First-Fit requires ℓ bins for a list of items with such sizes, whereas the optimal packing requires no more than $10\ell/17+1$ bins. Furthermore, each bin is filled to at least $\frac{1}{2}+\delta$ in the packing produced by First-Fit.

Let $\ell' = 10\ell/17$. The input list L for the List-Scheduling algorithm contains calls in a star with $2 + 2\ell' + 3\ell'(\ell' + 1)$ vertices adjacent to the central vertex c. These vertices are denoted u, $u_1, \ldots, u_{\ell'}$, $v_0, \ldots, v_{\ell'}$, and $w_{i,j}$ for $0 \le i \le \ell'$ and $1 \le j \le 3\ell'$. The list L contains the following calls ($\varepsilon < 1/(6\ell')$):

1. For $1 \le i \le \ell'$, $i - 1$ calls with bandwidth 1 connecting u_i and c.
2. For $1 \le i \le \ell'$, a call with bandwidth $\frac{1}{2} - \varepsilon$ connecting u and c and a call with bandwidth 3ε connecting u and u_i.
3. For $0 \le i \le \ell'$ and $1 \le j \le 3\ell'$, ℓ' calls with bandwidth 1 connecting $w_{i,j}$ and c.
4. For $0 \le i \le \ell'$, a 1.7-list of calls connecting v_i and some vertex $w_{i,j}$, such that no two calls connect v_i to the same $w_{i,j}$.
5. For $1 \le i \le \ell'$, a call with bandwidth $\frac{1}{2} + \varepsilon$ connecting u and v_i.
6. A call with bandwidth $\frac{1}{2}$ connecting u and v_0.

It is easy to verify that List-Scheduling will produce the schedule sketched in Figure 6.10. The edge $\{u, c\}$ is occupied by a call with bandwidth $\frac{1}{2} - \varepsilon$ and a call with bandwidth 3ε during the first ℓ' time steps. All 1.7-lists are scheduled from time step ℓ' to $\ell' + \ell - 1$, because every call in a 1.7-list is blocked during the first ℓ' time steps on an edge $\{w_{i,j}, c\}$. The calls with bandwidth $\frac{1}{2} + \varepsilon$ connecting u and v_i are scheduled from time step $\ell' + \ell$ to $2\ell' + \ell - 1$, because they are blocked on $\{u, c\}$ during the first ℓ' time steps and subsequently on $\{v_i, c\}$ during the next ℓ time steps. (Recall that the 1.7-lists occupy at least $\frac{1}{2} + \delta$ bandwidth from time step ℓ' to $\ell' + \ell - 1$ on all

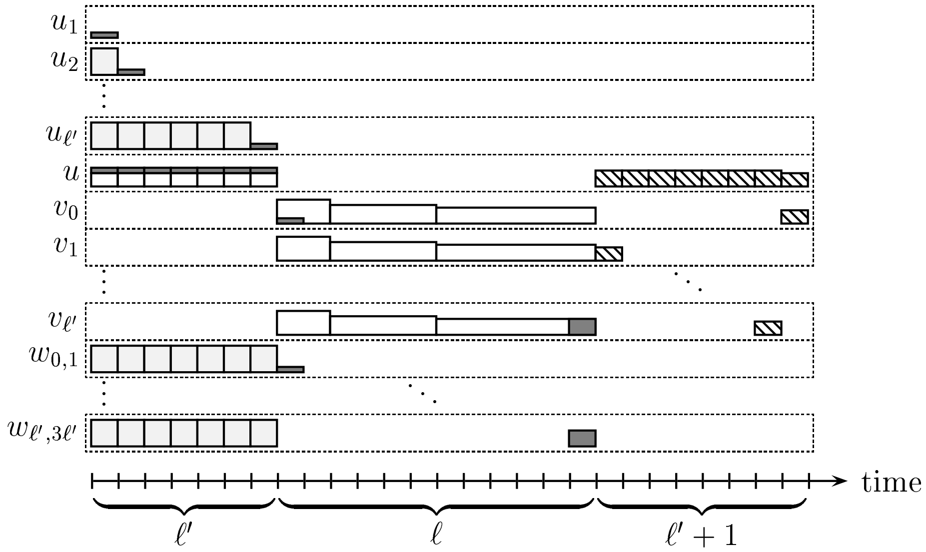


Figure 6.10: Example with $LS(L)/OPT(L) \approx 3.7$

edges $\{v_i, c\}$.) Finally, the call $(u, v_0, \frac{1}{2}, 1)$ is scheduled in time step $2\ell' + \ell$. Hence, $LS(L) = 37\ell/17 + 1$.

On the other hand, it is clear that L can be scheduled in $\ell'+1$ time steps. In particular, on edge $\{u, c\}$ one can schedule one call with bandwidth $\frac{1}{2} - \varepsilon$ and one call with bandwidth $\frac{1}{2}+\varepsilon$ during each of the first ℓ' time steps. Since ε has been chosen small enough, all calls with bandwidth 3ε together with the call $(u, v_0, \frac{1}{2}, 1)$ can then be scheduled together at time ℓ'. The 1.7-lists can be scheduled in $\ell' + 1$ time steps, such that one of the time steps has bandwidth at least $\frac{1}{2} + \varepsilon$ available. Hence, the schedule for the 1.7-list on v_i can be arranged such that the call connecting u and v_i is scheduled at that time step. Finally, the remaining 1-calls can be filled in without making the schedule longer. Hence, we have $OPT(L) \leq \ell' + 1$. This implies:

$$\frac{LS(L)}{OPT(L)} \geq \frac{\frac{37\ell}{17} + 1}{\frac{10\ell}{17} + 1} = \frac{37\ell + 17}{10\ell + 17} \xrightarrow{\ell \to \infty} 3.7$$

By choosing ℓ large enough, we obtain instances of call scheduling such that the approximation ratio of LS (or FF) on these instances is arbitrarily close to 3.7. $\square$

Note that Lemma 6.2.11 and Lemma 6.2.8 show that the exact bound on the worst-case approximation ratio of List-Scheduling lies somewhere between 3.7 and 3.875 if a call with bandwidth requirement at most $\frac{1}{2}$ finishes last in the list-schedule.

We have not been able to construct an instance where the calls that finish last in the list-schedule have bandwidth requirement greater than $\frac{1}{2}$ and where the approximation ratio of *LS* is worse than for the case where a call with bandwidth requirement at most $\frac{1}{2}$ finishes last in the list-schedule. This seems to indicate that the upper bound from Theorem 6.2.9 is not tight.

Decreasing-Bandwidth List-Scheduling

While our best general upper bound on the worst-case approximation ratio of *LS* (and *FF*) for calls with unit duration and arbitrary bandwidth requirements in stars is 4.875 (and we have examples where the ratio is arbitrarily close to 3.7), a slightly modified algorithm gives a much better approximation ratio. The algorithm Decreasing-Bandwidth List-Scheduling (*DBLS*) behaves just like standard List-Scheduling, but it sorts the given list of calls according to non-increasing bandwidth requirements before it begins to schedule the calls.

Theorem 6.2.12 *For a list L of calls with arbitrary bandwidth requirements and unit duration in a star, $DBLS(L) = FFDB(L) \leq \lceil (8/3) \cdot OPT \rceil$. Hence, DBLS and FFDB have asymptotic approximation ratio at most $\frac{8}{3} = 2\frac{2}{3}$ for call scheduling with arbitrary bandwidth requirements and unit duration in stars.*

Proof: *DBLS* and *FFDB* produce the same schedule for calls with unit duration, and we focus on *FFDB* for the remainder of the proof. Given a set R of calls, denote by L the list of calls obtained by sorting R in order of non-increasing bandwidth requirements, and denote by S the schedule produced by *FFDB*.

First, we show that the completion time t_c+1 of any call $c \in R$ with bandwidth requirement $b_c > \frac{1}{3}$ satisfies $t_c + 1 \leq 2 \cdot OPT(R)$. If c has bandwidth requirement $b_c > \frac{1}{2}$, consider the part L_c of L that contains all calls from the beginning of the list up to and including c. At the point when *FFDB* decided to schedule c at time t_c, it knew only about the calls in L_c. Hence, the schedule produced by *FFDB* for L_c already has length $t_c + 1$. Furthermore, note that no two calls in L_c can be scheduled at the same time if they use the same edge. Therefore, the optimal schedule length for L_c is the same as the optimal schedule length for a list L'_c that is identical to L_c except that the bandwidth requirements of all calls are set to 1. In addition, the lengths of the schedules produced by *FFDB* for L_c and L'_c are both equal to $t_c + 1$. Corollary 6.2.4 ensures that $t_c+1 = FFDB(L'_c) \leq 2 \cdot OPT(L'_c) = 2 \cdot OPT(L_c) \leq 2 \cdot OPT(R)$.

