Dynamical Configuration Of Transparent Optical Telecommunication Networks

Diplomarbeit
bei Prof. Dr. Martin Grötschel

vorgelegt von Andreas Tuchscherer
am Fachbereich Mathematik der
Technischen Universität Berlin

Berlin, den 31. März 2003

Acknowledgements

The research presented in this thesis is motivated by a joint project of T-Systems Nova GmbH Technologiezentrum and the Konrad-Zuse-Zentrum für Informationstechnik Berlin, financed by Deutsches Forschungsnetz e.V.

Working on this project was very instructive for me. The cooperation with Jörg Rambau, Diana Poensgen and Sven O. Krumke was very pleasant. I wish to thank them for their support as well as Monika Jäger and Ralf Hülsermann from T-Systems Nova, who informed us about the technical background.

Relating to this thesis I especially thank Jörg Rambau, Diana Poensgen and Sven O. Krumke for reading the preliminary versions, their professional advice, their helpful suggestions and their encouragement.

Berlin, den 31.März 2003

Andreas Tuchscherer

Contents

1 Introduction **1**

1.1 Technology of Optical Telecommunication Networks 1

1.1.1 Optical Fibers 2

1.1.2 Optical Switches 4

1.1.3 Wavelength Conversion 6

1.2 Management of Optical Telecommunication Networks 7

1.2.1 Basic Planning Decisions 8

1.2.2 Optimization Problems 9

1.2.3 Dynamic Call Admission 9

1.2.4 Distributed Algorithms vs. Centralized Algorithms 11

1.3 Related Work 11

1.4 Contribution and Outline 13

2 Modeling of Dynamic Call Admission in Optical Networks **14**

2.1 Online Optimization and Competitive Analysis 14

2.2 Notation 17

2.3 Problem Definition: Dynamic Multiclass Call Admission (**DMCA**) . . 19

2.3.1 Scope of the Model 20

2.3.2 Restriction: Dynamic Singleclass Call Admission (**DSCA**) . . 21

2.3.3 Previous Work on **DSCA** 21

2.3.4 Motivation for the Use of a Simulation Based Evaluation . . . 24

3 Online Algorithms for DSCA **25**

3.1 Greedy-Type Algorithms 26

3.1.1 Partial Wavelength Search 27

3.1.2 Total Wavelength Search 29

3.1.3 Analysis of Greedy-Type Algorithms 30

3.2 Network Fitness Algorithms 33

3.2.1 Introduction 33

3.2.2 The Algorithm ALR 34

3.2.3 The Algorithm SFR 44

3.2.4 The Algorithm NFR 51

3.2.5 Reduction to k Shortest Lightpaths Routings 55

4 An Algorithm for Finding the k Shortest Paths **56**

4.1 Introduction 56

4.2 Preliminaries 57

4.3 The Algorithm 59

4.4 Proof of Correctness 62

4.5 Runnning Time Complexity 66

5 Experimental Results **68**

5.1 Four Real-World Optical Networks 68

5.2 Traffic Model and Request Sequences 70

5.3 Simulation Model 71

5.4 Results 72

5.4.1 Greedy-Type Algorithms 72

5.4.2 Versions of ALR 75

5.4.3 Versions of SFR 80

5.4.4 Best Algorithms Revisited 80

6 Conclusions and Outlook **86**

A Table of Notations **88**

B Basic Definitions **89**

C Deutsche Zusammenfassung **90**

List of Algorithms **92**

Bibliography **96**

Chapter 1

Introduction

In this diploma thesis, we investigate methods for online call admission and routing and wavelength assignment in transparent optical telecommunication networks. We formulate a corresponding online optimization problem, present algorithms for it, and evaluate the performance of the algorithms by extensive simulation studies. The research is based on a joint project with T-Systems Nova GmbH, financed by the DFN.

The outline of this chaper is as follows. In Section 1.1, we present the underlying network technology and real-world applications for optical networks. General planning decisions in optical networks, as well as the online problem under consideration are introduced in Section 1.2. After a brief overview on previous work in Section 1.3, we give a review of the structure of this thesis in Section 1.4.

1.1 Technology of Optical Telecommunication Networks

In telecommunication networks, electronics have been the predominant operating technology for a long time. Network links were provided with copper cables for transmission of electronic signals. By electronic switching devices in network nodes, connections along paths of several links were established. However, electronic networks no longer provide enough capacity for today's high bandwidth applications, such as data browsing on the World Wide Web or video conferencing. Since the installation of copper cables is very expensive and a suitable increase of the transmission bit rate by a speed up of electronics is impossible due to physical restrictions, the development of new technology was required.

A promising approach are *optical networks* because of the immense capabilities which are provided by *optical fibers*. On the one hand, a huge bandwidth of nearly 50 Tb/s (terabits per second) per fiber is possible in theory, whereas only around 10 Gb/s (gigabits per second) could currently be achieved by copper cables. On the other hand, optical fibers feature small space requirement, low material usage, and low cost. Moreover, transmission on fibers affects only low signal attenuation and low distortion, is less susceptible to electromagnetic interference and provides more security because tapping optical signals is difficult.

In the following, we will present the main components of all-optical networks which are besides optical fibers, the *optical switches* and the *optical wavelength converters*. For more details on optical telecommunication networks, we refer to the books [Muk97, RS98, SB99].

1.1.1 Optical Fibers

In optical networks, each link consists of a cable containing several optical fibers, sometimes more than 100. Every fiber is composed of a cylindrical highly transparent core surrounded by a cladding which are both made mainly of silica (SiO_2), the basis of glass. Depending on the type of fiber, the diameter of its core is either about 10 μm or 50 μm, and that of the cladding is usually 125 μm.

Instead of electronic signals, in optical fibers, lightwaves propagate due to a series of total internal reflections that occur at the core-cladding interface. In order to send such signals, the digital information data is converted into short light pulses using a laser device. This takes place in a *transmitter* which is installed on one end node of the fiber. Upon arrival at its other end node, a *receiver* converts this optical signal back into its usable electronic form using photodetectors or photodiodes. Such an optical connection along a fiber is called an *optical channel*. Optical fibers have been used in telecommunication networks for over two decades.

Wavelength Division Multiplexing

The most important techniques that allows for expanding the inherent great capacity of optical fibers even more is *wavelength division multiplexing* (WDM). The idea of WDM is to use several independent lightwaves of different wavelengths on one fiber at the same time, each carrying data at the original bit rate that depends on the fiber quality. However, due to signal attenuation the usable spectral window of wavelengths is restricted to two ranges of about 200 nm each (1200–1400 nm and 1450–1650 nm), where loss is small. In this spectrum, all wavelengths have very similar propagation properties and deviate little when used for transmitting optical signals. Hence, the usable spectrum can be divided into separate wavelength ranges which may be used simultaneously since they do not affect each other. Note that it is not possible to use one wavelength on a fiber more than once since the corresponding optical signals would become useless. In order to use WDM on an optical fiber connecting two network nodes, each optical channel is set up on one wavelength by an appropriate tuned laser at the transmitter. Afterwards, all wavelengths of used channels are combined on the fiber by a *multiplexer*. At the other end of the fiber, a *demultiplexer* again decomposes the lightwaves and transfers them to corresponding receivers. Figure 1.1(a) depicts the described technical view of an optical fiber connecting two nodes, together with the necessary equipment to apply WDM. In contrast, Figure 1.1(b) shows the logical perception of a WDM fiber, which illustrates the different available wavelength channels on the fiber.

This way, the total capacity of a single WDM fiber is the number of available wavelengths times the capacity of one channel on the fiber. Together with the high bit rates

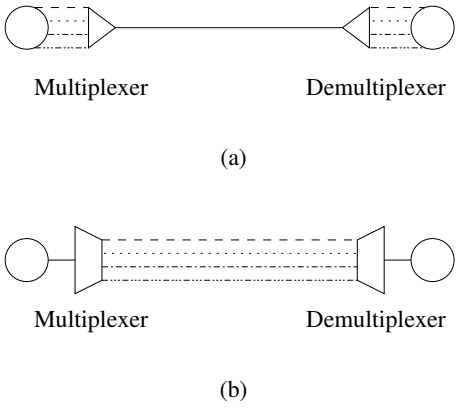

Figure 1.1: A WDM system with four wavelengths. (a) Technical view; (b) Presentation of the channels on the fiber.

of recent fiber qualities, this results in large bandwidths. In the late nineties, WDM systems with up to 32 wavelengths at bit rates of 1–10 Gb/s were commercially available, resulting in a total bandwidth per fiber of several hundred Gb/s. However, research laboratories already demonstrated transmission experiments with 160 wavelengths at 40 Gb/s each. Recall that the total capacity of one network link is accumulated from the capacities of up to 100 fibers.

Typically, optical channels can be established on a fiber in both directions, and one distinguishes between bidirectional and unidirectional WDM systems. A *simplex* fiber (bidirectional WDM system) realizes both directions on a single glass core, and often uses half the wavelengths for transmitting data in one direction and the other half for transmitting data in the opposite direction on the same fiber. In contrast, a *duplex* fiber (unidirectional WDM system) consists of two glass cores, one for each direction of traffic. Therefore, the capacity is totally doubled, but cannot be shared arbitrarily between both directions.

Regeneration

While transmitting optical signals over fibers, the light propagation is disturbed by some undesired effects, such as attenuation, dispersion, material impairments, and others. Even though these effects are small, especially when fibers of high quality are used, they disturb the light propagation, which results in signal loss. In order to counteract this, *amplifiers* are placed on the fibers in a distance of about 100 kilometers, regenerating optical signals. However, these devices are not able to restore the loss completely. In theory, the complete regeneration consists of several steps, but only the amplification of lightwaves can currently be performed on the optical signal directly. Unfortunately, it is not expected that sufficient optical regeneration devices will be available in the near future. Therefore, the lengths of optical channels stay limited, which must be considered in planning optical networks.

First-Generation Optical Networks

Telecommunication networks in which each link consists of one or more optical fibers on which WDM is applied are called *first-generation optical networks*. Note that the network nodes still use totally electronic devices. Hence, in order to establish connections over several fibers, switching and processing of data at intermediate nodes is performed by converting the optical signal back into its electronic form. As a consequence, realizing a connection along several network links requires a lot of so-called *opto-electronic* conversions.

Nowadays, most networks employ such electronic processing at nodes and use optical fibers as transmission medium. However, the speed of electronics is unable to match the high bandwidth of optical fibers. Therefore, network nodes must be equipped with a lot of expensive electronic devices, in order to provide sufficient switching capabilities. Moreover, conversions introduce significant delays. These drawbacks motivated the development of optical devices which inherit some of the switching and routing functions that were previously performed by electronics into the optical part of the network, thus avoiding opto-electronic conversion. Such devices are called *optical switches*. Since these network components have only been commercially available for a short time and are still very expensive, they are currently rarely used.

1.1.2 Optical Switches

In today's telecommunication networks, data connections need to be established and closed within milliseconds, which requires software-controlled reconfigurable switching equipment on network nodes. There are mainly two different all-optical switching devices: *optical cross connects* (OXCs) and *optical add drop multiplexers* (OADMs). Both switches provide a specific number of *input ports* and *output ports* which are linked with several optical fibers and can be connected in different ways. The established assignment of input ports to output ports is called the *configuration* of the switch.

An OXC is able to connect its input and output ports arbitrarily. The processing of optical signals at such a switch is handled as follows. After demultiplexing the different wavelengths that arrive on a fiber, each optical signal enters its own input port. Depending on the configuration of the OXC, each lightwave is guided to some output port without changing its wavelength. To this end, the switch uses for example fields of tiny mirrors whose adjustment determines the configuration. Leaving the output port, a lightwave is again multiplexed at the linked optical fiber and sent on.

In contrast to OXCs, which allow for guiding each optical channel individually, an OADM only provides restricted switching capability. The idea that restricted switching might suffice is due to the observation that a lot of channels usually follow similar paths in the network. Hence, these channels can be considered as a bundled data stream from which only a few channels are dropped or added at each network node. The main part of the stream stays together.

Second-Generation Optical Networks

The integration of optical switches completes the optical layer of the network since optical channels can now be transmitted in optical form over whole paths from source to destination and not only along single fibers. That is, no opto-electronic conversions are performed anymore since signals are directly processed on the optical channels. Optical networks which are equipped with optical switches in addition to optical fibers are called *second-generation optical networks* or simply *all-optical networks*. Due to the character of signal switching, the transmission of data in all-optical network is referred to as *transparent routing*, whereas one speaks of *opaque routing* in first-generation optical networks.

Further advantages of all-optical networks result from the substantial reduction of interfaces between optics and electronics: Unnecessary manipulations, bottlenecks, and costs are avoided. Moreover, this yields a better combination of both transmission and switching media and their main advantages (recall that the control of the switches is of course still handled by electronics).

For both, first-generation and second-generation optical networks, the graph or digraph, respectively, that represents the connections of network links and nodes is called the *physical topology* of the network. Naturally, a simplex fiber is represented by an edge, and a duplex fiber is represented by two opposite directed arcs. Notice that the physical topology may have parallel edges or arcs, respectively, if there are several fibers contained in one network link. Figure 1.2 shows the physical topology of a simple all-optical network with one simplex fiber on each link.

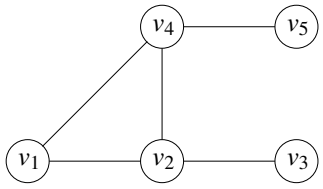

Figure 1.2: Physical topology of an optical network.

Lightpaths

By using optical switching devices in a network, an optical signal may be transmitted along several fibers without leaving the optical layer, i.e., no opto-electronic conversion is performed at intermediate nodes. The resulting optical channel from its transmitter to its receiver is called a *lightpath*. The concept of lightpaths is the main characteristic of all-optical networks.

Since optical switches maintain the wavelengths of processed lightwaves, each lightpath operates on exactly one wavelength. This wavelength is used on all optical fibers the signal traverses. Assume that in our exemplary all-optical network whose physical topology is depicted in Figure 1.2, each fiber is equipped with an identic WDM system that provides two wavelengths. Figure 1.3(a) shows four realized lightpaths in this network. The different line styles indicate different wavelengths.

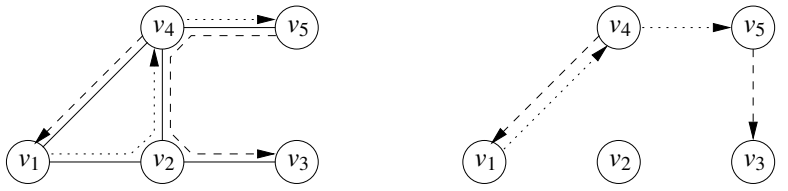

Figure 1.3: (a) Lightpaths in an optical network; (b) The virtual topology that results from the established lightpaths.

Note that the end nodes of a lightpath are its only interface to the electronic layer of an all-optical network. Hence, it is particularly interesting between which network nodes lightpath connections are established. This information is represented by the *virtual topology*, also called *logical topology*. The virtual topology is a digraph that contains for each realized lightpath one arc from its source to its destination node. The virtual topology for the set of lightpaths in Figure 1.3(a) is shown in Figure 1.3(b), the different line styles again reflect the different wavelengths which are used for the connections.

Even though lightpaths are directed optical signals, nearly in all telecommunication applications, data is transmitted bidirectionally. Therefore, each unit demand requires two lightpaths that connect the specified network nodes in opposite directions. In doing so, it is suitable and simple to realize both lightpath along the same network links. Moreover, such proceeding is even more simplified by providing the network links with duplex fibers. Then, both opposite directed lightpaths may also operate on the same wavelength.

1.1.3 Wavelength Conversion

As mentioned above, each lightpath must use the same wavelength on all fibers it uses. Due to this condition, an efficient capacity utilization is sometimes impossible: Instead of using all wavelengths on a fiber, the installation of additional fibers might be necessary to satisfy some given demands. For instance, it is impossible to establish an additional lightpath from node v_4 to node v_2 in the situation depicted in Figure 1.3(a) since the first wavelength is already being used on the fiber connecting v_4 and v_1, the second wavelength on the fiber between v_1 and v_2. Furthermore, additional wavelengths are not available. However, if it were possible to realize a lightpath using different wavelengths on the two fibers, a connection from v_4 to v_2 could still be established.

This disadvantage can be overcome by using *optical wavelength converters*. A wavelength converter is another optional component of all-optical networks that is installed on network nodes and is able to change the wavelength of passing lightpaths during the switching process. Wavelength converters can improve the capacity efficiency in the network by resolving wavelength conflicts of lightpaths. Notice that a

single lightpath in an all-optical network which provides wavelength conversion capabilities can use different wavelengths on the fibers in its path. Although optical wavelength converters are not commercially available yet and the features they will have are not yet completely clear, the following properties are of interest.

The most important characteristic of wavelength converters is the possible *conversion range*. As the name suggests, *full* wavelength conversion is able to transform a given wavelength into any other wavelength. Using an optical converter that provides *limited* conversion, each wavelength is assigned a subset of wavelengths into which it may be changed. Finally, with *fixed* conversion, a signal entering the node on one wavelength must always leave it at one specific wavelength that depends on the input, i.e., fixed conversion corresponds to a limited conversion where each set of output wavelengths contains exactly one element.

A second interesting aspect is the total number of wavelengths that can be transformed by one converter at the same time. Although most approaches aspire to perform only a single wavelength conversion, one techniques is being developed that may allow for several conversions simultaneously. It is expected that optical wavelengths converters will provide full wavelength conversion for single optical channels in the near future.

1.2 Management of Optical Telecommunication Networks

From the mathematical point of view, first-generation optical networks can be modeled in the same way as the former fully electronic networks, but with different link capacities and costs, since optical fibers merely serve as replacement for copper cables. The only difference is that the bandwidth of any connection results from an integer number of optical channels, since each wavelength on a fiber is used as a whole. Hence, each demand is specified as the number of required channels, i.e., an integer value. Nevertheless, there is no need to develop new optimization models for first-generation optical networks because even in electronic networks the cost structures on links require integral units.

As for electronic networks, the physical topology is modeled as a graph (or digraph) with capacitated edges (or arcs, respectively). Note that for first-generation optical networks these edge capacities are integer. Each connection is represented as a path in the graph and needs a specific amount of bandwidth. That is, the essential restriction requires that for each edge, the total bandwidth demand of all established connections whose paths contain this edge is bounded from above by the corresponding edge capacity.

However, such modeling is insufficient for all-optical networks. Based on the concept of a lightpath, the different available wavelengths must be distinguished unless arbitrary wavelength conversion is possible in the network nodes. Even if such converters will eventually be available, they are expected to be expensive and will most likely not be installed in all network nodes. If no converters exist in the network, each lightpath must use the same wavelength on all optical fibers the signal traverses. In the

mathematical model, the condition that reflects this characteristic is called the *wavelength continuity constraint*. Furthermore, another substantial restriction results from the available fiber capacities: Any two lightpaths which share one fiber must not use the same wavelength. In other words, each wavelength can be used on a fiber by only one lightpath (in each direction for duplex fibers). This condition is called the *wavelength conflict constraint*. As a consequence, the assignment of wavelengths to optical channels becomes an important task, further complicating the design of all-optical networks. The actual modeling of these networks is presented in the next chapter.

In this thesis, we are concerned with all-optical networks without any wavelength conversion capabilities. From now on, we will refer to these networks simply as *optical networks*. Transferred from electronic networks, a lot of different problems immediately emerge in the design and control of optical networks. Before we turn to the problem considered in this work, let us look at some general planning decisions.

1.2.1 Basic Planning Decisions

In order to satisfy a set of traffic demands each of which specifies two network nodes to be connected by some number of lightpaths, the first planning decision concerned with optical networks is obviously the construction of the network itself. That is, its *dimensioning* has to be defined. For this task, we are given a network topology consisting of network nodes and links, and have to specify which optical switches and fibers (and other devices) are installed on the nodes and links, respectively. We aim at providing enough capacities to meet the requirements. Each specification of those optical components to be installed yields a dimensioning of the network. For each link, the number of fibers as well as their kinds, qualities, and used WDM systems has to be chosen. Moreover, the types of optical switches must be specified. In constructing the dimensioning of the network, already existing devices have to be taken into account. In this case, we deal with the *redimensioning* problem. Along with the dimensioning of an optical network, its physical topology and the corresponding network capacities for lightpaths are defined.

Once a dimensioning has been fixed, the subsequent task is the assignment of lightpaths to the given demands. This second problem is called the *virtual topology design* and includes all decisions that are made to configure a virtual topology on a given physical topology. For each demand, the remaining problem of realizing corresponding lightpaths consists of assigning a path and a wavelength. Therefore, this part is also called *routing and wavelength assignment*. Sometimes, wavelength converters are available to be placed on the network nodes. There are many different strategies for both problems described above. For example, routing and wavelength assignment can be performed consecutively, or jointly. That is, wavelengths are assigned after the paths for the connections have been chosen, or lightpaths are determined in one step. Moreover, the delimitation between dimensioning and virtual topology design is not consistently defined. For instance, the specification where wavelength converters are placed can also be viewed as a part of the dimensioning.

1.2.2 Optimization Problems

One basic problem emerges from the combination of the two subproblems of network dimensioning and virtual topology design. Given a set of traffic demands, the task is to determine a network dimensioning and the design of a virtual topology such that all given demands are satisfied and the total costs for electronic devices, optical fibers, optical switches, and wavelength converters is minimized. Many special variants of this basic problem can be considered. For instance, one is given an existing dimensioning that has to be extended (redimensioning), or the demands include backup connections that have to be provided for the case of node or link failures.

In real-world applications, however, traffic demands of course change over time. This leads to the *network reconfiguration* problem. Given are traffic demands which differ from previous ones that were satisfied earlier. The task is to meet the new requirements by reconfiguring the network at a minimum cost. Since it is cheaper and faster to change the virtual topology design, this adaption is particularly done in the optical layer of the network, and it is desirable to find a solution which yields as few lightpath changes as possible. However, it may also be necessary to extend the physical topology.

In addition to such optimization problems that aim at minimizing the cost of equipment and operation, some problems aiming at configuration quality are of interest, too.

First, given an optical network with a fixed dimensioning and a lot of traffic demands which could possibly not be satisfied altogether, one would like to maximize the *throughput*, i.e., the total number of satisfied demands, or the total profit made by them, respectively.

In a second problem, we are given a network whose dimensioning always provides enough capacity to meet the given requirements. The task is to accept all demands in such a way that the maximum congestion on a link is minimized, where the congestion of a link is defined as the number of lightpaths using it. This problem is also referred to as *load balancing*. In doing so, the resulting load of lightpaths is uniformly divided in the network.

A similar task is to minimize the total number of wavelengths needed in order to satisfy the given demands. Here, it is possible to use homogeneous WDM systems on the fibers which provide only few wavelengths.

1.2.3 Dynamic Call Admission

In this thesis, we consider a completely different problem related to optical networks. In all problems above, traffic demands are considered as being static. That is, the corresponding connections are established permanently. In many real-world applications like telephone networks, however, demands change in a highly dynamic way since customers do not require the offered services all the time. Hence, connections are used only for short durations, e.g., hours. As a consequence, new lightpath connections must be established and already existing ones must be closed in a continuous process,

resulting in a *dynamic virtual topology*. The concept that connections are set up and taken down upon demand is called *circuit switching*. Even though OXCs and OADMs allow to establish lightpaths very fast, circuit switching is not yet performed in optical networks, but is expected for the near future.

Another aspect of dynamic traffic is that the demands are naturally not known in advance. Each of them gets known only very shortly before it is actually needed. We refer to these special demands which require lightpath connections for a short time and are not known in advance as *connection requests* or *calls*. Obviously, the type of planning problems that result from previously unknown demands is totally different compared to the problems above. In mathematical terms, these two different classes of optimization problems are called *offline problems* and *online problems*. While the input data of each instance of an offline problem is given completely in advance, this not the case for the instances of online problems. A formal definition of online problems is presented in the next chapter.

The above mentioned features lead to the online problem which is considered in this work. We are given an optical network with fixed dimensioning that does not provide wavelength converters. Calls arrive over time, each of which specifies basically two network nodes to be connected by some constant number of lightpaths for a given time period. As usual in most applications, the specified number of lightpaths has to be routed in both directions between the end nodes. We assume that the optical network is equipped exclusively with duplex fibers and require that for each established lightpath the corresponding opposite directed lightpath which has the same wavelength is also realized. For each newly arriving connection request, the network operator has to decide whether he accepts or rejects the call. This first decision is referred to as *call admission*. Moreover, for each accepted connection request, corresponding lightpaths has to be provided. In the standard operating state, these lightpaths must be fixed for the total holding time of the connection, i.e., the network operator must not suspend any established connection before its expiration time nor exchange the used lightpaths. As mentioned before, the problem of selecting lightpaths for an accepted connection request is called *routing and wavelength assignment*. Each accepted call yields a specific profit. The task of the problem is to maximize the total profit gained by accepted connection requests.

Depending on the provided capacities of the optical network, the customers may suffer service blocking, especially in situations of demand peaks. That is, unless the dimensioning of the network is immoderate, some connection requests must be rejected. From this observation results the need to develop reasonable algorithms for the described problem. The difficulty in the design of algorithms emerges from the uncertainty of future calls.

In real-world applications, the following more detailed parameters of connection requests are imaginable: the network nodes to be connected, the number of required lightpaths between the nodes in each direction, the time at which the call arrives, the time at which the decision must be made whether the call is accepted or not, the desired start time of the connection, the expiration time of the connection, a customer class, a required service class, and the profit gained by accepting the call. Using different customer classes allows to distinguish between more or less important customers.

In doing so, special constraints can be reflected, e.g., calls of important customers must always be accepted if possible. Moreover, each service class represents further requirements, e.g., rerouting priorities in the case of network failures.

The profit of a connection request usually depends on its other parameters, in particular on the number of required lightpaths, the duration of the connection, the customer, and the service class. Note that for connection requests that yield identic profits, the aim of maximizing the total profit gained is equivalent to minimize the probability that a connection request is rejected. This stochastic value is called *blocking probability*. Due to the consideration of several classes of customers and services, we refer to this special problem of online call admission and routing and wavelength assignment as the *Dynamic Multiclass Call Admission Problem*, briefly called **DMCA** in the following. The special version of **DMCA** with only one customer class and one service class, which is particularly considered in this thesis, is called *Dynamic Singleclass Call Admission Problem* (**DSCA**).

For the technical equipment of the given optical network, we assume the following: Each network node provides a sufficient number of electronic devices, especially transmitters and receivers, such that it is able to be the end node of arbitrarily many lightpaths. Moreover, the nodes contain OXCs that allow for arbitrary switching capabilities. Recall that no wavelength converters are installed on the network nodes. As mentioned above, each network link consists of one or several duplex fibers with some installed WDM systems. As a consequence, the only bottleneck for realizing lightpaths results from the fiber capacities, i.e., the numbers of provided wavelengths.

1.2.4 Distributed Algorithms vs. Centralized Algorithms

For dynamic tasks concerned with optical networks, e.g., the problems **DMCA** and **DSCA**, one distinguishes between *centralized* and *distributed* algorithms. While the first have full and consistent information about the network status and arriving connection request at any time, distributed algorithms consist of a collection of interacting processes each of which only has local informations. In this work, we exclusively consider centralized algorithms.

1.3 Related Work

In previous work for **DSCA**, the following simplifying assumptions are usually made for the problem parameters: Each network link consists only of one optical fiber, and each fiber is equipped with the same set of wavelengths. Furthermore, each call requires only one lightpath and yields the same profit. That is, the goal is to minimize the blocking probability.

Also motivated by the need for distributed implementations, particularly simple algorithms for the considered version of **DSCA** have previously been proposed, e.g., the so-called *fixed routing* and *fixed-alternate routing* algorithms. In fixed routing methods, one connecting path (usually a shortest path) is assigned to each pair of

network nodes. For a corresponding call, only lightpaths along the predefined path are taken into account. In fixed-alternate routing, a fixed set of a few paths is allowed to be chosen in some priority order instead of only a single path. In both variants, a connection request is rejected if it cannot be routed on any of the specified lightpaths. Otherwise, one of the possible wavelengths must be assigned to the connection.

First approaches used fixed and random wavelength search orders (see [CGK92, BK95]). That is, the first possible wavelength in the given order is used. In [BSSB95], Bala, Stern, Simchi, and Bala proposed adaptive search orders that incorporate network status information, namely the current utilizations of different wavelengths (the utilization of a wavelength was defined as the current number of established lightpath connections using it). Using the classification of Mokhtar and Azizoglu in [MA98], the above mentioned algorithms are counted among *constraint routing* schemes because of their restrictive use of lightpaths.

As a counterpart, Mokhtar and Azizoglu propose five *adaptive unconstraint routing* schemes, also called *greedy-type* algorithms, where any lightpath connecting two nodes can potentially be chosen. In particular, these algorithms completely omit the call admission part since connection requests are always accepted if possible. Greedy-type algorithms are based on shortest path computation and different sorting mechanisms of the wavelength set. Experimental results, using Poisson traffic and exponential distributed call holding times, showed the superiority of adaptive unconstraint routing over constraint routing with respect to the blocking probability. Moreover, the authors reveal that taking into account network status information like the wavelength utilization can improve the behaviour of algorithms (different as in [BSSB95], the utilization is here defined as the number of fibers on which the corresponding wavelength is currently used by some connection).

In joint work with Hülsermann, Jäger, Krumke, Poensgen, and Rambau, we proposed adapted versions of the greedy-type algorithms of Mokhtar and Azizoglu, including the specification of tie-breaking rules, see [HJK+03]. In our model, still one fiber is installed per link but the wavelengths provided by the fibers may differ. Using extensive experimental studies that are based on a well-founded traffic model, which is also applied in this thesis, we show that the choice of breaking ties can affect the blocking probability of a greedy-type algorithm significantly.

Another algorithmic approach, which is not followed up in this work, is proposed by Zhang and Acampora ([ZA95]). They consider two algorithms that take into account the current load of a fiber which is defined as the total number of wavelengths currently used on it. The first of their algorithms selects such a path for a required connection that has a minimum load, where the load of a path is defined as the maximum load on its fibers. The second algorithm routes a given call on a shortest possible path and uses the load of paths for breaking ties. Unfortunately, the authors did not specify how to select the wavelength to use for the lightpath.

To the best of our knowledge, no rejection criteria for the call admission part have yet been proposed, except for the restriction to few lightpaths due to constraint routing. That is, all algorithms proposed so far concentrate on the routing and wavelength assignment problem. We consider further related work on **DSCA** in Section 2.3.3 after the mathematical model for the problem has been defined.

1.4 Contribution and Outline

The contribution of this thesis is the following. We propose a mathematical problem formulation for **DMCA** and **DSCA**, and present new centralized algorithms for the latter problem, briefly called **DSCA**-algorithms.

On the one hand, these include adapted versions of the greedy-type algorithms ([MA98, HJK+03]). The purpose of reviewing the greedy-type algorithms here is to look at their ideas in more detail. Moreover, the serve as benchmark for further algorithm.