If c has bandwidth requirement b_c satisfying $\frac{1}{3} < b_c \leq \frac{1}{2}$, consider again the part L_c of L that contains all calls from the beginning of the list up to and including c. At the point when *FFDB* decided to schedule c at time t_c, it must have been the case that during all time steps prior to t_c, on at least one of the edges used by c more than $1-b_c$ bandwidth was occupied by other calls from L_c. Hence, there must be an edge e that is occupied to this extent during at least $\lceil t_c/2 \rceil$ time steps prior to t_c. During each such time step, the respective edge must be used either by a single call occupying more than $1-b_c$ bandwidth or by two calls occupying at least b_c bandwidth each. It is clear that even an optimal schedule requires $\lceil t_c/2 \rceil$ time steps for these calls and an additional time step for c, and thus $t_c+1 \le 2 \cdot OPT(L_c) \le 2 \cdot OPT(R)$. Hence, if a call c with bandwidth requirement $b_c > \frac{1}{3}$ is scheduled at time t_c by *FFDB*, we have $t_c + 1 \le 2 \cdot OPT(R)$.

Now we prove the theorem. Let r be a call with maximum bandwidth requirement among the calls that finish last in S. Note that $|S| = t_r + 1$. If $b_r > \frac{1}{3}$, the previous argument shows that $|S| \le 2 \cdot OPT$. If $b_r \le \frac{1}{4}$, note that at least one edge used by r has less than b_r bandwidth available during at least $\lceil t_r/2 \rceil$ time steps prior to t_r. Considering the load on that edge, we obtain $\lceil t_r/2 \rceil \cdot (1 - b_r) < OPT$. This implies $t_r < (2/(1-b_r)) \cdot OPT$ and, thus, $t_r + 1 \le \lceil (2/(1-b_r)) \cdot OPT \rceil$. With $b_r \le \frac{1}{4}$, this shows $t_r + 1 \le \lceil (8/3) \cdot OPT \rceil$.

Finally, consider the case that $\frac{1}{4} < b_r \le \frac{1}{3}$. By Lemma 6.2.1 we can assume that r is a 2-call. Denote by C the set of all calls with bandwidth requirement greater than $\frac{1}{3}$ that use at least one edge also used by r. If C is empty, r is blocked during the first t_r time steps entirely by calls d with bandwidth requirement b_d satisfying $b_r \le b_d \le \frac{1}{3}$. In addition, it is clear that two such calls are not enough to block r, because $2 \cdot \frac{1}{3} + b_r \le 1$. Therefore, whenever r is blocked on an edge during one of the first t_r time steps, at least $3b_r$ bandwidth is occupied on that edge in that time step. Since r is blocked on at least one edge during at least $\lceil t_r/2 \rceil$ time steps, we have $\lceil t_r/2 \rceil \cdot 3b_r < OPT$ and, consequently, $t_r + 1 \le \lceil (2/(3b_r)) \cdot OPT \rceil \le \lceil (8/3) \cdot OPT \rceil$, where the last inequality follows from $b_r > \frac{1}{4}$.

Now, assume that C is not empty, and let c be a call with the latest completion time among all calls in C. Note that $t_c + 1 \le 2 \cdot OPT$. Furthermore, note that starting from $t_c + 1$ call r is blocked entirely by calls d with bandwidth requirement b_d satisfying $b_r \le b_d \le \frac{1}{3}$, and that three such calls are necessary in each time step to block r. Hence, the sum of the loads on the two edges used by r gives the following lower bound on $2 \cdot OPT$:

$$(t_c + 1)(1 - b_r) + (t_r - t_c - 1)3b_r + 2b_r < 2 \cdot OPT \tag{6.6}$$

Inequality (6.6) can simply be transformed into:

$$3b_r(t_r - t_c - 1 + \frac{2}{3} + t_c + 1) < 2 \cdot OPT + (t_c + 1)(4b_r - 1) \tag{6.7}$$

Using $t_c + 1 \leq 2 \cdot OPT$, inequality (6.7) yields:

$$3b_r(t_r + \frac{2}{3}) < 2 \cdot OPT + 2 \cdot OPT(4b_r - 1) = 8b_r \cdot OPT$$

This implies $t_r + \frac{2}{3} < (8/3) \cdot OPT$ and, therefore, $t_r + 1 \leq (8/3) \cdot OPT$. Altogether, we have shown $FFDB(L) = DBLS(L) \leq \lceil (8/3) \cdot OPT(L) \rceil$ for all lists L of calls in a star with unit duration and arbitrary bandwidth requirements. □

Now we use a well-known family of worst-case instances I for First-Fit-Decreasing (*FFD*) bin-packing with $FFD(I) = (11/9)OPT(I)$ [Gra76, p. 220, Fig. 5.40] to construct instances L that provide a lower bound on the approximation ratio of *FFDB* for call scheduling. The basic idea is to exploit the fact that it suffices to block a call on only one of its two edges in each time step before it is scheduled; by blocking the call on each of its edges for approximately $(11/9)OPT$ time steps, the approximation ratio of *FFDB* can be made arbitrarily close to 22/9.

Theorem 6.2.13 *There are instances of call scheduling with arbitrary bandwidth requirements and unit duration in stars for which the approximation ratio of FFDB and DBLS is arbitrarily close to* 22/9.

Proof: We construct instances L using calls with bandwidth requirements $\alpha = \frac{1}{2} + \varepsilon$, $\beta = \frac{1}{4} + 2\varepsilon$, $\gamma = \frac{1}{4} + \varepsilon$, and $\delta = \frac{1}{4} - 2\varepsilon$. Note that $\alpha + \gamma + \delta = 1$ and $2\beta + 2\delta = 1$. For a given $n \in \mathbb{N}$, let $N = 62208n^5 + 3888n^3 + 30n$ and $M = 5184n^4 + 252n^2 + 1$, and consider a star with $N + M$ edges $e_1, \ldots, e_N, f_1, \ldots, f_M$. L contains the following calls:

(1) for $i = 1, \ldots, N$, we have $6n$ calls with bandwidth α using only edge e_i;

(2) for $i = 1, \ldots, M$ and $j = 0, \ldots, 6n-1$, we have one call with bandwidth α using edges f_i and $e_{N-(i-1)\cdot 6n-j}$;

(1') for $i = 1, \ldots, N$, we have $6n$ calls with bandwidth β using only edge e_i;

(2') for $i = 1, \ldots, M$ and $j = 0, \ldots, 6n-1$, we have one call with bandwidth β using edges f_i and $e_{N-M\cdot 6n-(i-1)\cdot 6n-j}$;

(3) for $i = 1, \ldots, N - M \cdot 12n = 864n^3 + 18n$ and $j = 0, \ldots, 6n - 1$, we have one call with bandwidth γ using edges e_i and $f_{M-(i-1)\cdot 6n-j}$;

(4) for $i = 1, \ldots, M-(864n^3+18n)\cdot 6n = 144n^2+1$ and $j = 0, \ldots, 6n-1$, we have one call with bandwidth γ using edges f_i and $e_{N-M\cdot 12n-(i-1)\cdot 6n-j}$;

(5) for $i = 1, \ldots, 12n$ and $j = 0, \ldots, 12n-1$, we have one call with bandwidth δ using edges e_i and $f_{2+(i-1)12n+j}$;

(6) for $j = 0, \ldots, 12n-1$, we have one call with bandwidth δ using edges f_1 and e_{1+j}.

Note that the calls are given in order of non-increasing bandwidth and will, therefore, be scheduled in that order by *FFDB* and *DBLS*. Furthermore, *FFDB* and *DBLS* will produce the same schedule, because the calls have unit duration; we focus on *FFDB* for the remainder of the proof. It is not difficult to see that groups (1) and (1') of calls will be scheduled by *FFDB* in the first $6n$ time steps, groups (2) and (2') in the second $6n$ time steps, group (3) from time step $12n$ to time step $14n-1$, group (4) from time step $14n$ to time step $16n-1$, group (5) from time step $16n$ to time step $19n-1$, and group (6) from time step $19n$ to time step $22n-1$. Hence, we have $FFDB(L) = 22n$.

Furthermore, the optimal schedule length is $OPT(L) = 9n+1$. To see this, note that the calls touching a fixed edge e can be scheduled in at most $9n+1$ time steps (disregarding all other edges for the moment). For example, the calls using edge f_1 can be scheduled as follows: combine three calls with bandwidth requirements α, γ and δ to fill each of $6n$ time steps, and combine two calls with bandwidth β and two calls with bandwidth δ to fill each of $3n$ additional time steps. Thus, $9n$ time steps suffice for edge f_1. For most other edges, the same kind of schedule (or a shorter schedule) exists; only for some edges (for example, edge e_1), $9n+1$ time steps are required, because there is one call in addition to the calls that fit into the schedule of length $9n$ sketched above.