On the other hand, we propose new **DSCA**-algorithms. Our intention in the review of the greedy-type algorithms is that they are very easy to implement and may serve as a benchmark. The main part of our work, however, is concerned with the new algorithms. In their design, we attached great importance to their practicability, not to their theorectial qualities. In order to evaluate the behaviour of the algorithms, we provide extensive experimental studies that allow us to relate the blocking probability of an algorithm to the offered traffic. These studies were carried using the simulation tool *CARWA* we developed within our project with T-Systems Nova.

We will see that the concept of the proposed new algorithms requires some computational effort. Therefore, we also look at methods to improve the performance of the **DSCA**-algorithms. To this end, we particularly consider the problem of finding the k shortest paths in a graph or digraph. We present a revised description of an algorithm of Martins, Pascoal, and Santos (see [MPS99]) that serves this purpose. In doing so, we state a new proof of the algorithm's correctness since the proof given in [MPS99] is insufficient, as we will show. Furthermore, we are able to answer the previously open question about the worst-case running time complexity of the algorithm.

Chapter 2 is devoted to the mathematical problem formulation of **DMCA** and **DSCA**. It includes a short introduction in that field of optimization which deals with problems of such kind, namely *online optimization*. Furthermore, the necessary notation used in this work is introduced. In Chapter 3, we present the new **DSCA**-algorithms. This chapter is subdivided into three sections that are concerned with the greedy-type algorithms, our new algorithms, and computational improvements of the latter. Chapter 4 yields the tools for this third part. One can view this chapter as an independent parenthesis that deals with the problem of finding the k shortest paths in a digraph. In Chapter 5, we present the experimental studies of the presented algorithms. To this end, we report on the considered networks, the used traffic and simulation model, and the obtained results. Chapter 6 is devoted to conclusions and an outlook. Finally, the appendix contains a table of notation, the used basic definitions on graph theory, and a short summary in German.

Chapter 2

Modeling of Dynamic Call Admission in Optical Networks

As described in the previous chapter, the *Dynamic Multiclass Call Admission Problem* (**DMCA**) and the *Dynamic Singleclass Call Admission Problem* (**DSCA**) consist of the following. So-called *connection requests* arrive over time, each of which specifies a pair of nodes in a given optical telecommunication network to be connected by a set of lightpaths for some time span. We will also refer to connection requests as *calls*. The network operator must process arriving calls dynamically, i.e., without knowledge of possible future connection requests. His first task is to decide whether some newly arriving call is accepted or rejected (*call admission*). Second, if the connection request is accepted, the operator must provide the specified number of lightpaths in order to connect the given end nodes for the required time interval (*routing and wavelength assignment*). Consequently, it is only possible to accept a given call if corresponding lightpaths can be established for that period of time.

In this chapter, we establish the mathematical model for **DMCA** and **DSCA**. Recall that the the first problem is suitable to reflect detailed requirements of real-world applications, while the latter is the simplified version for which we will propose algorithms later. The outline of the chapter is as follows. In Section 2.1, we give an introduction to the mathematical background of the considered problems, namely online optimization. Moreover, the standard instruments to evaluate the quality of online algorithms is explained. In Section 2.2, we introduce some special notations in order to specify the problem definitions, which are given in Section 2.3.

2.1 Online Optimization and Competitive Analysis

As mentioned before, **DMCA** and **DSCA** belong to a special class of optimization problems where the input data of an instance is not given completely in advance and becomes only known stepwise. In our context, the connection requests constitute this part of initially unknown data. Such problems must be solved *online*, i.e., decisions must be made without knowledge of future events by which further data is obtained.

These kinds of problems are called *online optimization problems*, or briefly *online problems*. The field of optimization that deals with online problems and the development and evaluation of corresponding *online algorithms* is consequently called *online optimization*. In the following, we will introduce the concepts of online optimization in more detail. Usually, the objective in an online problem is to maximize the gained profit or to minimize the cost. An online problem can be defined as a *request-answer game*, cf. [BEY98, Chapter 7]. For maximization problems, the definition is as follows.

Definition 2.1. An *online problem* is given by a triple $(\mathcal{R}, \mathcal{A}, \mathcal{C})$, where

- $\mathcal{R}$ is a set of requests $r_1, r_2, \ldots$,
- $\mathcal{A}$ is a sequence of answer sets $A_1, A_2, \ldots$, where A_j is the set of feasible answers for the jth request, and
- $\mathcal{C}$ is a sequence of profit functions $C_1, C_2, \ldots$, where $C_i : \mathcal{R}^i \times A_1 \times \ldots \times A_i \to \mathbb{R}_+$ and $C_i(r_1, \ldots, r_i, a_1, \ldots, a_i)$ is the total profit gained by giving answers $a_1, \ldots, a_i$ to requests $r_1, \ldots, r_i$ (the set $\mathcal{R}^i := \{(r_1, \ldots, r_i) \mid r_1, \ldots, r_i \in \mathcal{R}\}$ denotes the Cartesian product) .

For minimization problems, $\mathcal{C}$ is a sequence of cost functions. Online algorithms are defined as follows.

Definition 2.2. Let $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ be an online problem. An *online algorithm* for $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ is a sequence of functions $g_1, g_2, \ldots$, where $g_i : \mathcal{R}^i \to A_i$.

Note that this definition points out that an online algorithm must make decisions only based on the informations obtained by previous requests. For examples of well-investigated online problems and corresponding online algorithm, see [BEY98]. In contrast to online optimization, we also refer to classical optimization as *offline optimization*, where *offline problems* are solved using *offline algorithms*. Obviously, an online algorithm is no better than an optimal *offline* algorithm which knows the request sequence in advance. In order to evaluate the performance of an online algorithm ALG, the standard tool is the so-called *competitive analysis* (see [BEY98]). It is based on the idea to compare the profit made by ALG with the profit made by an optimal offline algorithm OPT that knows the sequence in advance and can process it at maximum profit. By $\mathsf{ALG}(\sigma)$ and $\mathsf{OPT}(\sigma)$, we denote the profit gained by ALG and OPT on a given sequence σ, respectively.

Definition 2.3. A deterministic online algorithm ALG is said to be *c-competitive* if

$$\mathsf{ALG}(\sigma) \geq \frac{1}{c} \cdot \mathsf{OPT}(\sigma)$$

holds for each sequence of requests σ. The *competitive ratio* of algorithm ALG is defined to be the infimum of all $c \geq 1$ such that ALG is c-competitive. ALG is called *competitive* if ALG is c-competitive for some $c \geq 1$.

Note that by definition, the total profit of a competitive algorithm must at least be a constant fraction of the profit that is achievable offline. As this must hold for all input sequences, competitive analysis is a type of worst-case analysis. Hence, in order to show that the competitive ratio of an online algorithm is bad, it suffices to find one sequence of requests on which the online algorithm looses too much compared with the optimal offline algorithm. For a given problem it is not clear in the beginning, whether there exists a competitive algorithm or not.

From another point of view, the competitive analysis of online algorithms may be considered as a game between the *online player* and a *malicious adversary*. The adversary yields an input sequence on which the online player applies an online algorithm. The adversary has complete knowledge of the algorithm's strategy, and he intends to construct a sequence such that the ratio of profit made by the online algorithm and the optimum offline profit is minimized. One possibility to reduce the power of the adversary is randomization. A *randomized online algorithm* is allowed to use random decisions for processing requests. The profit of a randomized algorithm is a random variable, and we are interested in its expectation. Normally, the malicious adversary knows the distribution used by the online player but cannot see the actual outcome of the random experiments. As a consequence, he must choose the complete input sequence in advance. This type of adversary is called *oblivious adversary* (see [BEY98] on different types of adversaries). For randomized online algorithms, we have the following definition of competitiveness.

Definition 2.4. A randomized online algorithm ALG is said to be *c-competitive* against the *oblivious adversary* if

$$\mathbb{E}[\mathsf{ALG}(\sigma)] \geq \frac{1}{c} \cdot \mathsf{OPT}(\sigma)$$

holds for each sequence of requests σ. The terms *competitive ratio* and *competitive* for randomized online algorithms are defined analogously to those for deterministic ones in Definition 2.3.

In this work, we only consider deterministic online algorithms. Finally, let us discuss the significance of competitiveness results for practical applications. As mentioned before, the competitive ratio of an online algorithm is a very pessimistic measure for the total profit the algorithm might gain compared to an optimal offline algorithm, since the competitive ratio is determined by worst-case scenarios. Therefore, the corresponding request sequences often represent pathological situations which are extremely unlikely to occur in reality. If this is the case, it is not possible to draw good conclusions from competitiveness results concerning the average behavior of an algorithm. On the one hand, if an online algorithm can be shown to be competitive with a good competitive ratio, this guarantees that the algorithm will perform well in any scenario. Its average case performace might even be better.

However, a competitive algorithm need not be usable for practice, since no requirements are made on its computational complexity. On the other hand, an online algorithm whose competitive ratio is bad or which is not competitive at all need not to perform bad on realistic instances. However, taking into account worst-case sequences

helps to characterize those scenarios that are dangerous for a good performace of the algorithm. In order to investigate the average-case behavior of an online algorithm, we rely on evaluation by simulation, as used in this work later on.

2.2 Notation

In this section, we focus on the necessary notations in order to formulate **DMCA** and **DSCA**, as well as algorithms for the latter, called **DSCA**-algorithms in the sequel. First of all, an optical network is modeled as follows.

Definition 2.5. An *optical network* is a triple (G,Λ,W), where

- $G=(V,E)$ is a simple and undirected graph,
- $\Lambda=\{\lambda_1,\ldots,\lambda_\chi\}$ is a set of wavelengths, and
- $W : E \to 2^\Lambda$ is a map from E to the power set of Λ with $\bigcup_{e\in E} W(e)=\Lambda$, which we call the *wavelength set function* of the optical network. If $W(e)=\Lambda$ for each edge $e\in E$, we say the optical network possesses *uniform edge equipment*.

The network nodes and the optical fibers of the physical topology are modeled as nodes and edges of the graph G. The wavelength set function indicates for each edge $e\in E$ which subset of wavelengths $W(e)$ is available on that edge. In this way, different fiber types and WDM systems can be modeled. Note that we can indeed model the underlying network topology as an undirected graph since although each signal is transmitted along a directed lightpath from the technical point of view, a corresponding backward channel using the same path must always be realized, too. This model captures optical networks with exactly one fiber per link on which an arbitrary WDM system may be installed. In order to allow for parallel fibers on one link, such that some wavelengths can be used on that link more than once, we would only need to neglect the property of G to be simple. Assuming that wavelength converters do not exist in the network, we represent a lightpath as a path in G together with a wavelength in Λ.

Definition 2.6. Let (G,Λ,W) be an optical network with graph $G=(V,E)$, let p be a path in G, and let $\lambda\in\Lambda$ be a wavelength. The pair (p,λ) is called a *lightpath* in the optical network (G,Λ,W) if $\lambda\in W(e)$ for each edge $e\in E(p)$. Its *length* is defined as the number of path edges $|E(p)|$. If p has end nodes $u,v\in V$, we call (p,λ) a $[u,v]$-*lightpath* or a lightpath with *end nodes* u and v.

Each lightpath satisfies the wavelength continuity constraint by definition, and thus the essential remaining restriction is the wavelength conflict constraint. It reads as follows.

> *Wavelength conflict constraint*: On every edge, each wavelength can be used by at most one lightpath. That is, for every two distinct simultaneously routed lightpaths (p_1,λ) and (p_2,λ) in the optical network that use the same wavelength, it must hold that $E(p_1)\cap E(p_2)=\emptyset$.

Since we will often be concerned with lightpaths whose routing would violate the wavelength conflict constraint, we also use the following terms.

Definition 2.7. Let (p_1,λ) and (p_2,λ) be two distinct lightpaths in an optical network (G,Λ,W) which use the same wavelength. We say that (p_1,λ) and (p_2,λ) *block*, *intersect*, or *overlap with* each other if $E(p_1)\cap E(p_2)\neq\emptyset$. Furthermore, a set of lightpaths in the network is called *conflict-free* if any two distinct lightpaths do not block each other.

Hence, all lightpaths contained in a conflict-free set can be routed simultaneously. Neglecting the possibility of failure situations, a wavelength which is installed on some edge $e\in E$ is either used on e by some lightpath or still unused. This time depending information defines the current *network status*. At each point in time, the network status is defined by the currently established lightpaths. It is usually stored by a so-called *link-status matrix*, where each row corresponds to an edge, and each column represents a wavelength.

Definition 2.8. Let (G,Λ,W) be an optical network with graph $G=(V,E)$ and wavelength set $\Lambda=\{\lambda_1,\ldots,\lambda_\chi\}$. Let $m:=|E|$ be the total number of edges in G, and let $E:=\{e_1,\ldots,e_m\}$. For a given network status S, let R_S be the set of currently realized lightpaths. If $R_S=\emptyset$, the optical network is called *empty*.

The *link-status matrix* of (G,Λ,W) in (network) status S is the $(m\times\chi)$-matrix $M_S=(m_{ij}^{(S)})_{1\le i\le m,1\le j\le\chi}$ with components defined as follows:

$$m_{ij}^{(S)}:=\begin{cases}1, & \text{if } \lambda_j\notin W(e_i) \text{ or there is a lightpath } (p,\lambda_j)\in R_S \text{ with } e_i\in p,\\ 0, & \text{otherwise.}\end{cases}$$

For $i=1,\ldots,m$ and $j=1,\ldots,\chi$, we say that wavelength λ_j *is available on* edge e_i in (network) status S if the corresponding matrix entry satisfies $m_{ij}^{(S)}=0$. If $\lambda_j\in W(e_i)$ but $m_{ij}^{(S)}=1$, we say that wavelength λ_j is *utilized* on e_i. Moreover, a lightpath (p,λ) in the optical network is called *free* or *available* in status S if λ is available on every edge $e\in E(p)$ in status S. Otherwise, the lightpath (p,λ) is called *blocked* in status S. If it is obvious which network status is considered, we also use the terms *currently* free, *currently* available, and *currently* blocked. If the network status changes by realizing an available lightpath (p,λ), we denote the resulting network status by $S+(p,\lambda)$.

Since each routing request requires that a connection between two appointed nodes $u,v\in V$ is established by some fixed $[u,v]$-lightpaths, it makes sense to classify lightpaths according to their end nodes. To this end, we use the following notations.

Definition 2.9. Let (G,Λ,W) be an optical network, where $G=(V,E)$. By $L(u,v,\lambda)$ we denote the set of lightpaths in (G,Λ,W) which connect the two nodes $u,v\in V$ and whose wavelength is λ. Furthermore, the set of all $[u,v]$-lightpaths is denoted by

$$L(u,v):=\bigcup_{\lambda\in W}L(u,v,\lambda),$$

and the set of all potential lightpaths in the optical network is denoted by

$$L:=\bigcup_{\substack{u,v\in V:\\ u\neq v}}L(u,v).$$

For a given network status S, we denote by L_S the set of all available lightpaths in status S and by $L_S(u,v) := L_S \cap L(u,v)$ the set of the currently free $[u,v]$-lightpaths for distinct nodes $u,v \in V$.

2.3 Problem Definition: Dynamic Multiclass Call Admission (DMCA)

We first state the general problem **DMCA** using a model that allows to represent many different real-world specifications. Simplifying assumptions which lead to the special problem considered in this work will be made later on. In the following, whenever we are given an optical network (G,Λ,W), the sets of nodes and edges of G are denoted by V and E, respectively.

A problem instance of **DMCA** is given by an optical network (G,Λ,W), a planning time interval $[T_1,T_2]$, an upper bound B on the number of lightpaths that can be requested, a set of customer classes C, a set of service classes Q, and a sequence of connection requests $\sigma := (\sigma_1,\sigma_2,\ldots)$, where each call σ_j, $j \in \mathbb{N}$ specifies the following parameters.

$u_j, v_j \in V$: End nodes of the connection
$b_j \in \{1,\ldots,B\}$: Number of lightpaths required (amount of demand)
$t_j^{\mathtt{arr}} \in [T_1,T_2]$: Arrival time of the call
$t_j^{\mathtt{ans}} \in [t_j^{\mathtt{arr}},T_2]$: Latest answer time (time by which the operator has to have made his decision whether to accept or to reject the request)
$t_j^{\mathtt{start}} \in [t_j^{\mathtt{ans}},T_2]$: Start time of the connection
$t_j^{\mathtt{stop}} \in [t_j^{\mathtt{start}},T_2]$: Expiration time of the connection
$c_j \in C$: Customer class
$q_j \in Q$: Service class
$p_j \in \mathbb{N}$: Profit of the connection

As a consequence, the duration of a call σ_j is $d_j := t_j^{\mathtt{start}} - t_j^{\mathtt{stop}}$. The profit p_j usually depends on the end nodes u_j and v_j (in particular on their distance in the network, i.e., the minimum length of a $[u_j,v_j]$-lightpath), the demand b_j, the time $t_j^{\mathtt{ans}} - t_j^{\mathtt{arr}}$ to perform the call admission part of the problem, the duration d_j, the customer class c_j, and the service class q_j. We assume that for the arrival times of any two connection requests σ_{j_1} and σ_{j_2} it holds that $t_{j_1}^{\mathtt{arr}} \le t_{j_2}^{\mathtt{arr}}$ if and only if $j_1 < j_2$. That is, the sequence is given in non-decreasing order of arrival times.

The task is to maximize the total profit gained by accepted calls such that *valid answers* are given to all connection requests. Let S_t be the planned network status at time $t \in [T_1,T_2]$ which results from decisions of previous answers. A valid answer to call σ_j consists of a pair (adm_j, L_j), where $adm_j \in \{$"accepted","rejected"$\}$ specifies whether call σ_j is accepted or rejected, and L_j is a conflict-free set of $[u_j,v_j]$-lightpaths

which are available in the planned network status S_t for each time $t \in [t_j^{\mathtt{start}}, t_j^{\mathtt{stop}}]$, such that the cardinality of L_j is

$$|L_j| := \begin{cases} 0, & \text{if } adm_j = \text{“rejected”}, \\ b_j, & \text{if } adm_j = \text{“accepted”}. \end{cases}$$

Note that the subdivision of each answer in two separate components reflects the call admission part and the routing and wavelength assignment part. The call admission part adm_j of the answer is given latest at time $t_j^{\mathtt{ans}}$ without knowledge of calls with later arrival times, the routing and wavelength assignment part L_j latest at time $t_j^{\mathtt{start}}$ without knowledge of calls which arrive after $t_j^{\mathtt{start}}$. If call σ_j is accepted, it contributes profit p_j to the total profit and its service requires that all lightpaths in L_j are established from $t_j^{\mathtt{start}}$ until $t_j^{\mathtt{stop}}$. As a consequence, the status S_t must be updated for each time $t \in [t_j^{\mathtt{start}}, t_j^{\mathtt{stop}}]$ by taking into account that the lightpaths in L_j are realized in this time interval. We refer to each lightpath that is contained in L_j as a *routing lightpath* for the call σ_j.

Notice that subsequent valid answers ensure valid routings for the planning horizon $[T_1, T_2]$, i.e., the wavelength conflict constraint will always be satisfied. For each accepted connection request, at most the two new network statuses $S_{t_j^{\mathtt{start}}}$ and $S_{t_j^{\mathtt{stop}}}$ have to be considered in addition to the updated ones. Hence, there is only a finite number of different network statuses to be stored if the sequence of requests σ is finite, and it takes finite time to determine whether a lightpath will be available in $[t_j^{\mathtt{start}}, t_j^{\mathtt{stop}}]$ or not. Obviously, **DMCA** is an online problem, as introduced in Definition 2.1.

2.3.1 Scope of the Model

The described model allows to represent many different settings. First, note that beyond the normal operating state where connection requests arrive and have to be processed, the formulation above can also reflect failure situations. Since a failure of an edge causes all established lightpaths using that edge to be suspended, and as a failing node corresponds to failures on all of its incident edges, the concerned connections have to be rerouted on lightpaths in the still operative part of the network, if possible. As a consequence, each planned network status up to the maximum original expiration time of the lightpaths is affected. Moreover, the failures imply a temporary change of the physical topology. For the link-status matrices which store the statuses, the impact is the following: All entries in the row of the matrix which corresponds to an edge that is either broken itself or incident to a broken node are set to one. This must be kept up until the failure is repaired.

The special task of rerouting connections in the case of a failure can be modeled as the arrival of a cumulative set of new requests, each of which corresponds to an original call σ_j and arrives at the time when the failure occurs $t_{\mathtt{fail}}$. For each new request, we set its latest answer time as well as its start time to $t_{\mathtt{fail}}$. The remaining parameters of the connection request are defined as those of the original call, except for the service class, which may additionally indicate that the call is to be rerouted due to failure, and the profit. The profit is set to zero since the customer has already

paid, and each rejected call yields a penalty cost that reflects the compensation which must be paid to the customer for the suspended connection. By recognizing the special service class, the task of call processing can be transferred to a special algorithm which can handle a large set of connection requests arriving simultaneously. Since such an algorithm disposes of more information, completely different routing procedures can be used, i.e., offline planning.

Furthermore, the request parameters model a variety of features. Different service classes enable the network operator to distinguish the diverse services he offers. For example, the service class of a call may indicate the priority of rerouting in case of network failures. A high priority may also force the network operator to reserve backup lightpaths that could be used immediately if necessary. Different customer classes may be treated differently, e.g., important customers might get a discount. This is easily modeled by a profit depending on the customer class. Note that the profit function has to be defined by the network operator in order to reflect his special business objectives. Its definition will affect the decisions of an algorithm substantially.

2.3.2 Restriction: Dynamic Singleclass Call Admission (DSCA)

From now on, we restrict ourselves to the problem **DSCA**. This problem is a restricted version of **DMCA** with the following assumptions.

1. $t_j^{\mathtt{arr}} = t_j^{\mathtt{ans}} = t_j^{\mathtt{start}}$ for each connection request σ_j.
2. There is only one customer class.
3. There is only one service class.
4. No failure situations occur.

Consequently, each connection request σ_j is defined by six components:

$$\sigma_j = (u_j, v_j, b_j, t_j^{\mathtt{start}}, t_j^{\mathtt{stop}}, p_j).$$

With only one customer class, one service class, and identical decision times, the profit of a call now only depends on the nodes u_j and v_j to be connected, the requested number of lightpaths b_j, and the duration of the connection d_j. Taking also into account the distance between the end nodes, the profit function p_j can be defined as the product of three functions that reflect the mentioned dependencies. Since a connection request must be handled immediatly at the time of its arrival, we assume that computations can be made in zero time.

2.3.3 Previous Work on DSCA

Let us review some competitiveness results for special versions of the problem **DSCA**. Let (G, Λ, W) be the given optical network and $\sigma = (\sigma_1, \sigma_2, \ldots)$, the request sequence. We assume the following restrictions to hold in any instance of **DSCA**:

1. Each call σ_j is permanent, i.e., $d_j = \infty$.
2. Each call σ_j requires only one lightpath: $b_j = 1$.
3. Each accepted call σ_j yields a profit $p_j = 1$.
4. The optical network possesses uniform edge equipment.

We refer to the resulting problem version as the *Dynamic Permanent Call Admission Problem* (**DPCA**) with *single demands*. If additionally only one wavelength is available in the network, i.e., $|\Lambda| = 1$, the wavelength conflict constraint implies that each edge can at most be used in one routing lightpath. Therefore, the routing and wavelength assignment part of **DPCA** with single demands reduces to the problem of finding edge-disjoint paths for the connection requests. The resulting call admission problem is called *Edge-Disjoint Paths Allocation* (**EDPA**). Via this problem, the following result about the complexity of the offline version of **DSCA** is obtained. Recall that in the offline version of an online problem, complete information about the request sequence σ is known in advance. Consider the decision variant of the offline version of **EDPA** with only two calls: Are there two edge-disjoint paths between two given pairs of end nodes in a graph? [GJ79] states that already this decision problem (a special *Undirected Two-Commodity Intergral Flow Problem*) is NP-complete.

Theorem 2.10. *The corresponding offline problems of* **DSCA** *and* **DPCA** *with single demands are NP-hard.*

For the problem **DPCA** with single demands, Awerbuch, Bartal, Fiat, and Rosén proposed an algorithm called *First-fit-coloring* (FFC) that uses an algorithm for **EDPA** as subroutine, called SLAVE in the sequel. FFC is based on the idea to view the optical network with χ wavelengths per edge as χ copies of the graph G, each of which represents one wavelength. The code of FFC for handling a single call that requires a connection between the nodes $u, v \in V$ is shown in Algorithm 1.

Input : An optical network (G, Λ, W) with $G = (V, E)$, $\Lambda = \{\lambda_1, \ldots, \lambda_\chi\}$, uniform edge equipment, and any network status; two nodes $u, v \in V$ between which a connection should be routed.
Output : An available $[u, v]$-lightpath or a message "rejected".

1 For each wavelength $\lambda \in \Lambda$, let SLAVE_λ be a copy of SLAVE, and let G_λ be the graph G restricted to all edges where λ is currently available;
2 If existing, choose the minimum index $1 \leq i \leq \chi$ such that $\mathsf{SLAVE}_{\lambda_i}$ accepts the connection request in graph G_{λ_i}, and return the lightpath (p, λ_i), where p is the path $\mathsf{SLAVE}_{\lambda_i}$ selects in G_{λ_i} for routing the connection;
3 If there is no algorithm SLAVE_λ for $\lambda \in \Lambda$ that accepts the request, return the message "rejected".

Algorithm 1: FFC

The following has been proved about the competitiveness of FFC.

Theorem 2.11 ([ABFR94]). *Let* SLAVE *be a c-competitive algorithm for* **EDPA**. *Then the algorithm* FFC *is* $(c+1)$*-competitive for* **DPCA** *with single demands.*

Striking about this result is that the competitive ratio of FFC does not depend on the number of wavelengths in the network and furthermore hardly differs from the competitive ratio of SLAVE. That is, if the competitive ratio of SLAVE is good, the one of FFC is good, too. Note that each algorithm for **EDPA** which accepts the first call is m-competitive, since the optimal offline algorithm OPT can at most accept m calls (one for each edge in the graph), where $m := |E|$. Unfortunately, competitive algorithms with better competitive ratio are only known for special graphs like lines, trees, and grid graphs. More precisely, the currently best competitive ratios of randomized algorithms are $\lceil \log n \rceil$ for the line with n nodes ([ABFR94, AAF+96]), $2\log n$ for the tree with n nodes ([ABFR94, AAF+96, LMSPR98]), and $O(\log n)$ for the $n \times n$ grid graph ([KT95, LMSPR98]). For arbitrary graphs, a lower bound for the competitiveness of any randomized algorithm for **EDPA** is $n^{1-\log_3 2/1+\log_3 2}$ ([BFL96]).

In [KP02], Krumke and Poensgen consider another version of **DSCA** which is less restrictive than **DPCA** with single demands. It allows for calls each of which requires up to χ lightpaths and yields a profit corresponding to the demands if it is accepted. We call the resulting problem **DPCA** with changing demands. It is easily shown that all deterministic competitive algorithms for this problem demands have the same competitive ratio.

Theorem 2.12. *For* **DPCA** *with changing demands, each deterministic algorithm is either not competitive or has a competitive ratio of* χm*, where* m *denotes the total number of edges and* χ *the number of wavelengths in the optical network.*

Proof. Let ALG be an arbitrary deterministic algorithm for **DPCA**. If ALG rejects the first given call, it is not competitive: For a request sequence $\sigma = (\sigma_1)$ consisting of one request, ALG makes zero profit, while OPT achieves at least a profit of one (depending on the demand of σ_1).

If ALG accepts the first call of a request sequence σ, we have $\mathsf{ALG}(\sigma) \geq 1$. Moreover, OPT can at most gain a profit of one for each edge and wavelength, yielding χm in total. This implies that ALG is χm-competitive.

It is shown in [KP02] that no deterministic algorithm can be better. Denote each call by (u_j, v_j, b_j) specifying the end nodes u_j, v_j for the connection and the number b_j of required lightpaths. Consider the line graph with nodes denoted by $v_1, \ldots, v_n$ from left to right and the following request sequence:

$$\sigma_1 = (v_1, v_n, 1), \sigma_2 = (v_1, v_2, \chi), \sigma_3 = (v_2, v_3, \chi), \ldots, \sigma_n = (v_{n-1}, v_n, \chi).$$

Since ALG accepts σ_1, it must reject all subsequent calls, achieving a total profit of $\mathsf{ALG}(\sigma) = 1$. In contrast, OPT accepts the calls $\sigma_2, \ldots, \sigma_n$ at a total profit of χm. □

In [KP02], Krumke and Poensgen propose the first randomized competitive algorithms for **DPCA** with changing demands. They show that the bound of χm does not

hold for the competitive ratio of randomized algorithms. Their algorithm called *First-fit-coloring-scaled* (FFCS) uses FFC as subroutine and achieves a competitive ratio of $8(c+1)(\lceil \log \chi \rceil)+1)$ for the line and $12(c+1)(\lceil \log \chi \rceil)+1)$ for the tree, again given a c-competitive algorithm for **EDPA**. However, the algorithm FFCS is only of theoretical interest. It will perform bad in practice.

2.3.4 Motivation for the Use of a Simulation Based Evaluation

Theorem 2.12 reveals that competitive analysis is not an appropriate method to evaluate the quality of deterministic online algorithms for **DPCA** with changing demands. Although all deterministic competitive algorithm perform equally bad with respect to their competitive ratios, their practical quality can differ significantly. The more general **DSCA** with limited durations and with uniform edge equipment might in principle yield different results, but it is too complex to be evaluated by competitive analysis. Note that Theorem 2.12 also yields a bound on the competitve ratio of an deterministic **DSCA**-algorithm from which follows that there are no **DSCA**-algorithms which achieve a better competitive ratio. This motivates the use of simulation for the evaluation of online algorithms for **DSCA**.

Chapter 3

Online Algorithms for DSCA

In this chapter, we present existing and new online **DSCA**-algorithms which are developed in order to perform well in practical applications within our joint project with T-Systems Nova.

The basic versions of each algorithm gets as input an optical network (G,Λ,W), its current network status S, and two node between which a connection of one lightpath should be established. That is, an algorithm receives the connection requests of the sequence σ consecutively, and each call σ_j only requires one lightpath, i.e., $b_j = 1$. Note the algorithms can be easily adapted for calls of higher demands: We simply treat a call σ_j that requires b_j lightpaths as a collection of b_j similar calls with demand one each, also referred to as *call fragments*. The collection of calls is referred to as a *call packet* of size b_j. The only difference is that the original call σ_j must is rejected if any of its corresponding call fragments is rejected. Furthermore, all presented **DSCA**-algorithms does not make use of optional call rejection. That is, a connection request is always accepted if possible.

The structure of this chapter is as follows. In Section 3.1, we present algorithms that are based on shortest path routing with varying wavelength selecting strategies. Most of these so-called *greedy-type algorithms* have been proposed by Mokhtar and Azizoglu in [MA98]. Furthermore, we consider adapted versions that incorporate, among other things, special tie-breaking rules whose impact on the performance of the greedy-type algorithms will turn out to be substantial in the experimental results in Chapter 5. Most of these greedy-type algorithms have been proposed in a joint work with Hülsermann, Jäger, Krumke, Poensgen, and Rambau, see [HJK+03]. However, in the main part of the chaper (Section 3.2), we propose new **DSCA**-algorithms that are based on the concept of the *network fitness*. In order to reduce the substantial computational effort of most of these algorithms, we propose (among others) one easy method, whereby the decisions of the adapted algorithms will change only little.