Now, we claim that the schedules of length $9n+1$ for individual edges can be combined into a schedule of the same length for all calls. This follows because the edges of the star can be considered in a certain order such that the schedule for all calls can be constructed by starting with an empty schedule and then, at each edge, merging the schedule obtained so far with the schedule for calls using that edge. The order of the edges will ensure that at most one call using the current edge e also used an edge e' preceding e in the order. Consequently, the optimal schedule for the calls using the current edge can be permuted such that this one call is scheduled in a time step compatible with the schedule obtained so far, and the two schedules can be merged in the obvious way.

It remains to show that there exists an order of the edges with the required property. Let $t(e)$ denote the latest completion time of a call using edge e in the schedule produced by *FFDB* for all calls. It is easy to see that an order with the required property can be obtained by sorting the edges according to non-increasing values of $t(e)$; some care must be taken only to order edges with the same values of $t(e)$ appropriately as well. □

6.2.3 Arbitrary Bandwidth and Duration

Having examined different restricted cases in the previous sections, we will now investigate the unrestricted call-scheduling problem in stars. Call duration can be arbitrary positive integers, and bandwidth requirements can be arbitrary rational numbers b satisfying $0 < b \leq 1$.

Theorem 6.2.14 *LS is a batch-style on-line algorithm for scheduling calls with arbitrary bandwidth requirements and arbitrary duration in stars. Its competitive ratio is at most 5. If one of the calls that finish last in the list-schedule has bandwidth requirement at most $\frac{1}{2}$, the list-schedule is at most 4 times as long as an optimal schedule.*

Proof: It is clear that LS is a batch-style on-line algorithm, because it does not require advance knowledge of call duration. Let r be a call with the smallest bandwidth requirement b_r among all calls that finish last in S, i.e., at time $|S|$. Since call r is blocked during all time steps prior to t_r, we have $OPT > \lceil t_r/2 \rceil \cdot (1 - b_r) + d_r b_r$. This implies:

$$\begin{aligned} t_r &\leq \frac{2}{1-b_r} OPT - \frac{2b_r}{1-b_r} d_r \\ t_r + d_r &\leq \frac{2}{1-b_r} OPT + \frac{1-3b_r}{1-b_r} d_r \end{aligned}$$

For $b_r \leq \frac{1}{3}$, we have $1 - 3b_r \geq 0$ and, using $d_r \leq OPT$, obtain $|S| = t_r + d_r \leq ((2+(1-3b_r))/(1-b_r))OPT = 3 \cdot OPT$; for $\frac{1}{3} < b_r \leq \frac{1}{2}$, $2/(1-b_r)$ is at most 4, and with $1 - 3b_r < 0$ we obtain $|S| = t_r + d_r \leq 4 \cdot OPT$. Hence, we assume that $b_r > \frac{1}{2}$. By Lemma 6.2.1 we can assume that r is a 2-call. Consider all calls with bandwidth requirement at most $\frac{1}{2}$ that use at least one edge that is also used by r. If there is no such call, at least one of the edges used by r is blocked by a call with bandwidth requirement greater than $\frac{1}{2}$ in at least $\lceil t_r/2 \rceil + d_r \leq OPT$ time steps, and consequently, $|S| = t_r + d_r \leq 2 \cdot OPT$. Otherwise, let c be a call with the latest completion time $t_c + d_c$ among all such calls. Assume that c finishes before r is established. (Otherwise, $|S| \leq 5 \cdot OPT$ follows directly from $t_c + d_c \leq 4 \cdot OPT$, which can be shown as above, and $d_r \leq OPT$.) By Lemma 6.2.2 we can assume that c is a 2-call that does not use the same two edges as r. Let the edges used by r be e_1 and e_2, and let the edges used by c be e_2 and e_3. The list-schedule is partitioned into the following disjoint time intervals:

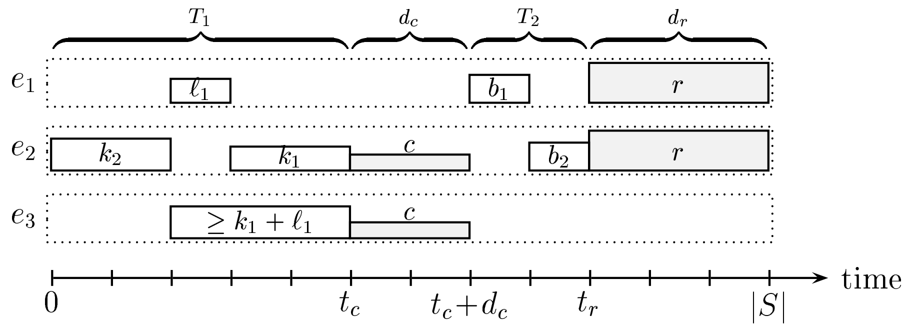


Figure 6.11: List-schedule S, $b_r > \frac{1}{2}$, $b_c \leq \frac{1}{2}$

Part A: T_1 time steps from the beginning of the schedule until t_c (the time when call c is scheduled)

Part B: d_c time steps during which call c is active

Part C: T_2 time steps from the completion time of c until t_r (the time when call r is scheduled)

Part D: d_r time steps during which call r is active

Obviously, $|S| = T_1 + d_c + T_2 + d_r$. We introduce the following variables (cf. Figure 6.11):

- b_1 = number of time steps in Part C during which r is blocked on e_1, but not on e_2
- b_2 = number of time steps in Part C during which r is blocked on e_2
- k_1 = number of time steps in Part A during which r is blocked on e_2, but not c
- k_2 = number of time steps in Part A during which c is blocked on e_2
- l_1 = number of time steps in Part A during which r is blocked on e_1, but not on e_2

Considering the load on edge e_3, we obtain $(T_1 - k_2)(1 - b_c) + d_c b_c \leq OPT$. Since $T_1 = k_1 + k_2 + l_1$, this implies:

$$k_1 + l_1 \leq \frac{1}{1 - b_c} OPT - \frac{b_c}{1 - b_c} d_c \,.$$

Adding d_c on both sides of this inequality and using $d_c \leq OPT$, $1-2b_c \geq 0$, we have:

$$k_1 + l_1 + d_c \leq \frac{1}{1-b_c}OPT + \frac{1-2b_c}{1-b_c}OPT = 2 \cdot OPT \tag{6.8}$$

In addition, the following equation and inequalities follow directly from the definitions of the variables:

$$\begin{aligned} |S| &= k_1 + k_2 + l_1 + d_c + b_1 + b_2 + d_r \\ k_2 + b_2 &\leq 2 \cdot OPT \\ b_1 + d_r &\leq OPT \end{aligned}$$

Using these and inequality (6.8), we obtain $|S| \leq 5 \cdot OPT$. □

For the case that a call with bandwidth requirement at most $\frac{1}{3}$ finishes last in the list-schedule, the next theorem gives matching upper and lower bounds on the competitive ratio of List-Scheduling.

Theorem 6.2.15 *Let L be a list of calls with arbitrary duration and bandwidth requirements. Let S be the schedule computed for list L by List-Scheduling. If there is a call with bandwidth requirement at most $\frac{1}{3}$ that finishes last in S, then $|S| \leq 3 \cdot OPT(L)$. Furthermore, for arbitrary $k > 2$, $k \in \mathbb{N}$, there are stars and lists L of calls such that a call with bandwidth requirement $1/k$ finishes last in the list-schedule and $LS(L)/OPT(L)$ is arbitrarily close to* 3.

Proof: First, we prove the upper bound. Let r be a call that finishes last in S and that has bandwidth requirement $b_r \leq \frac{1}{3}$ and duration d_r. Since r is blocked on at least one edge during at least $\lceil t_r/2 \rceil$ time steps prior to t_r, the load on that edge is more than $\lceil t_r/2 \rceil \cdot (1-b_r) + d_r b_r$. Hence, we can calculate as follows (using $1 - 3b_r \geq 0$ and $d_r \leq OPT$):

$$\begin{aligned} \frac{t_r}{2}(1-b_r) + d_r b_r &< OPT \\ t_r + d_r &< \frac{2}{1-b_r}OPT + \frac{1-3b_r}{1-b_r}d_r \\ t_r + d_r &< \frac{2 \cdot OPT + (1-3b_r)OPT}{1-b_r} = 3 \cdot OPT \end{aligned}$$

Hence, $|S| = t_r + d_r < 3 \cdot OPT$.

For the lower bound on the competitive ratio of List-Scheduling, fix an arbitrary integer $k > 2$ and an arbitrary positive integer ℓ. We construct a list of calls such that $LS(L) = 3\ell$, $OPT(L) = \ell + 1$, and the call that finishes last in the list-schedule for L has bandwidth requirement $1/k$.