Throughout this chaper, whenever we are given an optical network (G,Λ,W), the set of nodes is denoted by V, and the set of edges of G is denoted by E. Furthermore, let $\Lambda = \{\lambda_1,\ldots,\lambda_\chi\}$ be the set of wavelengths unless something else is stated.

3.1 Greedy-Type Algorithms

In nowadays networking applications, many routing algorithm are based on routing along shortest paths. This also applies to the presented *greedy-type algorithms* for the problem **DSCA**. They accept given connection requests whenever it is somehow possible to provide corresponding lightpaths, thus omitting the call admission part of the problem. Among each other, the greedy-type algorithms vary only in their wavelength selection strategy. For the routing choice, the strategy applied by all greedy-type algorithms is to establish a shortest lightpath among all currently availalble lightpaths in the chosen wavelength. Recall that the length of a lightpath is defined to be the number of its edges. If no lightpath of any wavelength connecting the start and end node is available, the call is rejected.

Many of the greedy-type algorithms in this section are defined in terms of the following quantities which depend on the network status.

Definition 3.1. Let (G,Λ,W) be an optical network with network status S. Moreover, let $M_S = (m_{ij}^{(S)})_{1\le i\le m, 1\le j\le \chi}$ be the corresponding link-status matrix, where $m := |E|$ is the total number of edges in the network. For each $j = 1,\ldots,\chi$, we call

$$util_S(\lambda_j) := \sum_{\substack{i=1:\\ \lambda_j\in W(e_i)}}^{m} m_{ij}^{(S)}$$

the *edge utilization of* wavelength λ_j in (network) status S and

$$avail_S(\lambda_j) := \sum_{i=1}^{m} (1 - m_{ij}^{(S)})$$

the *edge availability of* wavelength λ_j in (network) status S.

The classification of greedy-type algorithms in this chapter follows the work of Mokhtar and Azizoglu [MA98], who proposed the variants called FIXED, RANDOM, PACK1, SPREAD1, and a version of EXHAUSTIVE in the following. In their paper, only the optical networks with uniform edge equipment were considered. However, if the edges provide different sets of wavelengths, the algorithms PACK1 and SPREAD1 have to be adapted by incorporating the current edge availability of wavelengths instead of the current edge utilization in order to implement their underlying ideas. Furthermore, especially for EXHAUSTIVE, the description in the paper leaves the tie-breaking decision open, which can considerably affect the performance of the algorithm. By the specification of a reasonable tie-breaking rule, we could improve this algorithm significantly compared to the version breaking ties randomly. Most of the new versions have already been proposed together with the co-authors Hülsermann, Jäger, Krumke, Poensgen, and Rambau in [HJK+03].

We distinguish between two classes of greedy-type algorithms, which differ in their way of wavelength selection.

3.1.1 Partial Wavelength Search

Upon arrival of a connection request, the algorithms of the first class partially search the wavelengths in a certain order until they find the first wavelength λ in which a connection can be established. Therefore, they are called greedy-type algorithms with *partial wavelengths search*. The given call is routed on a shortest lightpath using wavelength λ. All of these algorithms are based on the generic greedy approach of Algorithm 2 and differ in the way *how* the order of wavelengths in Step 1 is chosen.

Input : An optical network (G, Λ, W) with any network status; two nodes $u, v \in V$ in the network between which a connection should be routed.

Output : An available $[u, v]$-lightpath if there is any or a message "rejected" otherwise.

1 Let $\lambda_{i_1}, \ldots, \lambda_{i_\chi}$ be some order on the set of all wavelengths Λ;
{*The way how the order of the wavelengths is chosen leads to different versions of the algorithm, see text.*}
2 For each wavelength $\lambda \in \Lambda$, let G_λ be the graph G restricted to all edges where λ is currently available;
3 Choose the first wavelength λ in the order where there is still a path in G_λ connecting u and v. If no such wavelength exists, return the message "rejected";
4 Compute a shortest $[u, v]$-path p in G_λ and return the lightpath (p, λ).

Algorithm 2: Generic greedy-type algorithm with partial wavelengths search.

The variants we summarize under the name greedy-type algorithms with partial wavelengths search use the following wavelength orders (cf. Definition 3.1 for the quantities edge utilization and edge availability).

- FIXED: The wavelength search order is fixed a priori.
- RANDOM: Wavelengths are searched in a randomly varying order (using a uniform distribution).
- PACK1: Wavelengths are searched in order of decreasing edge utilization.
- PACK2: Wavelengths are searched in order of increasing edge availability.
- SPREAD1: Wavelengths are searched in order of increasing edge utilization.
- SPREAD2: Wavelengths are searched in order of decreasing edge availability.

Notice that, except for FIXED, the wavelength orders change over time. Furthermore, the two versions of PACK and SPREAD, respectively, are equivalent if the network possesses unifrom edge equipment. This can formally be shown as follows.

Theorem 3.2. *Let (G,Λ,W) be an optical network with uniform edge equipment. Moreover, let S be the network status and $M_S = (m_{ij}^{(S)})_{1\le i\le m, 1\le j\le \chi}$ be the corresponding link-status matrix, where $m := |E|$ is the total number of edges in the network. Then,* PACK1 *and* PACK2 *as well as* SPREAD1 *and* SPREAD2 *are equivalent concerning their routing decisions, that is, both algorithms choose the same lightpath for a given connection request if they use identical rules for tie-breaking.*

Proof. Note that by definition of the wavelength set function, $W(e) = \Lambda$ for each edge $e \in E$. It suffices to show for each $j,k = 1,\ldots,m$ that

$$util_S(\lambda_j) < util_S(\lambda_k) \iff avail_S(\lambda_j) > avail_S(\lambda_k).$$

By definition of the edge utilization and edge availability, we obtain:

$$\begin{aligned} util_S(\lambda_j) - util_S(\lambda_k) &= \sum_{\substack{i=1:\\ \lambda_j\in W(e_i)}}^{m} m_{ij}^{(S)} - \sum_{\substack{i=1:\\ \lambda_k\in W(e_i)}}^{m} m_{ik}^{(S)} \\ &= \sum_{i=1}^{m} (m_{ij}^{(S)} - m_{ik}^{(S)}) \\ &= \sum_{i=1}^{m} \Big((1-m_{ik}^{(S)}) - (1-m_{ij}^{(S)}) \Big) \\ &= avail_S(\lambda_k) - avail_S(\lambda_j). \end{aligned}$$

□

The idea of PACK1 is to route given calls on a lightpath using the most utilized wavelength λ_{i_1} as long as possible. Further wavelengths are not taken into account unless no potential routing lightpath (p,λ_{i_1}) is currently available for the given connection request. For this reason, if the network is initially empty, all wavelengths except λ_{i_1} remain unused until arrival of the first call that cannot be routed using λ_{i_1} anymore. Applying this strategy in an optical network with uniform edge equipment, the probability that for a path p the lightpath (p,λ_{i_k}) is currently available tends to decrease with decreasing k. That is, PACK1 intends to fill up the network load in such wavelengths for which it is less likely to find possible lightpaths.

This strategy seems reasonable since the little remaining capacities of currently frequently utilized wavelengths can rarely be used, and therefore one should try to use them whenever it is possible. If one refrain from routing on heavily utilized wavelengths until the network load gets high, it is unlikely that one arriving call can be routed using such a wavelength, which might lead to its rejection. Moreover, other wavelengths in which a call can more likely be routed are prevented from being utilized if not necessary.

However, PACK1 might route some connection requests on long lightpaths even though there are much shorter free lightpaths using less utilized wavelengths. This looks disadvantageous since much network capacity is required. As shown in the prove of Theorem 3.2, the most utilized wavelength is also the less available wavelength for uniform edge equipment. But obviously, this must not necessarily hold if the edges

provide different sets of wavelengths. Hence, the idea described above is no longer realized by PACK1 in this case. We can maintain the idea of PACK1, if wavelengths are searched in order of increasing edge availability. This is done by the greedy-type algorithm PACK2.

Obviously, the concept of both SPREAD versions are completely opposite to those of PACK. Instead of filling up the network in some wavelengths, SPREAD1 attempts to distribute the total network load uniformly among all wavelengths, which also balances the remaining capacities in the different wavelengths but only if the network has uniform edge equipment. That is, SPREAD1 achieves similar edge utilizations and similar edge availablities for the different wavelengths.

If the optical network is equipped with different sets of wavelengths on the edges and the network load is higher, the edge utilization of those wavelengths which exist on many edges will be greater than the utilization of less often installed wavelengths. Therefore, the latter will be continuously preferred by SPREAD1. Furthermore, the edge availablilities of the different wavelengths will no longer be similar anymore. This is still achieved by SPREAD2 which searches the wavelengths in order of decreasing edge availability.

Moreover, we consider two version of FIXED. The first one, FIXED1, searches the wavelengths in order of increasing index while the second, FIXED2, searches in order of decreasing index. If the network has uniform edge equipment, both FIXED1 and FIXED2 perform equally with interchanged wavelengths. By the used procedure of network dimensioning in this work, an edge $e \in E$ is equipped with the wavelengths $\lambda_1, \ldots, \lambda_{\chi_e}$, where $\chi_e := |W(e)|$. Hence, both FIXED variants are opposite in this case. While FIXED1 tends to select more frequently installed wavelengths, FIXED2 chooses frequently installed wavelengths. Note that the edge availability of a wavelength λ in a network with medium load tends to be larger the more λ is installed in the network. That is, edge availability of a wavelength λ_i tend to decrease with increasing index. Hence, the wavelength selection strategies are similar for FIXED1 and SPREAD2 and for as FIXED2 and PACK2 for the used way of network dimensioning. This tendency will be confirmed by the experimental results in Chapter 5.

3.1.2 Total Wavelength Search

In contrast to the greedy-type algorithms with partial wavelengths search, the algorithms of the second class always take into account the total set of wavelengths by computing shortest available lightpaths for *all* wavelengths. Among them, a globally shortest one is chosen. Hence, we call these algorithms greedy-type algorithms with *total wavelengths search* or shortening *exhaustive*. They only differ in their rule for tie-breaking if there are several wavelengths which yield globally shortest lightpaths. The proceeding of the greedy-type algorithms with *total wavelengths search* is depicted in Algorithm 3.

The tie-breaking orders to be considered correspond to the wavelengths search orders of the greedy-type algorithms with partial wavelengths search.

- EXHAUSTIVE$_f$: The wavelength tie-breaking order is fixed a priori.

Input : An optical network (G,Λ,W) with $G=(V,E)$, $\Lambda=\{\lambda_1,\ldots,\lambda_\chi\}$, and any network status; two nodes $u,v\in V$ between which a connection should be routed.

Output : An available $[u,v]$-lighpaths if there is any or a message "rejected" otherwise.

1 Let $\lambda_{i_1},\ldots,\lambda_{i_\chi}$ be some order on the set of all wavelengths Λ;
{*The above order is used as a tie-breaking rule which leads to different versions of the algorithm, see text.*}
2 For each wavelength $\lambda\in\Lambda$, let G_λ be the graph G restricted to all edges where λ is currently available;
3 For each wavelength $\lambda\in\Lambda$ compute a shortest $[u,v]$-path p_λ in G_λ. If no path is found at all, return the message "rejected";
4 Let $\lambda'\in\Lambda$ be the first wavelengths in the order such that $|E(p_{\lambda'})|\leq|E(p_\lambda)|$ for all wavelength $\lambda\in\Lambda\setminus\{\lambda'\}$. Return $p_{\lambda'}$.

Algorithm 3: Generic greedy-type algorithm with total wavelengths search.

- EXHAUSTIVE$_r$: Wavelengths are ordered randomly varying (using a uniform distribution).
- EXHAUSTIVE$_{p1}$: Wavelengths are ordered by decreasing edge utilization.
- EXHAUSTIVE$_{p2}$: Wavelengths are ordered by increasing edge availability.
- EXHAUSTIVE$_{s1}$: Wavelengths are ordered by increasing edge utilization.
- EXHAUSTIVE$_{s2}$: Wavelengths are ordered by decreasing edge availability.

Compared to the greedy-type algorithms with partial wavelength search, it seems to be advantageous to choose globally shortest lightpaths because less network capacity is required in order to realize them. Since there are often many globally shortest lightpaths (particularly if the network load is small), additional tie-breaking rules should significantly affect the behaviour of the greedy-type algorithms with total wavelength search. The ideas of different tie-breaking rules are the same as before. As for FIXED, we also consider two versions for EXHAUSTIVE$_f$ that are equivalent if the network has uniform edge equipment: EXHAUSTIVE$_{f1}$ orders the wavelengths by increasing index and EXHAUSTIVE$_{f2}$ by decreasing index.

3.1.3 Analysis of Greedy-Type Algorithms

In this section, we look at the properties of the greedy-type algorithms. First, all greedy-type algorithms are very easy to implement. Their main ingredient is the computation of shortest paths. Since all edge lengths are one in our length model, this subproblem can be solved in linear time $O(n+m)$ using breadth-first search, where $n:=|V|$ and $m:=|E|$ denote the number of nodes and edges in the network, respectively (cf. [KV00, Chapter 2]). Moreover, the restriction of the graph G to G_λ as

well as the computation of $util_S(\lambda)$ or $avail_S(\lambda)$ is done in $O(m)$ time for each wavelength $\lambda \in \Lambda$, and network status S. Since the optical network is equipped with χ wavelengths, the worst case complexity of all greedy-type algorithms is $O(\chi(n+m))$. However, their average case complexity differs as follows. While the exhaustive versions always have to determine a shortest $[u,v]$-lightpath in each wavelength, the greedy-type algorithms with partial wavelengths search compute shortest lightpaths at most χ times. In particular, if the network load is not very high, the latter will only need few shortest path computations. Moreover, this number is also affected by the different wavelengths search orders. Above all SPREAD2, but also RANDOM, may find a free $[u,v]$-lightpath quickly, whereas FIXED, and naturally PACK2, need longer to find an available connection. For a stochastic analysis on the number of shortest path computations of the greedy-type algorithms with partial wavelengths search, see [MA98].

Even though the greedy-type algorithms have efficient running times, it is crucial for each **DSCA**-algorithm that ignores the call admission part how decisions concering routing and wavelength assignment are made. Such decisions are reasonable if as much network capacity as possible stays available in order to accept potential future calls. The motivation of the greedy-type algorithms for this is the following: The shorter a lightpaths (p,λ), the less network capacity is required by realizing it. However, this only holds for the number of edges where λ is available, but we are interested in available lightpaths instead of edges.

The following example illustrates a scenario in which the greedy type algorithms fail, because a single decision already proves to be bad: It leads to an allocation which makes the routing of many further calls impossible.

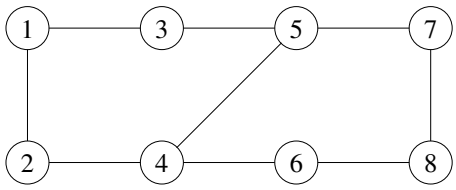

Figure 3.1: First example graph G_1.

Example 3.3. Consider the optical network (G_1,Λ,W) with G_1 as depicted in Figure 3.1, $\Lambda = \{\lambda\}$, and $W(e) = \Lambda$ for each edge $e \in V(G_1)$. Suppose that each call requires a connection between nodes 3 and 6 and yields a profit of one if it is accepted. Upon arrival of a first call, each greedy-type algorithm routes it on the lightpath $((3,5,4,6),\lambda)$ since this lightpath has the minimum length of 3 while both other alternative lightpaths have length 4. Its path, however, is a $[3,6]$-*cut* in the network separating the nodes $1,2,3,4$ from the remaining nodes $5,6,7,8$. Hence, only one call can be routed at each point in time. But since the two lightpaths $((3,1,4,6),\lambda))$ and $((3,5,7,8,6),\lambda))$ are edge-disjoint, two connection request could be realized simultaneously. In doing so, twice as much profit could be made if the frequency of call arrivals is sufficiently high. What was the mistake of the greedy-type algorithm? The problem is that it simply maximized the number of arcs on which λ stays available after the routing. If we instead realize such a lightpath that the number of lightpaths which remain available is maximum, another decision will be made (this approach leads to

the first algorithm proposed by us in Section 3.2). While the realized lightpath keeps available the 12 lightpaths with paths

$$(3,1),(3,1,2),(3,1,2,4),(1,2),(1,2,4),(2,4),$$
$$(5,7),(5,7,8),(5,7,8,6),(7,8),(7,8,6),(8,6),$$

both other routing alternatives yield 14 available lightpaths: For $((3,5,7,8,6),\lambda)$, the corresponding paths are

$$(3,1),(3,1,2),(3,1,2,4),(3,1,2,4,5),(3,1,2,4,6),$$
$$(1,2),(1,2,4),(1,2,4,5),(1,2,4,6),$$
$$(2,4),(2,4,5),(2,4,6),$$
$$(4,5),(4,6).$$

Furthermore, if it is know that there is only traffic between the nodes 3 and 6, lightpaths connecting these nodes are obviously much more important than others. In this case, realizing the lightpath $((3,5,4,6),\lambda)$ is obviously bad since each other $[3,6]$-lightpath is blocked afterwards. If instead of G_1, the graph G_2 which is shown in Figure 3.2 forms the underlying topology of the optical network and if all calls need connections between node 7 and node 12, the profit made by each greedy-algorithm is only $1/3$ of the optimal profit: In this network, all greedy-type algorithms route the lightpath $((7,10,8,11,9,12),\lambda)$, and now there are even three edge-disjoint $[7,9]$-lightpaths.

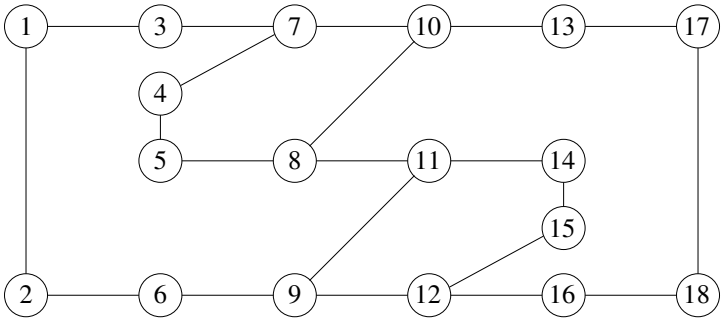

Figure 3.2: Example graph G_2.

Without specification of the lightpaths that remain available if another routing lightpath is selected, we state that mentioned strategy will realize another lightpath and achieve the optimal profit. Furthermore, beginning with G_1 and G_2, we can construct a sequence of graphs such that the greedy-type algorithms become arbitrarily bad against the optimal **DSCA**-algorithm which must accept calls if possible. The general idea is that the shortest path is a cut in these networks.

Although Example 3.3 represents a worst-case scenario for the greedy-type algorithms, it shows that always choosing a shortest lightpaths can be very bad. Moreover, it motivates the development of more intelligent algorithms.

3.2 Network Fitness Algorithms

In this section we present new **DSCA**-algorithms in order to overcome the discussed drawbacks of greedy-type algorithms.

3.2.1 Introduction

All **DSCA**-algorithms that we propose are based on the concept of the *network fitness*. This corresponds to an approach often used in online optimization. For a given system, one aspires to evaluate its different possible statuses resulting in a *fitness function*. Using such a fitness function, one intends to draw conclusions concerning the decisions to make. For **DSCA**, the idea is to route a given connection request, if accepted, in such a way that the resulting network status allows for a maximum potential profit of future calls. Since we do not know the calls to come, we can only relate potential future profit at any point in time to the current network status. Hence, we would like to assign a *fitness* level to each network status that reflects the network's capability to accept further calls. Obviously, it is desirable to maintain the status of the network as fit as possible. To this end, we route an accepted call in such a way that the resulting decrease in network fitness is minimized, or equivalently, the remaining network fitness after lightpath realization is maximized. After expiration of a routed connection, the fitness value increases again. The crucial task is to define a proper measure for the network fitness.

Probably the most reasonable definition of the term network fitness is the expected maximum future profit which can be obtained for a coming sequence. Recall that the offline version of the **DSCA** is NP-hard, as shown in Theorem 2.10. Hence, the computation of the expected value of maximum future profitit is completely out of reach since solving the corresponding stochastic program is even harder. In this section, we will propose different simplified measures for the current fitness of a network which can be computed more easily.

The proposed **DSCA**-algorithms only differ in their fitness functions. This function maps a network status S of a given optical network to a non-negative real number $fit(S)$, which shall measure the corresponding fitness of the network. Omitting the call admission part of the problem, all network fitness algorithms are based on the scheme of Algorithm 3.2.1. However, the selection of the returned lightpath in Step 5 may include a tie-breaking rule, and the cost definition in Step 4 may vary a little when additional weight functions are used.

Note that also the greedy-type algorithms with total wavelengths search (see 3.1) belong to this class of algorithms. Given a network status, its fitness value is defined as the sum of the edge availabilities for all wavelengths. Even the greedy-type algorithms with partial wavelengths search can be regarded as network fitness algorithms: They attempt to maintain as much edge availability as possible but the wavelengths are weighted corresponding to the wavelength order.

Input : An optical network (G,Λ,W) with any network status S; two nodes $u,v \in V$ between which a connection should be routed;

Output : An available $[u,v]$-lighpaths if there is any or a message "rejected" otherwise.

1 Let $fit : \mathcal{S} \to \mathbb{R}_+$ be some network fitness function, where $\mathcal{S}$ denotes the set of all possible network statuses of (G,Λ,W);
{*The exact definition of this function leads to different versions of the algorithm.*}
2 Let $L_S(u,v) \subseteq L(u,v)$ be the set of currently free $[u,v]$-lightpaths in status S;
3 If $L_S(u,v) = \emptyset$, return the message "rejected";
4 For each lightpath $(p,\lambda) \in L_S(u,v)$, let $c(S,(p,\lambda)) := fit(S) - fit(S+(p,\lambda))$ be its cost, which is defined as the difference of the network fitness before and after routing of the lightpath (p,λ);
5 Return a lightpath $(p,\lambda) \in L_S(u,v)$ whose cost $c(S,(p,\lambda))$ is minimum.

Algorithm 4: Generic network fitness algorithm.

3.2.2 The Algorithm ALR

In contrast to the greedy-type algorithms, the first proposed network fitness algorithm *available-lightpaths-reduction* (ALR) keeps a view on the impact of all lightpaths in the optical network. Let (G,Λ,W) be an optical network with network status S.

Recall that for each pair of distinct nodes $u,v \in V$, the set of currently available $[u,v]$-lightpaths is denoted by $L_S(u,v) \subseteq L(u,v)$. The fitness function of ALR is defined as

$$fit_{\mathsf{ALR}}(S) := \sum_{\substack{u,v \in V: \\ u \neq v}} |L_S(u,v)|.$$

That is, the network fitness is the total number of lightpaths which are currently free with respect to the network status S. Consequently, the corresponding cost of an available lightpath (p,λ) is

$$c_{\mathsf{ALR}}(S,(p,\lambda)) = \sum_{\substack{u,v \in V: \\ u \neq v}} |L_S(u,v)| - |L_{S+(p,\lambda)}(u,v)|.$$

Hence, ALR routes a given call on such a lightpath for which the decrease of available lightpaths caused by its routing is minimized, i.e., the algorithm maintains as many lightpaths as possible.

Compared to the greedy-type algorithms, it seems advantageous that ALR directly takes into account the availability of whole lightpaths, because these are the necessary utilities in order to establish connections. Only to consider the edge availabilities, as done by greedy-type algorithms, certainly affects the availability of lightpaths, but the impact may be very different. For example, let G be a line graph with n succeeding nodes $v_1,\ldots,v_n$, where n is even, and let $\lambda \in W(e)$ for each edge $e \in E$. Realizing a first single-edge lightpath $((v_i,v_{i+1}),\lambda)$ in the empty network, where $i \in \{1,\ldots,n-1\}$,

leads to the same decrease of edge availability of the utilized wavelength independent of the position of edge $v_i v_{i+1}$ on the line. For the decrease of available lightpaths using wavelength λ, however, the impact of that position is significant. If $i = 1$, that is, the edge lies at the boundary of the graph, only the $n-1$ lightpaths which connect v_1 with the other nodes become blocked. But if the edge is located in the middle of the graph, i.e., $i = n/2$, the graph is cut into two parts each of which contain $n/2$ nodes. Hence, for each two nodes v_i and v_j with $1 \leq i \leq n/2 < j \leq n$, the only lightpath connecting them become blocked. The total number of those lightpaths is $(n/2)^2$, which is for large n much more than in the first case. As shown by this example, the number of edges of a lightpath does not correlate with the number of lightpaths blocked by it. Therefore, the measure of the greedy-type algorithms may lead to bad routing decisions if the edges differ in their importance for routing of calls. As already shown, this is the case in Example 3.3, where ALR performs optimally.

A major drawback of ALR is concerned with its computational complexity: It is very unlikely that the cost computation of ALR can be implemented efficiently. Given two nodes u and v in a graph, it is shown in [Val79] that the problem of counting the number of $[u,v]$-paths is #P-complete. In Section 3.2.2, we present a Branch-and-Bound method for ALR which turned out to be very fast in practice compared to our first implementation.

For a given path p in G, the set of all other paths which share an edge with p could theoretically be calculated in a preliminary step of the algorithm. One way to store this information is to encode it as a graph which we call the *path graph* $G_{\mathcal{P}}$ of G. Formally, $G_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ is defined by

$$V_{\mathcal{P}} := \{ p \mid p \text{ is a path in } G \},$$

and

$$E_{\mathcal{P}} := \{ p_1 p_2 \in V_{\mathcal{P}}^{(2)} \mid E(p_1) \cap E(p_2) \neq \emptyset \},$$

where $V_{\mathcal{P}}^{(2)}$ denotes the set of unordered pairs of elements in $V_{\mathcal{P}}$. However, the actual determination of the path graph $G_{\mathcal{P}}$ of a graph G might be computationally too expensive and too memory consuming, since the size of the path graph $G_{\mathcal{P}}$ is exponential in the size of G. Indeed, the path graph of the 17-nodes network (see Section 5) contains 13641 nodes (where each node in $G_{\mathcal{P}}$ corresponds to an undirected path in G). In the computation of its path graph on a state-of-the-art PC, a memory overflow occurred. At that time, 12663 nodes and $59,373,015$ edges of the path graph were determined.

Once we have the path graph, we simply obtain for each path p in G the number of other paths which intersect p as the degree of p in $G_{\mathcal{P}}$. Obviously, the path graph would be useful for ALR. For each wavelength λ, we store a path graph $G_{\mathcal{P}}(\lambda) = (V_{\mathcal{P}}(\lambda), E_{\mathcal{P}}(\lambda))$ of the graph G restricted to all edges $e \in E$ where λ is installed, i.e., $\lambda \in W(e)$. Upon arrival of a connection request between nodes $u, v \in V$, the cost of each $[u,v]$-lightpath using wavelength λ can be accessed in constant time if each node in $G_{\mathcal{P}}(\lambda)$ stores its degree. After a lightpath (p,λ) has been established it is necessary to update $G_{\mathcal{P}}(\lambda)$. However, the computational effort of this update is again considerable. Each adjacent node $p' \in V_{\mathcal{P}}(\lambda)$ of p as well as p itself has to be removed from $G_{\mathcal{P}}(\lambda)$ temporarily which includes to reduce the degree of each neighbor node of p' by 1. A

similar update has to be made once a routed connection using a lightpath (p,λ) expires: The temporarily removed adjacent nodes of p in $G_{\mathcal{P}}(\lambda)$ have to be restored if their other neighbors also correspond to currently free lightpaths. Furthermore, the degrees of all these nodes have to be updated. In an application where routing decisions have to be made very fast after arrival of a call and where is much time between the arrival of two calls, the above implementation could indeed work if the considered network is small.

One conceptual disadvantage of the above definition of network fitness is that all lightpaths are considered to be equally important. However, there are different reasons why a lightpath should be more important to be available than another.

First of all, the average demands between the network nodes may be given, i.e., we know calls which arrive more frequently than others. Obviously, corresponding lightpaths are of particular importance: Lightpaths connecting nodes $u_1, v_1 \in V$ are more important than lightpaths which connect the nodes $u_2, v_2 \in V$ if the average demand between u_1 and v_1 is greater than that between u_2 and v_2. For a pair of nodes which are never required to be connected, lightpaths connecting them are of no importance.

Second, we would like to protect specially those lightpaths whose end nodes can currently be connected only by few free lightpaths. For example, if arrivals of requests for connections between network nodes $u_1, v_1 \in V$ and $u_2, v_2 \in V$ are equally likely but there are more free lightpaths left which connect u_1 and v_1, we would like to protect $[u_2, v_2]$-lightpaths more than $[u_1, v_1]$-lightpaths. Therefore, if two possible routing lightpaths for the same call might block the same set of lightpaths except that the first intersects one $[u_1, v_1]$-lightpath, whereas the second intersects one $[u_2, v_2]$-lightpath, the first routing should be chosen.

The last criterion is the length of a lightpath. Obviously, a lightpath which contains almost all nodes in the network (nearly Hamiltonian) will hardly be established since it blocks a huge number of other lightpaths. Such a lightpath is completely unimportant, whereas short lightpaths which connect the same nodes will be realized often and are therefore more valuable to be protected.

In the next section, we will extend our basic version of ALR by a weight function which takes into account the different importances of lightpaths.

ALR with Generalized Weight Functions

As already mentioned, the lightpaths in an optical network (G, Λ, W) are not equally important. In this section we will define a weight function in order to take into account their different importances. Given a certain network status S, the weight w of a lightpath (p,λ) in the network whose end node are u and v depends on three factors. We set

$$w(S,(p,\lambda)) := w_1(u,v) \cdot w_2(S,u,v) \cdot w_3(p),$$

where w_1 is a function that depends only on the end nodes of the path p, w_2 also depends on the current network status, while w_3 takes into account the structure of p itself. These three weight functions reflect the three ways the importance of the

$[u,v]$-lightpaths (p,λ) may be affected (see last section): the average traffic demand between u and v, the number of currently available $[u,v]$-lightpaths, and the length of p. Note that the weight function does not depend on the wavelength of the lightpaths. We propose the following variants for the three functions.

For the first function w_1, which reflects the importance of a lightpath resulting from the relative average demand between its end nodes, we distinguish two options `C` (= constant) and `T` (= traffic):

(`C`) $w_1(u,v) := 1$,

(`T`) $w_1(u,v) := \text{demand}(u,v)$,

where $\text{demand}(u,v)$ denotes the average arrival frequency of calls which require connections between the nodes $u,v \in V$ in any period of time. If call arrivals between different pairs of nodes are different likely, the option `T` seems reasonable since the higher the demand between the nodes u and v compared to other node pairs, the higher is the probability that the next arriving call requires a $[u,v]$-lightpath, and the more important it is to protect these lightpaths.