The star used for the construction has $k\ell + 3$ nodes: a central node c, and nodes $u, v, u_1, \ldots, u_\ell, v_1, \ldots, v_{(k-1)\ell}$ adjacent to c. The list L contains the following calls (ε is chosen sufficiently small):

(1) For $i = 1, \ldots, \ell$: $k-1$ calls $(u, c, 1/k, 1)$, $k(i-1)$ calls $(u_i, c, 1/k, 1)$, and one call $(u_i, u, \varepsilon, 1)$.

(2) For $i = 1, \ldots, (k-1)\ell$: $k\ell$ calls $(v_i, c, 1/k, 1)$.

(3) For $i = 0, \ldots, \ell - 1$: for $j = 1, \ldots, k-1$: one call $(v, v_{i(k-1)+j}, \beta_{i,j}, 1)$, where the bandwidth requirement $\beta_{i,j}$ is specified below.

(4) One call $z = (u, v, 1/k, \ell)$.

For δ chosen sufficiently small, the $\beta_{i,j}$ are defined as follows:

$$\begin{aligned} \beta_{i,1} &= \frac{1}{k} + k\delta_i \\ \beta_{i,2} &= \frac{1}{k} - \delta_i \\ &\vdots \\ \beta_{i,k-1} &= \frac{1}{k} - \delta_i \end{aligned}$$

Here, $\delta_0 = \delta$ and $\delta_{i+1} = ((k-2)/k)\delta_i$.

The schedule produced by List-Scheduling for the list L is as follows. The calls (1) fill the edge $\{u, c\}$ to $(k-1)/k + \varepsilon$ during the first ℓ time steps. Each of the calls with bandwidth ε is blocked on one of the edges $\{u_i, c\}$ in all time steps before its starting time. The calls (2) fill the edges $\{v_i, c\}$ completely during the first ℓ time steps. The calls (3) are scheduled in time steps ℓ to $2\ell - 1$, because each call is blocked on a different edge $\{v_i, c\}$ during the first ℓ time steps and blocked on the edge $\{v, c\}$ from time step ℓ up to its starting time minus one. Exactly $k-1$ calls (3) are scheduled in each time step, because their bandwidth requirements add up to $(k-1)/k + 2\delta_i$ and, therefore, block all subsequent calls (3). Finally, call z is scheduled at time 2ℓ, because it is blocked on $\{u, c\}$ during the first ℓ time steps and on $\{v, c\}$ during the second ℓ time steps. Hence, $LS(L) = 3\ell$.

In an optimal schedule, call z is scheduled at time 0. In each of the first ℓ time steps, $k-1$ calls from (1) using edge $\{u, c\}$ and with bandwidth requirement $1/k$ can be scheduled together with z. All the calls from (1) with bandwidth ε are scheduled together at time ℓ. The remaining calls from (1) can easily be scheduled in the remaining free time slots during the first ℓ time steps.

Among the calls from (3), the $k-1$ calls with bandwidth requirements $\beta_{i,2}, \ldots, \beta_{i,k-1}, \beta_{i+1,1}$ are scheduled together in time step i, for $0 \leq i \leq \ell - 1$. (For $i = \ell - 1$, there is no call with bandwidth $\beta_{i+1,1}$, and only $k-2$ of the calls from (3) are scheduled in time step $\ell - 1$.) The bandwidth requirements of the calls from (3) scheduled during one of the time steps $0, \ldots, \ell - 2$ add up to exactly $(k-1)/k$. Hence, they can be scheduled concurrently with call z. The call with bandwidth $\beta_{0,1}$ is scheduled at time ℓ. Now, the calls from (2) can easily be scheduled in the remaining free time slots during the first $\ell + 1$ time steps. Therefore, $OPT = \ell + 1$. □

Note that the calls used in the proof of this theorem have bandwidth requirements smaller than $1/(k-1)$.

6.3 Approximation Results for Trees

In the previous section we have seen that variants of List-Scheduling achieve small, constant approximation ratios or competitive ratios for call scheduling in stars. Now we investigate the performance of LS variants for call scheduling in trees. For the case of unit duration (Section 6.3.1), we present a variant with constant approximation ratio; for the case of arbitrary duration (Section 6.3.2), we obtain a batch-style on-line variant with logarithmic competitive ratio.

6.3.1 Arbitrary Bandwidth, Unit Duration

It is known that the performance of LS can be arbitrarily bad in trees or even in chains if arbitrary bandwidth requirements are allowed. Feldmann et al. gave a list of calls with unit duration in a chain with $n+1$ nodes such that the approximation ratio of LS is $\Omega(n)$ on that instance [FMS+95].

Therefore, we consider the following variant of the basic List-Scheduling algorithm. Pick an arbitrary node of the tree network as the root and recall that the level of a node of the tree is its distance (number of edges) from the root. (The root has level 0.) Let m_r be that node on P_r (the path corresponding to call r) whose level is minimum among all nodes on P_r. The level of a call r is defined to be equal to the level of the node m_r. We consider the Level-List-Scheduling algorithm (LLS), which is identical to List-Scheduling except that it sorts the list of calls according to non-decreasing levels before it starts to schedule the calls.

Theorem 6.3.1 *LLS has approximation ratio at most* 6 *for call scheduling with arbitrary bandwidth requirements and unit duration in trees.*

Proof: Let S be a schedule computed by LLS for a given set R of calls. First, we show that any call r with bandwidth requirement $b_r \leq \frac{1}{2}$ finishes no later than at time $4 \cdot OPT$. To see this, consider the node m_r, and let e_1 and e_2 be the edges incident to m_r that are used by r. (If r uses only one edge incident to m_r, it can be proved by similar arguments that $t_r + 1 \leq 2 \cdot OPT$.) It is clear that call r is blocked either on edge e_1 or on edge e_2 by calls with equal or smaller level during all time steps prior to t_r. Hence, at least one of these edges has less than $\frac{1}{2}$ bandwidth available during at least $\lceil t_r/2 \rceil$ time steps prior to t_r. Therefore, $OPT > \frac{1}{2}\lceil t_r/2 \rceil$ and, consequently, $t_r + 1 \leq 4 \cdot OPT$.

Now, let r be a call with minimum bandwidth requirement b_r among all calls that finish last in S. If $b_r \leq \frac{1}{2}$, the argument above implies $|S| \leq 4 \cdot OPT$. Therefore, assume that $b_r > \frac{1}{2}$. Let e_1 and e_2 be the edges incident to m_r that are used by r. Again, it is clear that call r is blocked either on edge e_1 or on edge e_2 by calls with equal or smaller level during all time steps prior to t_r. Let c be a call with bandwidth requirement $b_c \leq \frac{1}{2}$ that has the latest completion time among all such calls. (If no such call exists, call r is blocked only by calls with smaller or equal level and with bandwidth requirements greater than $\frac{1}{2}$, and $|S| \leq 2 \cdot OPT$.) The argument above implies $t_c + 1 \leq 4 \cdot OPT$, and $t_r - t_c \leq 2 \cdot OPT$ follows from the fact that call r is blocked by calls with bandwidth requirements greater than $\frac{1}{2}$ either on e_1 or on e_2 during all time steps from $t_c + 1$ to t_r. Combining these inequalities, we obtain $|S| = t_r + 1 \leq 6 \cdot OPT$. □

Note that the proof technique from Theorem 6.2.9, which established that $LS(L) \leq 4.875 \cdot OPT$ for calls with unit duration in stars, cannot be used directly to improve the upper bound of 6 on the approximation ratio of LLS for calls with unit duration in trees. The reason is that in the case of trees a call with small bandwidth requirement can block the call r that finishes last in the list-schedule without being itself blocked on any of the two top edges used by r; therefore, it cannot be concluded that the load on one of the top edges used by r must be high, like we did in the proof of Theorem 6.2.9.

Nevertheless, we suspect that the exact worst-case approximation ratio of LLS for calls with unit duration in trees is in fact smaller than 6.

6.3.2 Arbitrary Bandwidth and Duration

Finally, we deal with the most general variant of the call-scheduling problem considered in this chapter: we allow calls with arbitrary bandwidth requirements and arbitrary duration in tree networks of arbitrary degree. We will obtain a batch-style on-line algorithm with competitive ratio $5 \log n$ for trees with n nodes. This improves on the batch-style on-line algorithm due to Feldmann, for which competitive ratio $12 \log n$ was proved only for the special case of binary trees [Fel95].