As before, let $L_S(u,v) \subseteq L(u,v)$ be the set of available $[u,v]$-lightpaths with respect to network status S. For the second function w_2 that considers the number of available lightpaths, we propose the following variants `C` (= constant), `L` (= linear), and `E` (= exponential):

(`C`) $w_2(S,u,v) := 1$,

(`L`) $w_2(S,u,v) := |L_S(u,v)|^{-1}$,

(`E`) $w_2(S,u,v) := \mu^{|L_S(u,v)|^{-1}} - 1$,

where $\mu > 1$ is some parameter. Both options `L` and `E` take into account the distribution of still available lightpaths. More precisely, the fewer available $[u,v]$-lightpaths are left in comparison to their initial number, the more important they are. This aims at preventing the algorithm from making a routing choice which decreases the relative connectivity between two nodes too much. Rather than routing a given call in such a way that the last available lightpath between two nodes becomes blocked, the algorithm chooses a routing lightpath which might block more free lightpaths, but only those whose end nodes are still easily connectable. This is achieved by relating the weight inversely to the number of available lightpaths with the same end nodes. Notice that the versions (`L`) and (`E`) of $w_2(S,u,v)$ are only well defined if not all lightpaths connecting u and v are blocked yet. However, the weight $w_2(S,u,v)$ will only be calculated by `ALR` if this is the case.

In order to take into account the lengths of the lightpaths, we consider the following versions for the last function w_3 labeled `C` (= constant), `L` (= linear), and `E` (= exponential):

(`C`) $w_3(p) := 1$,

(L) $w_3(p) := n - |E(p)|$,

(E) $w_3(p) := \nu^{\frac{n-|E(p)|}{n-1}}$.

Here, $n := |V|$ again denotes the total number of nodes in the network and $\nu > 1$ is some parameter. The linear (L) and exponential (E) versions of function w_2 make it more expensive to block short lightpaths. As already mentioned, this realizes the following idea. The longer a lightpath, the more network capacity tends to be required by realizing it. Therefore, long routing lightpaths are usually not chosen as long as there are available lightpaths left. Hence, they may be allowed to become blocked anyway, while short lightpaths are worth to be protected.

The exponential weight function in the third versions of w_2 and w_3 aims at increasing the corresponding weight more than linearly when the number of available $[u,v]$-lightpaths decreases or the length of p increases linearly, respectively. Hence, the effect is strengthened compared to each linear second version.

In using exponential weight functions, we adopt the main feature of the algorithm AAP for online call admission and routing in electronic networks which was proposed by Awerbuch, Azar, and Plotkin [AAP93]. However, it is to be doubted whether the exponential function in SALR is as effective as it proved to be for AAP which is good in theory (competitive analysis) and (in an adapted version) also in practice.

Having introduced the weight function, we define the cost of a lightpath (p,λ) in some network status S for the weighted version of ALR by

$$c_{\mathsf{ALR}}(S,(p,\lambda)) := \sum_{(q,\lambda)\in B} w(S,(q,\lambda)),$$

where $B := L_S \setminus L_{S+(p,\lambda)}$ is the set of lightpaths which are available with respect to network status S and which overlap with (p,λ). That is, instead of taking into account the number of currently free lightpaths which become blocked as before, the cost contribution of each of those lightpaths is now its weight. Note that unless $w \equiv 1$, this definition does not correspond directly to the general fitness algorithm cost function since it cannot be expressed to be the change in some network fitness.

However, there are still situations in which the usage of w_2 with option L or E is not as effective as desired. These may occur if at least two lightpaths with the same end nodes become blocked. For example, if a first routing lightpath would in total block two of four available $[u_1,v_1]$-lightpaths and two of four available $[u_2,v_2]$-lightpaths, it has the same cost value as another lightpath which would block all four available $[u_1,v_1]$-lightpaths since the weight of each of the four newly blocked lightpaths is the same, respectively ($w_2(S,u_1,v_1) = w_2(S,u_2,v_2)$). In particular, if connection requests between u_1 and v_1 are at least as likely to arrive as those between u_2 and v_2, the first choice seems better since the number of available lightpaths between one pair of nodes is not decreased too much. By realizing the second lightpaths, u_1 and v_1 may soon be not connectable anymore.

We will refer to the different combination of the three weight function by their corresponding letters, e.g., the combination with (T) for w_1, (L) for w_2, and (E) for w_3

is denoted by TLE. As mentioned before, the original unweighted version of ALR fits into this scheme: it is equal to version CCC. Experimental results were carried out for all 8 combinations of the three weight functions that result from (C) or (T) for w_1 and (C) or (E) for w_2 and w_3. These are reported in Chapter 5.

An Implementation of ALR Using Branch and Bound

Upon the arrival of a connection request between nodes s and t in an optical network (G,Λ,W), the network fitness algorithm ALR has to determine all currently available lightpaths connecting s and t and calculate for each of these its cost in order to determine a cheapest one. Moreover, each cost value is computed as the sum of weights of newly blocked lightpaths. In doing so, again a huge number of lightpaths has to be taken into account. In this section, we describe a Branch-and-Bound method which determines a cheapest routing lightpath fast and can be applied to all weighted versions of ALR. As a subroutine for the computation of the newly blocked lightpaths, it uses a recursive method which is also described below. Together, these new techniques resulted in a speed up of factor 40 in comparison to the previously implemented procedures that consider for each edge of a possible routing lightpath all lightpaths intersected by this edge.

We want to prevent, if possible, the cost of a lightpath from being computed completely and independently from the cost of other lightpaths. To this end, we construct in a preprocessing step for any pair of distinct nodes $s,t \in V$, a so-called *(s,t)-prefix tree*. The (s,t)-prefix tree is a unique way of representing all paths in G which connect s and t. It is a rooted tree with the following set of nodes. Given a node $v \in V$, it contains one node for each $[s,v]$-path p_1 in G which can be extended to an $[s,t]$-path, i.e., there is a path p_2 from v to t such that the concatenation p_1p_2 does not contain a cycle. In other words, given an $[s,t]$-path $p = (s,u_1,\ldots,u_i,t)$ in G, the tree has one corresponding node for each of the paths $(s),(s,u_1),\ldots,(s,u_1,\ldots,u_i),p$. Note that a path can be specified by the sequence of its nodes since G is required to be simple. All nodes in depth $i \in \mathbb{N}_0$ represent paths of length i: The root node corresponds to the empty path (s), and its children are those single-edge paths whose edges are incident to s and contained in any $[s,t]$-path. Given a node in the tree which represents a subpath $(s,v_1,\ldots,v_j)$ of a $[s,t]$-path, its children are obtained by attaching those edges $v_jv_{j+1} \in E$ for which there exists a path from v_{j+1} to t whose attachment to $(s,v_1,\ldots,v_j,v_{j+1})$ does not yield a cycle. Hence, the leafs of the tree are exactly the $[s,t]$-paths in the network.

In the following, we present a recursive algorithm which determines for a given node $s \in V$ the (s,t)-prefix trees for all nodes $t \in V \setminus \{s\}$. Starting with the empty path (s), we process all paths in G with end node s by successively attaching edges $v_jv_{j+1} \in E$ to the currently considered path $p := (s,v_1,\ldots,v_j)$, where $v_1,\ldots,v_j \in V$, and $0 \leq j \leq n-1$ such that $q := (s,v_1,\ldots,v_j,v_{j+1})$ does not contain a cycle. Then, the path q is inserted into the (s,v_{j+1})-prefix tree under construction along with all its subpaths $(s),(s,v_1),\ldots,(s,v_1,\ldots,v_{j-1}),p$ which have not yet a corresponding node in the tree. Hence, for each constructed path, $O(n)$ nodes are inserted in some prefix tree.

This is done best by starting the construction with the node for path q followed by that for p, that for $(s, v_1, \ldots, v_{j-1})$, and so on.

Notice that a path from s to some node $v \in V \setminus \{s\}$ may have corresponding nodes in many (s,t)-prefix trees. For each previously found path and each prefix tree, if the tree contains a node which corresponds to the path, we store a pointer to it. Hence, for storing q and its subpaths in the (s, v_{j+1})-prefix tree, it can be determined in constant time whether there is already a node for a subpath of q in the (s, v_{j+1})-prefix tree. Note that for the first subpath q' which has already a node in the tree, all its ancestors correspond to the remaining shorter subpaths of q such that the construction of nodes in the tree terminates with path q'. All together, it takes $O(n)$ time to insert the nodes for these paths. Afterwards, similar to depth-first search ([KV00, Chapter 2]), this method is iteratively applied to q.

Once there is no edge left which is allowed to be attached to the current path q, the procedure continues at its predecessor p by considering the remaining edges. Since it can be determined in constant time whether the attachment of an edge yields a cycle by using an array that stores for each node in V the information whether it is already contained in the current path, we can detect in $O(n)$ time whether no more edges can be attached to path q. Therefore, using this algorithm, we can construct all prefix trees in $O(kn)$ time, where k is the total number of paths in G. Next, we will prove that this running time is optimum. For the 17-nodes network (see Section 5) this algorithm constructs all prefix trees in less than one second on a state-of-the-art PC.

Theorem 3.4. *Let $G = (V,E)$ be a graph, and let k be the total number of paths in G. Then the running time complexity to construct all prefix trees in G is $\Theta(kn)$, where $n := |V|$ is the total number of nodes in G.*

Proof. In order to proof that the computation can not be performed faster, consider the line graph with succeeding nodes $v_1, \ldots, v_n$. Since there is exactly one $[v_i, v_j]$-path for $i, j \in \{1, \ldots, n\}$ with $i \neq j$, the total number of paths is $\binom{n}{2} = n(n-1)/2$. Hence, it suffices to show that the total number of nodes in the $n(n-1)/2$ prefix trees is $\Theta(n^3)$. Note that for $i, j \in \{1, \ldots, n\}$ with $i < j$ the (v_i, v_j)-prefix tree has $j - i + 1$ nodes since the path from v_i to v_j has $j - i$ nodes, resulting in $j - i + 1$ subpaths (the $[v_i, v_j]$-path

inclusive). Therefore, the total number of nodes in all trees is

$$\begin{aligned}
\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(j-i+1) &= \sum_{i=1}^{n-1}\left((n-i)+\sum_{j=i+1}^{n}(j-i)\right)\\
&= \sum_{i=1}^{n-1}\left((n-i)+\sum_{l=1}^{n-i}l\right)\\
&= \sum_{i=1}^{n-1}\left((n-i)+\frac{(n-i)(n-i+1)}{2}\right)\\
&= \sum_{i=1}^{n-1}n-\sum_{i=1}^{n-1}i+\sum_{i=1}^{n-1}\frac{(n-i)(n-i+1)}{2}\\
&= n(n-1)-\frac{n(n-1)}{2}+\frac{1}{2}\cdot\sum_{i=1}^{n-1}(n^2-2ni+n+i^2-i)\\
&= \frac{n(n-1)}{2}+\frac{1}{2}\cdot\sum_{i=1}^{n-1}(n^2-(2n+1)i+n+i^2)\\
&= \frac{n(n-1)}{2}+\frac{n^2(n-1)}{2}-(2n+1)\frac{n(n-1)}{4}+\frac{n(n-1)}{2}\\
&\quad +\frac{(n-1)n(2n-1)}{12}\\
&= n(n-1)+\frac{2n^2(n-1)}{4}-\frac{2n^2(n-1)+(n-1)}{4}+\frac{n(n-1)}{2}\\
&\quad +\frac{(n-1)n(2n-1)}{12}\\
&= n(n-1)-\frac{n(n-1)}{4}+\frac{n(n-1)(2n-1)}{12}\\
&= \Theta(n^3).
\end{aligned}$$

□

Given the (s,t)-prefix tree, a cheapest lightpath with respect to the cost function of ALR connecting nodes s and t can be computed fast as follows. For each wavelength, the tree is traversed in a depth-first manner, starting at the root node. A child and all its descendants need not be taken into account if its newly attached edge, i.e., that one which is not contained in the path of the parent node, is not available anymore in the considered wavelength. For each node which is passed in the process, its cost is computed, that is, the cost of the corresponding lightpath. This can be done easily by incrementing the cost of its parent node by the cost of the attached edge. The cost of an edge in turn is calculated by the recursive procedure described below.

The first advantage of using the prefix tree is that we usually do not need to compute the costs of all $[s,t]$-lightpaths in order to determine a cheapest one: Once the first leaf of the tree is reached during the process, we obtain an upper bound on the cost of the cheapest $[s,t]$-lightpath. Obviously, the cost of a node is less or equal than the cost of each of its children. Hence, when processing a node whose cost is not smaller than the currently best upper bound, we can prune the tree at this node since it will

never lead to a leaf which is cheaper than the cheapest one known yet. That is, in this case we can discard the node and all its descendants and continue with the procedure at its parent node. This Branch-and-Bound method results in the pruning of many subtrees, and therefore, it avoids computing the cost of a lot of lightpaths which cannot be cheapest. However, in the worst case where we can never prune the tree, the cost of all contained paths is computed. The smallest bound on the total number of nodes in the (s,t)-prefix tree is the number of paths in G which have s as one end node. The worst case example is again the line graph as defined above, where the degenerated (v_1, v_n)-prefix tree has n nodes which equals the number of paths with end node v_1.

Furthermore, the computing efficiency is affected by the determination of the cost of a node in the prefix tree for the considered wavelength. As mentioned before, the cost of a node is the sum of the cost of its parent node and the cost of the attached edge. Let q be the path that corresponds to any node in the tree different from the root, let p be the path that corresponds to the parent node, and let $e \in E$ be the edge whose attachment to p yields q. The cost of e depends on p and the considered wavelength $\lambda \in \Lambda$, and it is obtained as the sum of the weights of all lightpaths which

a) are currently available in wavelength λ,

b) contain the edge e, and

c) do not contain any edge in p.

Constraint c) prevents us from adding the weight of a lightpath, becoming newly blocked by routing the considered lightpath (q, λ), more than once to the cost of the node. Obviously, by adding the weights of those lightpaths to the cost of the parent node we obtain exactly the cost which was defined in the last section.

The set of the lightpaths whose weights we have to sum up can again be represented by a tree which is constructed by the following recursive procedure. Different than the prefix trees it cannot be stored in advance since it depends on many things, e.g., the network status and the path p. Hence, it also makes no sense to store it. Instead of the two end nodes of a lightpath, we will refer to a tail node and a head node in the further description. The root of the tree is the lightpath using wavelength λ which only consists of the edge e whose cost we wish to compute. Its tail and head nodes may be chosen arbitrarily. Then, we construct the *tail children* of a tree node by attaching a suitable edge to the tail node of the corresponding lightpath, and the other end node of that edge becomes the tail node of the resulting lightpath. An edge is suitable if the constructed lightpath satisfies conditions a) and c) above. Analogously, the *head children* are constructed, and their head nodes are defined. The children of these tail and head children themselves are then obtained in the same way, except for one additional rule: A node constructed as a head child is not allowed to have tail children. This restriction ensures that no lightpath appears twice in the tree.

The leaves of the tree obtained in this way are those lightpaths which cannot be extended anymore. In this tree, the weights of all nodes have to be computed and added up. This is done during the construction of the tree. Since the number of nodes

in the tree equals in the worst-case the number of paths which contain the considered edge, the size of the tree can be exponential in the size of the graph, even if the graph is planar (for an example, see Section 4.5). But note that the number of nodes reduces significantly if the considered wavelength is not available anymore on every edge (condition a)), and particularly if the path p in condition c) is not the empty path.

Additionally to the pruning of subtrees, the described procedure has a second advantage in the computation of a cheapest lightpath compared with the possiblity to determine the cost of each potential routing lightpath for a given call directly. Let $s,t \in V$ be the nodes to be connected. Note that the paths corresponding to most interior nodes in the (s,t)-prefix tree can usually be extended to many different $[s,t]$-paths, i.e., the subtrees rooted at these nodes are large. By computing the costs of these nodes with respect to a given wavelength, we can reuse these partial calculations for the cost computation of their descendants. If the costs of the $[s,t]$-lightpaths, i.e., the leaves in the tree, are determined directly, the cost of joint ancestors have to be counted several times. For example, let p be the path of an interior node, and let $p_1,\ldots,p_k$ for some $k \in \mathbb{N}$ be the leaves of the subtree rooted at p. That is, p is a joint subpath of $p_1,\ldots,p_k$. If the costs of the lightpaths $(p_1,\lambda),\ldots,(p_k,\lambda)$ had been determined independent of each other, all free lightpaths which would become blocked by routing (p,λ) would have been counted k times. Since the set of those lightpaths may be huge, the computation time can reduced significantly by the proposed data structure.