Given a tree network T with n nodes, we use a well-known technique [ABFR94, BL97] based on a tree separator [vL90] to assign separator-levels to the nodes of T as follows:

- Choose a node v whose removal splits T into subtrees $T_1, T_2, \ldots, T_k$ with at most $n/2$ nodes each. Assign node v the separator-level 0.
- In each subtree T_i of size n_i, find a node v_i whose removal splits T_i into subtrees with at most $n_i/2$ nodes. Assign all such nodes v_i the separator-level 1.
- Continue recursively until every node of T is assigned a separator-level.

This way every node of T is assigned a separator-level ℓ, $0 \leq \ell \leq \log n$. For each call $r = (u, v, b, d)$ in T, the level of r is defined to be the smallest separator-level of all nodes on the path P_r from u to v in T. In addition, the *root* node of r is defined to be that node on P_r whose separator-level is equal to the level of r. (Note that the root node is uniquely determined; if two nodes of equal separator-level are on a path P, there must exist a node of smaller separator-level on P.) Given a list L of calls in T, L_ℓ is the sublist of L that contains all calls of level ℓ, $0 \leq \ell \leq \lceil \log n \rceil$. Note that scheduling a list L_ℓ is equivalent to scheduling calls in a number of disjoint stars: calls in L_ℓ with the same root node intersect if and only if they use the same edge incident to that root node; calls in L_ℓ with different root nodes never intersect. Therefore, $LS(L_\ell) \leq 5 \cdot OPT(L_\ell)$ as a consequence of Theorem 6.2.14. The algorithm List-Scheduling by Levels (*LSL*) is shown in Figure 6.12; it simply uses List-Scheduling to schedule the lists L_ℓ, $0 \leq \ell < \lceil \log n \rceil$, one after another. ($L_{\lceil \log n \rceil}$ is empty, because the root node of a call can never have separator-level $\lceil \log n \rceil$.) Note that *LSL* is a batch-style on-line algorithm, because it does not require advance knowledge of call duration. Hence, we obtain the following theorem.

Theorem 6.3.2 *LSL is a batch-style on-line algorithm for scheduling calls with arbitrary bandwidth requirements and arbitrary duration in trees. Its competitive ratio is at most* $5 \log n$.

For comparison, note that a lower bound of $\Omega(\log \log n / \log \log \log n)$ on the competitive ratio of any deterministic batch-style on-line algorithm was obtained by Feldmann for call scheduling with arbitrary (unknown) duration in tree networks, even in the case of unit bandwidth requirements [Fel95].

Algorithm: List-Scheduling by Levels (*LSL*)

Input: tree T and set R of calls, arranged in a list L

begin
for $\ell = 0$ to $\lceil \log n \rceil - 1$ **do**
  begin
  schedule L_ℓ with List-Scheduling;
  wait until all active calls have finished;
  end
end

Figure 6.12: Algorithm List-Scheduling by Levels

Chapter 7

Conclusion

In this final chapter we will first, in Section 7.1, summarize and discuss the results presented in the previous chapters and then, in Section 7.2, give possible directions for future research.

7.1 Summary of Results

We have studied optimization problems that model the allocation of resources to individual connections in modern network architectures. In particular, we investigated the problem of assigning wavelengths to connections in all-optical networks with wavelength-division multiplexing and the problem of call scheduling in networks with bandwidth reservation. For all problems considered, we have given a full account of the boundary between tractable and intractable ($\mathcal{NP}$-hard) versions:

- Call scheduling for calls with arbitrary bandwidth requirements is $\mathcal{NP}$-hard in any network topology.
- Call scheduling for calls with arbitrary duration is $\mathcal{NP}$-hard in any network topology containing either a node of degree at least 3, or a pair of nodes with at least two edge-disjoint paths between them, or a path of length 8.
- Path coloring is $\mathcal{NP}$-hard for bidirected and undirected rings.
- Path coloring is $\mathcal{NP}$-hard for bidirected trees even in the binary case, and for undirected trees in the case of arbitrary degree; it can be solved optimally in polynomial time for undirected trees of bounded degree.

- MaxPC and MaxPP in bidirected trees can be solved optimally in polynomial time if the degree of the tree and the number of available wavelengths are both bounded by a constant; they are $\mathcal{NP}$-hard for bidirected binary trees if the number of wavelengths can be arbitrary, and for bidirected trees of arbitrary degree even in the case of a single available wavelength.

These complexity results show that only very restricted versions of the optimization problems under consideration can be solved optimally in polynomial time. This implies that the goal should be to devise efficient approximation algorithms and on-line algorithms that perform well on practical instances and for which a good bound on the worst-case performance can be proved. Our main results are two such approximation algorithms for the path coloring problem and the maximum edge-disjoint paths problem in bidirected tree networks:

- An efficient algorithm for path coloring in bidirected trees that uses at most $\lceil (5/3)L \rceil$ colors for sets of paths with maximum load L.

- For every fixed $\varepsilon > 0$, an efficient $(5/3 + \varepsilon)$-approximation algorithm for the maximum edge-disjoint paths problem in bidirected trees.

Our algorithm for path coloring in bidirected trees is the best known algorithm for this problem, and it improves on previous algorithms that used at most $2L - 1$ colors, $(15/8)L$ colors, and $(7/4)L$ colors, respectively. The algorithm shows that any set of paths with maximum load at most $(3/5)W$ can be colored using W colors. In a network with full wavelength converters, however, any set of paths with maximum load at most W can be established using W wavelengths (colors). Therefore, our algorithm and its analysis provides a basis for assessing the trade-off between the cost for wavelength converters and the cost for additional wavelengths required in networks without converters. Besides, the algorithm can easily be implemented in a distributed network because it is a local greedy algorithm, and its average-case performance can be improved by additional heuristics.

Our algorithm for the maximum edge-disjoint paths problem is the first known approximation algorithm for this problem in bidirected trees. It can be converted into an approximation algorithm for MaxPC with arbitrary number W of wavelengths, and it achieves approximation ratio 2.2 in this case.

In addition, we have analyzed variants of List-Scheduling (LS) for call scheduling in undirected stars and trees. In the case of unit call duration, we have given variants of LS with asymptotic approximation ratio at most 8/3 for calls with arbitrary bandwidth requirements in stars, and with approximation ratio at most 6 for calls with arbitrary bandwidth requirements in trees. In the case of arbitrary call duration, our variants of LS do not require advance knowledge of the duration of a call; hence, they are batch-style online algorithms. We have devised variants with competitive ratio at most 5 for calls with arbitrary bandwidth requirements and arbitrary, unknown duration in stars, and with competitive ratio at most $5 \log n$ for calls with arbitrary bandwidth requirements and arbitrary, unknown duration in trees with n nodes. It seems likely that our upper bounds are not tight, and only contrived examples can force the LS variants to have approximation ratio moderately close to our upper bounds; therefore, we suspect that variants of LS produce schedules with makespan close to the optimal makespan in practice.

7.2 Directions for Future Research

There are a number of promising directions for future research. In particular, it would be interesting to see whether our approximation algorithms for bidirected trees can be improved. For path coloring, it is known that no local greedy algorithm can use less than $\lfloor (5/3)L \rfloor$ colors in the worst case, but it is open whether a different approach might lead to an improved approximation ratio. Further results regarding the non-approximability of path coloring in bidirected trees would also be interesting; we have shown that no approximation algorithm can achieve absolute approximation ratio smaller than 4/3 unless $\mathcal{P} = \mathcal{NP}$, but no lower bound is known regarding the asymptotic approximation ratio. It is also desirable to determine how many colors are required in an optimal coloring for paths with maximum load L in the worst case; in particular, it is not known whether there are instances with arbitrarily large L for which more than $\lceil (5/4)L \rceil$ colors are necessary [KS97].

The maximum edge-disjoint paths problem for bidirected trees was proved $\mathcal{APX}$-hard in Chapter 5, so we cannot expect a polynomial approximation scheme unless $\mathcal{P} = \mathcal{NP}$; however, it is an open question whether an algorithm with approximation ratio smaller than 5/3 can be found. It might even be possible to improve the algorithm presented in Chapter 5 to achieve approximation ratio 3/2 or even smaller. Furthermore, it would be interesting to know whether there is an underlying reason for the fact that our best algorithms for path coloring and for the maximum edge-disjoint paths problem in bidirected trees achieve the same approximation ratio 5/3. The problems are dual to each other in some sense, but our algorithms proceed very differently and it is not clear why the same approximation ratio is achieved for both problems. In addition, it would be interesting to see whether techniques we used in the $(5/3+\varepsilon)$-approximation for MaxPC and MaxPP with $W = 1$ can lead to improved approximation algorithms for the integral multicommodity flow problem in trees or for special cases thereof; for the latter problem, the best known approximation is still the 2-approximation from [GVY93].

We used a reduction from MaxPC with arbitrary number of wavelengths to the maximum edge-disjoint paths problem in order to obtain approximation algorithms for MaxPC; a different approach that tackles the MaxPC problem directly might lead to a better approximation ratio, but no such approach is known so far.

Regarding the analysis of variants of List-Scheduling for call scheduling in stars and trees (Chapter 6), the comparatively large gaps between upper and lower bounds on the worst-case approximation ratio for some versions of the problem are unsatisfactory; it is a challenging problem to tighten these bounds. Furthermore, there is also a gap between the upper and lower bounds on the competitive ratio of any batch-style on-line algorithm for call scheduling in trees: our algorithm implies an upper bound of $O(\log n)$ for trees with n nodes, and Feldmann's lower bound [Fel95] is $\Omega(\log\log n/\log\log\log n)$.

Our results were obtained for models that make a number of simplifying assumptions regarding the architecture of the communication network and regarding the objectives of the optimization problems; it would be desirable to generalize the results to more flexible models. In particular, the following generalizations appear reasonable:

- All-optical networks with limited wavelength converters and with different sets of available wavelengths on different links.
- Weighted versions of MaxPC and MaxPP, where every connection request is associated with a certain benefit and the goal is to maximize the sum of the benefits of the accepted requests.
- Networks with different capacities on different links.
- Call scheduling with a mixture of unidirectional and bidirectional calls in bidirected networks (a bidirectional call would require reservation of bandwidth on two directed paths with opposite directions between the endpoints of the call).[1]
- Approximation algorithms and on-line algorithms for call scheduling that try to minimize the average response time (which is impossible for deterministic algorithms in the worst case, however).

[1] It is easy to see that our results from Chapter 6 generalize to the case of unidirectional calls in bidirected networks directly.

- Call scheduling for calls with release times, deadlines, or precedence constraints.
- Call scheduling, path coloring, path packing, MaxPC, MaxPP, and the maximum edge-disjoint paths problem for network topologies other than trees, and for fully on-line scenarios with dynamic arrivals of connection requests with finite and possibly unknown duration.

Regarding network topologies, an interesting starting point would be the class of graphs with bounded treewidth (also called partial k-trees) [Bod93]. These graphs resemble trees in some well-defined sense, and it is often possible to generalize techniques from algorithms for trees to algorithms for graphs with bounded treewidth. For example, the algorithm for MaxPC with bounded number of wavelengths in trees of bounded degree (Section 5.1.2) can in fact be generalized to graphs with bounded treewidth and of bounded degree.

Bibliography

[AAF+96] Baruch Awerbuch, Yossi Azar, Amos Fiat, Stefano Leonardi, and Adi Rosén. On-line competitive algorithms for call admission in optical networks. In *Proceedings of the 4th Annual European Symposium on Algorithms ESA '96*, LNCS 1136, pages 431–444, 1996.

[ABFR94] Baruch Awerbuch, Yair Bartal, Amos Fiat, and Adi Rosén. Competitive non-preemptive call control. In *Proceedings of the 5th Annual ACM–SIAM Symposium on Discrete Algorithms SODA '94*, pages 312–320, 1994.

[ABNC+94] Alok Aggarwal, Amotz Bar-Noy, Don Coppersmith, Rajiv Ramaswami, Baruch Schieber, and Madhu Sudan. Efficient routing and scheduling algorithms for optical networks. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms SODA '94*, pages 412–423, 1994.

[ABNC+96] Alok Aggarwal, Amotz Bar-Noy, Don Coppersmith, Rajiv Ramaswami, Baruch Schieber, and Madhu Sudan. Efficient routing in optical networks. *Journal of the ACM*, 46(6):973–1001, November 1996.

[ACKP97] Vincenzo Auletta, Ioannis Caragiannis, Christos Kaklamanis, and Pino Persiano. Bandwidth allocation algorithms on tree-shaped all-optical networks with wavelength converters. In *Proceedings of the 4th International Colloquium on Structural Information and Communication Complexity SIROCCO'97*, pages 24–39. Carleton Scientific, 1997.

[ACKP98a] Vincenzo Auletta, Ioannis Caragiannis, Christos Kaklamanis, and Pino Persiano. On the complexity of wavelength converters. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science MFCS'98*, LNCS 1450, pages 771–779. Springer-Verlag, 1998.

[ACKP98b] Vincenzo Auletta, Ioannis Caragiannis, Christos Kaklamanis, and Pino Persiano. Efficient wavelength routing in trees with low-degree converters. In *Proceedings of the DIMACS Workshop on Optical Networks (March 16–19, 1998)*, 1998. To appear.

[AHK87] S. B. Akers, D. Harel, and B. Krishnamurthy. The star graph: An attractive alternative to the n-cube. In Sartaj K. Sahni, editor, *Proceedings of the 1987 International Conference on Parallel Processing*, pages 393–400, University Park-London, 1987. Pennsylvania State University Press.

[AHU76] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1976.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, New Jersey, 1993.

[AR95] Yonatan Aumann and Yuval Rabani. Improved bounds for all optical routing. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms SODA '95*, pages 567–576, 1995.

[ATM95] The ATM Forum, Upper Saddle River, NJ. *ATM User-Network Interface (UNI) Specification Version 3.1.*, 1995.

[BBG+97] Bruno Beauquier, Jean-Claude Bermond, Luisa Gargano, Pavol Hell, Stéphane Perennes, and Ugo Vaccaro. Graph problems arising from wavelength-routing in all-optical networks. In *Proceedings of IPPS '97, Second Workshop on Optics and Computer Science (WOCS)*, 1997.

[BDO+98] Gian-Luca Bona, Wolfgang E. Denzel, Bert J. Offrein, Roland Germann, Huub W. M. Salemink, and Folkert Horst. Wavelength division multiplexed add/drop ring technology in corporate backbone networks. Technical Report RZ3046, IBM Research Division, August 1998.

[BDW86] Jacek Blazewicz, Mieczyslaw Drabowski, and Jan Weglarz. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Transactions on Computers*, c-35(5):389–393, May 1986.

[Ber76] Claude Berge. *Graphs and Hypergraphs.* North-Holland, Amsterdam, second edition, 1976.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[BGP+96] J.-C. Bermond, L. Gargano, S. Perennes, A. A. Rescigno, and U. Vaccaro. Efficient collective communication in optical networks. In *Proceedings of the 23rd International Colloquium* *on Automata, Languages and Programming ICALP '96*, LNCS 1099, pages 574–585. Springer-Verlag, 1996.

[BH94] Richard A. Barry and Pierre A. Humblet. On the number of wavelengths and switches in all-optical networks. *IEEE Transactions on Communications*, 42(2/3/4):583–591, 1994.

[BL97] Yair Bartal and Stefano Leonardi. On-line routing in all-optical networks. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming ICALP '97*, LNCS 1256, pages 516–526. Springer-Verlag, 1997.

[BM96] Dhritiman Banerjee and Biswanath Mukherjee. A practical approach for routing and wavelength assignment in large wavelength-routed optical networks. *IEEE Journal on Selected Areas in Communications*, 14(5):903–908, June 1996.

[Bod93] Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11, 1993.

[Bra90] Charles A. Bracket. Dense wavelength division multiplexing networks: Principles and applications. *IEEE Journal on Selected Areas in Communications*, 8(6):948–964, August 1990.

[CFN77] Gerard Cornuejols, Marshall L. Fisher, and George L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science*, 23(8):789–810, April 1977.

[CGJ97] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, pages 46–93. PWS Publishing Company, Boston, MA, 1997.

[CGJL85] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and A. S. Lapaugh. Scheduling file transfers. *SIAM J. Comput.*, 14(3):744–780, August 1985.

[CGK92] Imrich Chlamtac, Aura Ganz, and Gadi Karmi. Lightpath communications: An approach to high bandwidth optical WAN's. *IEEE Transactions on Communications*, 40(7):1171–1182, July 1992.

[CH82] R. Cole and J. Hopcroft. On edge coloring bipartite graphs. *SIAM J. Comput.*, 11(3):540–546, August 1982.

[CHK+96] R. L. Cruz, G. R. Hill, A. L. Kellner, R. Ramaswami, G. H. Sasaki, and Y. Yamabayashi, editors. Special issue on *Optical Networks*, volume 14, number 5 of *IEEE Journal on Selected*

Areas in Communications. IEEE Communications Society, June 1996.

[CKP97] Ioannis Caragiannis, Christos Kaklamanis, and Pino Persiano. Bounds on optical bandwidth allocation on directed fiber tree topologies. In *Proceedings of IPPS '97, Second Workshop on Optics and Computer Science (WOCS)*, 1997.

[CKST95] P. Crescenzi, V. Kann, R. Silvestri, and L. Trevisan. Structure in approximation classes. In Ding-Zhu Du and Ming Li, editors, *Proceedings of the 1st Annual International Conference on Computing and Combinatorics COCOON'95*, LNCS 959, pages 539–548. Springer-Verlag, 1995.

[Cla97] Martin P. Clark. *ATM Networks: Principles and Use*. Wiley-Teubner, Chichester, 1997.

[CNW90] N. K. Cheung, K. Nosu, and G. Winzer, editors. Special issue on *Dense Wavelength Division Multiplexing Techniques for High Capacity and Multiple Access Communication Systems*, volume 8, number 6 of *IEEE Journal on Selected Areas in Communications*. IEEE Communications Society, August 1990.

[DL87] J. Doenhardt and T. Lengauer. Algorithmic aspects of one-dimensional layout compaction. *IEEE Transactions on Computer-Aided Design*, CAD-6(5):863–878, September 1987.

[EIS76] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, December 1976.

[EJ97a] Thomas Erlebach and Klaus Jansen. Call scheduling in trees, rings and meshes. In *Proceedings of the 30th Hawaii International Conference on System Sciences HICSS-30*, volume 1, pages 221–222. IEEE Computer Society Press, 1997.

[EJ97b] Thomas Erlebach and Klaus Jansen. Off-line and on-line call-scheduling in stars and trees. In *Proceedings of the 23rd International Workshop on Graph-Theoretic Concepts in Computer Science WG '97*, LNCS 1335, pages 199–213. Springer-Verlag, 1997.

[EJ97c] Thomas Erlebach and Klaus Jansen. Scheduling of virtual connections in fast networks. In *Proceedings of the 4th Parallel Systems and Algorithms Workshop PASA '96*, pages 13–32. World Scientific Publishing, 1997.

[EJ98a] Thomas Erlebach and Klaus Jansen. Efficient implementation of an optimal greedy algorithm for wavelength assignment in directed tree networks. In Kurt Mehlhorn, editor, *Proceedings* *of the 2nd Workshop on Algorithm Engineering WAE'98*, Technical Report MPI-I-98-1-019, pages 13–24, Max-Planck-Institut für Informatik, Saarbrücken, August 1998.

[EJ98b] Thomas Erlebach and Klaus Jansen. Maximizing the number of connections in optical tree networks. In Kyung-Yong Chwa and Oscar H. Ibarra, editors, *Proceedings of the 9th Annual International Symposium on Algorithms and Computation ISAAC'98*, LNCS 1533, pages 179–188. Springer-Verlag, 1998.

[EJKP98] Thomas Erlebach, Klaus Jansen, Christos Kaklamanis, and Pino Persiano. An optimal greedy algorithm for wavelength allocation in directed tree networks. In *Proceedings of the DIMACS Workshop on Network Design: Connectivity and Facilities Location*, volume 40 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 117–129. AMS, 1998.

[Fel95] Anja Feldmann. On-line call admission for high-speed networks (Ph.D. Thesis). Technical Report CMU-CS-95-201, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, October 1995.

[FMS+95] Anja Feldmann, Bruce Maggs, Jiri Sgall, Daniel D. Sleator, and Andrew Tomkins. Competitive analysis of call admission algorithms that allow delay. Technical Report CMU-CS-95-102, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, January 1995.

[FNS+92] András Frank, Takao Nishizeki, Nobuji Saito, Hitoshi Suzuki, and Éva Tardos. Algorithms for routing around a rectangle. *Discrete Applied Mathematics*, 40:363–378, 1992.

[Fra85] András Frank. Edge-disjoint paths in planar graphs. *J. Comb. Theory Series B*, 39(2):164–178, October 1985.

[Fra90] András Frank. Packing paths, circuits, and cuts – a survey. In Bernhard Korte, László Lovász, Hans Jürgen Prömel, and Alexander Schrijver, editors, *Paths, Flows, and VLSI-Layout*, pages 47–100. Springer-Verlag, Berlin, 1990.

[FW77] S. Fiorini and R. J. Wilson. *Edge-Colourings of Graphs.* Research Notes in Mathematics 16. Pitman, London, 1977.

[FW98] Amos Fiat and Gerhard J. Woeginger, editors. *Online Algorithms: The State of the Art.* LNCS 1442. Springer-Verlag, Berlin, 1998.

[Gar98] Luisa Gargano. Limited wavelength conversion in all-optical tree networks. In *Proceedings of the 25th International Colloquium* on *Automata, Languages and Programming ICALP '98*, LNCS 1443, pages 544–555. Springer-Verlag, 1998.

[Gav72] Fănică Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.*, 1:180–187, 1972.

[GG75] M. R. Garey and R. L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM J. Comput.*, 4(2):187–200, June 1975.

[GHP97] Luisa Gargano, Pavol Hell, and Stephane Perennes. Colouring paths in directed symmetric trees with applications to WDM routing. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming ICALP '97*, LNCS 1256, pages 505–515. Springer-Verlag, 1997.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of $\mathcal{NP}$-Completeness.* W. H. Freeman and Company, New York-San Francisco, 1979.

[GJ85a] Martin Charles Golumbic and Robert E. Jamison. Edge and vertex intersection of paths in a tree. *Discrete Math.*, 55:151–159, 1985.

[GJ85b] Martin Charles Golumbic and Robert E. Jamison. The edge intersection graphs of paths in a tree. *J. Comb. Theory Series B*, 38(1):8–22, February 1985.

[GJMP80] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM J. Algebraic Discrete Methods*, 1(2):216–227, 1980.

[GLS88] Marin Grötschel, Lászlό Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, Berlin, 1988.

[Gol80] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.

[Gol84a] M. Goldberg. An approximate algorithm for the edge-coloring problem. *Congressus Numerantium*, 43:317–319, 1984.

[Gol84b] M. Goldberg. Edge-coloring of multigraphs: Recoloring technique. *Journal of Graph Theory*, 8:123–137, 1984.

[Gra69] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17(2):416–429, March 1969.

[Gra76] R. L. Graham. Bounds on the performance of scheduling algorithms. In Edward G. Coffman, Jr., editor, *Computer and Job-Shop Scheduling Theory*, pages 165–227. John Wiley & Sons, Inc., New York, 1976.

[Gre91] Paul E. Green. The Future of Fiber-Optic Computer Networks. *IEEE Computer*, 24(9):78–87, September 1991.

[Gre93] Paul E. Green. *Fiber Optic Networks.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[GSKR97] Ornan Gerstel, Galen Sasaki, Shay Kutten, and Rajiv Ramaswami. Dynamic wavelength allocation in optical networks. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing PODC'97*, page 293, 1997.

[GVY93] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees, with applications to matching and set cover. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming, ICALP '93*, LNCS 700, pages 64–75, 1993.

[GW97] Michael X. Goemans and David P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 144–191. PWS Publishing Company, Boston, MA, 1997.

[HK73] John Hopcroft and Richard Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, December 1973.

[HLC97] Eric J. Harder, Sang-Kyu Lee, and Hyeong-Ah Choi. On wavelength assignment in WDM optical networks. In *Proceedings of the 4th International Conference on Massively Parallel Processing Using Optical Interconnections MPPOI'97.* IEEE, 1997.

[Hoc97] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, MA, 1997.

[Hol81] I. Holyer. The NP-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720, November 1981.

[HvdVV94] J. A. Hoogeveen, S. L. van de Velde, and B. Veltman. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Applied Mathematics*, 55:259–272, 1994.

[IMI$^+$95] A. Iwata, N. Mori, C. Ikeda, H. Suzuki, and M. Ott. ATM connection and traffic management schemes for multimedia internetworking. *Communications of the ACM*, 38(2):72–89, February 1995.

[Jan97] Klaus Jansen. Approximation Results for Wavelength Routing in Directed Trees. In *Proceedings of IPPS '97, Second Workshop on Optics and Computer Science (WOCS)*, 1997.

[JDU+74] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3:299–326, 1974.

[Kan91] Viggo Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete. *Information Processing Letters*, 37:27–35, 1991.

[Kie98] Hal A. Kierstead. Coloring graphs on-line. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms: The State of the Art*, LNCS 1442. Springer-Verlag, Berlin, 1998.

[KK99] Jon Kleinberg and Amit Kumar. Wavelength conversion in optical networks. In *Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms SODA '99*, pages 566–575, 1999.

[Kla98] Ralf Klasing. Methods and problems of wavelength-routing in all-optical networks. Technical Report CS-RR-348, Department of Computer Science, University of Warwick, September 1998. Presented as invited talk at the MFCS'98 Workshop on Communications.

[KLPS90] Bernhard Korte, László Lovász, Hans Jürgen Prömel, and Alexander Schrijver, editors. *Paths, Flows, and VLSI-Layout.* Springer-Verlag, Berlin, 1990.

[Kön31] D. König. Graphen und Matrizen. *Mat. Fiz. Lapok*, 38:116–119, 1931.

[KP96] Christos Kaklamanis and Pino Persiano. Efficient wavelength routing on directed fiber trees. In *Proceedings of the 4th Annual European Symposium on Algorithms ESA '96*, LNCS 1136, pages 460–470. Springer-Verlag, 1996.

[KPEJ97] Christos Kaklamanis, Pino Persiano, Thomas Erlebach, and Klaus Jansen. Constrained bipartite edge coloring with applications to wavelength routing. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming ICALP '97*, LNCS 1256, pages 493–504. Springer-Verlag, 1997.

[KPRS97] S. Ravi Kumar, Rina Panigrahy, Alexander Russel, and Ravi Sundaram. A note on optical routing on trees. *Inf. Process. Lett.*, 62:295–300, 1997.

[KS97] Vijay Kumar and Eric J. Schwabe. Improved access to optical bandwidth in trees. In *Proceedings of the 8th Annual ACM–*

SIAM Symposium on Discrete Algorithms SODA '97, pages 437–444, 1997.

[KT81] Henry A. Kierstead and William T. Trotter, Jr. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143–153, 1981.

[KT95a] Jon Kleinberg and Éva Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing STOC '95*, pages 26–35, 1995.

[KT95b] Jon Kleinberg and Éva Tardos. Disjoint paths in densely embedded graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science FOCS'95*, pages 52–61, 1995.

[Kum98] Vijay Kumar. Approximating circular arc coloring and bandwidth allocation in all-optical ring networks. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization APPROX'98*, LNCS 1444, pages 147–158. Springer-Verlag, 1998.

[KvL84] M. E. Kramer and J. van Leeuwen. The complexity of wire routing and finding the minimum area layouts for arbitrary VLSI circuits. In Franco P. Preparata, editor, *Advances in Computing Research; VLSI Theory*, volume 2, pages 129–146. JAI Press Inc., Greenwich, CT-London, 1984.

[KW95] B. G. Kim and P. Wang. ATM network: Goals and challenges. *Communications of the ACM*, 38(2):39–44, February 1995.

[Lei92] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers Inc., San Mateo, CA, 1992.

[Len81] H. W. Lenstra. Integer programming with a fixed number of variables. Technical Report 81-03, Mathematics Dept., Univesity of Amsterdam, 1981.

[Leo98] Stefano Leonardi. On-line network routing. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms: The State of the Art*, LNCS 1442. Springer-Verlag, Berlin, 1998.

[LV98] Stefano Leonardi and Andrea Vitaletti. Randomized lower bounds for online path coloring. In *Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science*, LNCS 1518, pages 232–247. Springer-Verlag, 1998.

[McA91] Alastair D. McAulay. *Optical Computer Architectures.* Wiley, New York, 1991.

[MHIR96] M. V. Marathe, H. B. Hunt III, and S. S. Ravi. Efficient approximation algorithms for domatic partition and on-line coloring of circular arc graphs. *Discrete Applied Mathematics*, 64:135–149, 1996.

[MKR95] Milena Mihail, Christos Kaklamanis, and Satish Rao. Efficient access to optical bandwidth. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science FOCS'95*, pages 548–557, 1995.

[MN95] Kurt Mehlhorn and Stefan Näher. LEDA: A platform for combinatorial and geometric computing. *Communications of the ACM*, 38, 1995.

[MPS98] Ernst W. Mayr, Hans Jürgen Prömel, and Angelika Steger, editors. *Lectures on Proof Verification and Approximation Algorithms*. LNCS 1367. Springer-Verlag, Berlin, 1998.

[Muk92a] Biswanath Mukherjee. WDM-based local lightwave networks part I: Single-hop systems. *IEEE Network*, 6(3):12–27, May 1992.

[Muk92b] Biswanath Mukherjee. WDM-based local lightwave networks part II: Multihop systems. *IEEE Network*, 6(4):20–32, July 1992.

[Muk97] Biswanath Mukherjee. *Optical Communication Networks*. McGraw-Hill, 1997.

[MV80] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science FOCS'80*, pages 17–27, 1980.

[NK90] Takao Nishizeki and Kenichi Kashiwagi. On the 1.1 edge-coloring of multigraphs. *SIAM J. Disc. Math.*, 3(3):391–410, August 1990.

[NPZ97] Christos Nomikos, Aris Pagourtzis, and Stathis Zachos. Efficient coloring with applications in multiwavelength routing. Presented at the *ICALP'97 Workshop on Algorithmic Aspects of Communication (July 11–12, 1997, Bologna, Italy)*, 1997.

[NZ97] Christos Nomikos and Stathis Zachos. Coloring a maximum number of paths in a graph. Presented at the *ICALP'97 Workshop on Algorithmic Aspects of Communication (July 11–12, 1997, Bologna, Italy)*, 1997.

[OBB81] James B. Orlin, Maurizio A. Bonuccelli, and Daniel P. Bovet. An $O(n^2)$ algorithm for coloring proper circular arc graphs. *SIAM J. Algebraic Discrete Methods*, 2(2):88–93, 1981.

[OS81] Haruko Okamura and P. D. Seymour. Multicommodity flows in planar graphs. *J. Comb. Theory Series B*, 31(1):75–81, August 1981.

[Pan92] R. K. Pankaj. *Architectures for linear light-wave networks*. PhD thesis, MIT, 1992.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.

[PY91] C. Papadimitriou and M. Yannakakis. Optimization, approximization and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[Rab96] Yuval Rabani. Path coloring on the mesh. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science FOCS '96*, pages 400–409, 1996.

[Ram93] Rajiv Ramaswami. Multiwavelength lightwave networks for computer communication. *IEEE Communications Magazine*, 31(2):78–88, February 1993.

[RS92] M. Niel Ransom and Dan R. Spears. Applications of public gigabit networks. *IEEE Network*, 6(2):30–40, March 1992.

[RS95] Rajiv Ramaswami and Kumar N. Sivarajan. Routing and wavelength assignment in all-optical networks. *IEEE/ACM Transactions on Networking*, 3(5):489–500, October 1995.

[RT87] P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[RU94] Prabhakar Raghavan and Eli Upfal. Efficient routing in all-optical networks. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing STOC '94*, pages 134–143, 1994.

[Sch98] Alexander Schrijver. Bipartite edge-coloring in $O(\Delta m)$ time. *SIAM J. Comput.*, 28(3):841–846, 1998.

[SDRF95] James A. Schnepf, David H. C. Du, E. Russell Ritenour, and Aaron J. Fahrmann. Building future medical education environments over ATM networks. *Communications of the ACM*, 38(2):54–69, February 1995.

[SH90] Wei-Kuan Shih and Wen-Lian Hsu. An approximation algorithm for coloring circular-arc graphs. In *SIAM Conference on Discrete Mathematics*, 1990.

[Sha49] C. E. Shannon. A theorem on coloring lines of a network. *J. Math. Phys.*, 28:148–151, 1949.

[SSW98] Alexander Schrijver, Paul Seymour, and Peter Winkler. The ring loading problem. *SIAM J. Disc. Math.*, 11(1):1–14, February 1998.

[ST85] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, February 1985.

[Str97] Bjarne Stroustrup. *The C++ Programming Language: Third Edition*. Addison-Wesley Publishing Co., Reading, Mass., 1997.

[SWW91] David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science FOCS '91*, pages 131–140, 1991.

[Tar85] Robert E. Tarjan. Decomposition by clique separators. *Discrete Math.*, 55:221–232, 1985.

[Tuc75] Alan Tucker. Coloring a family of circular arcs. *SIAM J. Appl. Math.*, 29(3):493–502, November 1975.

[VD93] Ronald J. Vetter and David H. C. Du. Distributed computing with high-speed optical networks. *IEEE Computer*, 26(2):8–18, February 1993.

[Vet95] Ronald J. Vetter. ATM concepts, architectures, and protocols. *Communications of the ACM*, 38(2):30–38, February 1995.

[vL90] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science. Volume A: Algorithms and Complexity*. Elsevier North-Holland, Amsterdam, 1990.

[Wil96] Gordon Wilfong. Minimizing wavelengths in an all-optical ring network. In *Proceedings of the 7th Annual International Symposium on Algorithms and Computation ISAAC'96*, LNCS 1178, pages 346–355. Springer-Verlag, 1996.

[WW98] Gordon Wilfong and Peter Winkler. Ring routing and wavelength translation. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms SODA'98*, pages 333–341, 1998.

[YG87] Mihalis Yannakakis and Fănică Gavril. The maximum k-colorable subgraph problem for chordal graphs. *Inf. Process. Lett.*, 24(2):133–137, January 1987.

Index