The improvement in computing time for the determination of a cheapest routing lightpath achieved by the described Branch-and-Bound method is substantial. In comparison with calculating the costs of all possible routing lightpaths independent of each other, the described procedure results in a speed up by a factor of 40.

Selective ALR

The variants we summarize under the name *Selective* ALR (SALR) are specializations of the weighted versions CEC and TEC. Only these versions are considered since we propose another way to take into account the lengths of lightpaths becoming blocked. The idea of SALR is the following. So far, in calculating the cost of a routing lightpath, both versions consider *all* lightpaths which would be become blocked by it. In our opinion, the strategy has two disadvantages: first, it is time-consuming because of the huge number of lightpaths, and second, as already told, those lightpaths which are long and use edges in the outskirts of the network are less important for routings than short and central lightpaths. In order to overcome these drawbacks, we restrict the lightpaths SALR takes into account as follows. In calculating the cost of a routing choice, only the weights of those affected lightpaths are counted which belong to the l shortest ones between their two end nodes for some $l \in \mathbb{N}$. Therefore, the total number of lightpaths in the optical network which can contribute to the cost when they become blocked is at most

$$l \cdot \frac{n(n-1)}{2},$$

which is polynomial in the size of the graph. Hence, the cost of a given lightpath can now be efficienty computed. If the set of considered routing choices is also restricted to

a polynomial number, as described in Section 3.2.5, we can also efficiently determine a cheapest routing lightpath. Depending on l, we will denote the versions of this algorithm by SALR_l.

3.2.3 The Algorithm SFR

As described in the last section, the algorithm ALR in its basic version uses the number of currently available lightpaths as measure for the fitness of the network. Hence, for each lightpath on which a given call can be routed, ALR computes the total number of lightpaths which are currently free but would become blocked if this lightpath was realized. Then, it chooses a routing lightpath whose realization yields the smallest number of newly blocked lightpaths.

However, the currently available lightpaths in the optical network are obviously in general all not *edge-disjoint*, i.e., some of them may have an edge in common. In particular, if the actual load in the network is small, a huge number of lightpaths will be free and many of them will intersect. Therefore, it would not be possible to establish all of them simultaneously, due to the wavelength conflict constraint. Hence, counting all available lightpaths might not be the best of measures for the potential future profit since this definition of network fitness usually overestimates the profit substantially.

This drawback motivated the definition of another network fitness algorithm for **DSCA** which we call *single-flow-reduction* (SFR). The reason for this name will become apparent later. Again, we are given an optical network (G,Λ,W) with $G=(V,E)$ and a network status S. Instead of taking into account all available lightpaths in the network, SFR computes for each pair of distinct nodes $s,t \in V$ the maximum number of edge-disjoint $[s,t]$-lightpaths which are still available. More precisely, we denote by $d_S(s,t,\lambda)$ the maximum number of edge-disjoint $[s,t]$-lightpaths which use wavelength $\lambda \in \Lambda$ and which are available in status S. Furthermore, we define for each wavelength $\lambda \in \Lambda$ the current network fitness in this wavelength by

$$d_S(\lambda) := \sum_{\substack{s,t \in V: \\ s \neq t}} d_S(s,t,\lambda).$$

Consequently, the total fitness function of SFR is defined as

$$fit_{\mathsf{SFR}}(S) := \sum_{\lambda \in \Lambda} d_S(\lambda).$$

Since realizing a lightpath (p,λ) only affects the network fitness in wavelength λ, the cost of such a lightpath can be computed by

$$\begin{aligned} c_{\mathsf{SFR}}(S,(p,\lambda)) &= fit_{\mathsf{SFR}}(S) - fit_{\mathsf{SFR}}(S+(p,\lambda)) \\ &= d_S(\lambda) - d_{S+(p,\lambda)}(\lambda). \end{aligned}$$

Before we turn to the question how the values d_S can be computed efficiently, let us mention that, in general, not all of those lightpaths counted in $d(\lambda)$ are edge-disjoint,

since lightpaths connecting different nodes may still overlap. But much fewer lightpath than counted by ALR will intersect, since the considered $[s,t]$-lightpaths are edge-disjoint for two distinct nodes $s,t \in V$. In the Section 3.2.4, we describe an approach in which the network fitness is measured by the maximum cardinality of a set of available lightpaths such that any two lightpaths in that set are edge-disjoint, that is, all of them could be routed simultaneously.

In addition, we also consider a weighted version of SFR, called SFR(T), which takes into account the different average traffic demands for different connections, similar to the version (T) of the weight function w_1 for ALR in Section 3.2.2. To this end, we adapt the definition of the value $d_S(\lambda)$ for the network status S and a wavelength $\lambda \in \Lambda$ as follows:

$$\bar{d}_S(\lambda) := \sum_{\substack{s,t \in V: \\ s \neq t}} \text{demand}(s,t) \cdot d_S(s,t,\lambda),$$

where $\text{demand}(s,t)$ denotes the average arrival frequency of calls which require connections between the nodes $s,t \in V$ in any period of time. The cost function of SFR(T) is analogously defined to that of SFR, but in terms of the values $\bar{d}_S(\lambda)$. Using this cost function, a decrease of the number of available edge-disjoint lightpaths which connect important end nodes causes a higher cost contributions than a decrease between rarely connected end nodes. In doing so, one aspires to protect particularly lightpaths for important connections.

Implementation of SFR

The implementation of the algorithm SFR works as follows. At any time, SFR stores for each pair of distinct nodes $s,t \in V$ and each wavelength $\lambda \in \Lambda$ the maximum number of currently available edge-disjoint $[s,t]$-lightpaths using wavelength λ. Furthermore, upon arrival of a connection request, the algorithm needs to compute for each possible routing lightpath the same values that would result from realizing it. Given a network status S, two distinct nodes $s,t \in V$, and a wavelength $\lambda \in \Lambda$, the value $d_S(s,t,\lambda)$ can be determined efficiently via solving the instance $(D,s,t,\kappa_S^{(\lambda)})$ of the well-known *maximum flow problem*, where parameters are defined as follows. D emerges from G by replacing each edge in E by two opposite directed arcs, i.e., $D = (V,A)$, where

$$A := \{(u,v),(v,u) \mid uv \in E\}, \tag{3.1}$$

and s is set to be the source and t the sink. For each arc $(u,v) \in A$, let $e((u,v)) := uv \in E$ be the corresponding edge in G whose end nodes are u and v. Finally, the arc capacities of the instance are defined as

$$\kappa_S^{(\lambda)}(a) := \begin{cases} 1, & \text{if } \lambda \text{ is available on arc } e(a) \text{ in status } S, \\ 0, & \text{otherwise.} \end{cases}$$

The *maximum flow problem* is defined as follows. A function $x : A \to \mathbb{R}$ is called an (s,t)-*flow* if it fulfills the *capacity constraints*

$$0 \leq x(a) \leq \kappa_S^{(\lambda)}(a) \text{ for all } a \in A,$$

and the *flow conservation constraints*

$$\sum_{a\in\delta^-(v)} x(a) = \sum_{a\in\delta^+(v)} x(a) \text{ for all } v \in V \setminus \{s,t\},$$

where $\delta^-(v) := \{(u,v) \in A\}$ and $\delta^+(v) := \{(v,u) \in A\}$. The *value* of a flow x is defined by

$$\text{val}(x) := \sum_{a\in\delta^+(s)} x(a) - \sum_{a\in\delta^-(s)} x(a).$$

The task of the *maximum flow problem* is to find an (s,t)-flow whose value is maximum.

In the following, we show that the value $d(s,t,\lambda)$ can indeed be derived as the optimal solution of the instance defined as above.

Theorem 3.5. *Let (G,Λ,W) be an optical network with graph $G=(V,E)$ and network status S, let $s,t \in V$ be two distinct nodes, and let $\lambda \in \Lambda$ be an arbitrary wavelength. Then, the maximum number of available edge-disjoint $[s,t]$-lightpaths $d_S(s,t,\lambda)$ in the network equals the maximum value of an (s,t)-flow in the digraph $D=(V,A)$ with arc capacities $\kappa_S^{(\lambda)}(a)$ for each $a \in A$ as defined above.*

Proof. As stated by the max-flow min-cut theorem ([AMO93, Chapter 6]), the maximum value of an (s,t)-flow equals the minimum capacity of an (s,t)-cut. Let D' be the subgraph of D which is restricted to the arcs with capacity one, i.e., $D' := (V,A')$, where $A' := \{a \in A \mid \kappa_S^{(\lambda)}(a) = 1\}$. Obviously, the minimum capacity of an (s,t)-cut in $(D,\kappa_S^{(\lambda)})$ equals the minimum cardinality of an (s,t)-cut in D'. The book [AMO93, Chapter 6] also contains a proof that the latter is the same as the maximum number of arc-disjoint (s,t)-paths in D', denote it by h. Hence, it remains to be shown that h equals $d_S(s,t,\lambda)$. Obviously, $d_S(s,t,\lambda) \leq h$ since given a set of free edge-disjoint $[s,t]$-lighpaths, for each such lightpath (p,λ), we can construct a corresponding directed (s,t)-path in D' since for each edge in $E(p)$ both corresponding arcs in D' exist. In order to prove $d_S(s,t,\lambda) \geq h$, note that we can construct for a given set of arc-disjoint (s,t)-paths in D' a corresponding set of available edge-disjoint $[s,t]$-lightpaths if for any two adjacent nodes $u,v \in V$ not both opposite directed arcs $(u,v),(v,u) \in A'$ are contained in these paths. However, if this is the case, we can construct another set of arc-disjoint (s,t)-paths in D' as follows. Let

$$p = p_1(u,(u,v),v)p_2 \text{ and } q = q_1(v,(v,u),u)q_2$$

be two such paths (the concatenation of two paths p and q is denoted by pq). Then, we define

$$p' := p_1q_2 \text{ and } q' := q_1p_2.$$

Obviously, these paths may contain cycles. Let p'' and q'' be the corresponding paths after removing all cycles, respectively. Notice that replacing p and q by p'' and q'' still yields a maximum set of arc-disjoint paths in D', but the number of pairs of opposite directed arcs which are both used is reduced by at least one (by removing cycles more such pairs may be eliminated). Repeating this replacement, we obtain a valid set of arc-disjoint (s,t)-paths in D' which is still maximum, and in which only one of the arcs (u,v) and (v,u) is contained for each $u,v \in V$. □

That is, we have shown that the cost of a possible routing lightpath can be computed as the decrease in flow values for single commodities, which motivates the name Single-flow-reduction for the considered **DSCA**-algorithm. Finally, let us consider the running time of the cost computation for SFR. Denote by $n := |V|$ the total number of nodes and by $m := |E| = |A|/2$ the total number of edges in G. Efficient algorithms to solve the *maximum flow problem* are known: For arbitrary capacities, the Goldberg-Tarjan algorithm solves the problem with running time $O(n^3)$ and even faster, using sophisticated data structures [AMO93, Chapter 7]. If all arc capacities are either 0 or 1, the running time can be improved to $O(\min\{n^{2/3}m, m^{3/2}\})$ (cf. [AMO93, Chapter 8]).

This shows that the numbers $d_S(s,t,\lambda)$ can be computed efficiently. In order to obtain for each possible routing lightpath (p,λ) its cost $c_{\mathsf{SFR}}(S,(p,\lambda))$, SFR needs to compute the values $d_{S+(p,\lambda)}(s,t,\lambda)$ for any pair of distinct nodes $s,t \in V$ by solving the corresponding instance of the *maximum flow problem*. Hence, for each routing choice, SFR has to solve $n \cdot (n-1)/2$ instances. That is, the total running time for this task is $O(n^2 \cdot \min\{n^{2/3}m, m^{3/2}\})$. Another possibility in order to obtain the maximum number of edge-disjoint lightpaths for each pair of nodes in G is to use the Gomory-Hu algorithm [KV00, Chapter 8] which requires only $n-1$ flow computations in total. However, since the capacities change during that procedure due to node contractions, solving each instance of the *maximum flow problem* takes $O(n^3)$ time. In total, we obtain a running time of $O(n^4)$, which is faster than the procedure above for networks with $m = \Omega(n^{4/3})$.

The Algorithm ASFR

So far, all of the proposed routing algorithms only distinguish between lightpaths which are currently available at the point in time when a new connection request arrives and which are not. Consequently, a lightpath (p,λ) not available at arrival of a call σ_j will not be accounted for when computing the cost of a routing lightpath for σ_j. However, all previously routed lightpaths which currently block (p,λ) might expire soon, whereupon the lightpath would become available again. But some possible routing lightpaths for σ_j could also intersect (p,λ) and it would not become available again if such a lightpath was established. In this case, it may be preferable to realize a different lightpath. Therefore, it makes sense to take into account also those lightpaths which are currently blocked at arrival of σ_j but become available very soon thereafter.

In order to identify them, however, we need that each given connection request σ_j also specifies its start and expiration times $t_j^{\mathtt{start}}$ and $t_j^{\mathtt{stop}}$, which is assumed from now on. Hence, it is known for each edge and each wavelength currently utilized on it, at which point in time the wavelength will be available again. The following simple example illustrates a concrete situation in which it makes sense to take into account the start and expiration times of calls, too.

Example 3.6. Consider the optical network defined by the three-node circle shown in Figure 3.3, together with two wavelengths λ_1 and λ_2 installed on each edge. The given network is 2-edge-connected, i.e., there are two edge-disjoint paths connecting each pair of nodes. Moreover, there are no further $[s,t]$-paths for any two distinct nodes s

and t. Therefore, SFR and ALR are equivalent if applied to this network. Assume that the algorithm is given a sequence of calls, beginning with σ_1 and σ_2, each of which requires a connection between the node 1 and 2. Let $t_i^{\mathtt{stop}} = t_i^{\mathtt{start}} + 1$ for $i = 1,2$, and $t_2^{\mathtt{start}} = t_1^{\mathtt{stop}} - \varepsilon$. That is, both call have duration one, and the first call ends very shortly after the beginning of the second call.

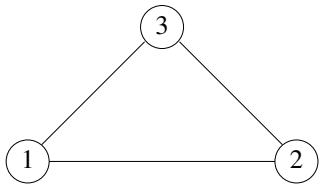

Figure 3.3: Circle with three nodes.

Upon arrival of the first call σ_1, it can be routed on the path $(1,2)$ or the path $(1,3,2)$ using either wavelength λ_1 or λ_2. While the first path intersects $(1,2)$, $(1,2,3)$, and $(2,1,3)$, the second intersects $(1,3)$, $(2,3)$, $(1,2,3)$, $(1,3,2)$, and $(2,1,3)$. Hence, the costs are

$$c_{\mathsf{SFR}}((1,2),\lambda_1) = c_{\mathsf{SFR}}((1,2),\lambda_2) = 3$$

and

$$c_{\mathsf{SFR}}((1,3,2),\lambda_1) = c_{\mathsf{SFR}}((1,3,2),\lambda_2) = 5.$$

Consequently, SFR realizes for instance the lightpath $((1,2),\lambda_1)$. As the second call σ_2 arrives, the first connection is still active. Therefore, the cost of lightpath $((1,3,2),\lambda_1)$ has decreased. It is now

$$c_{\mathsf{SFR}}((1,3,2),\lambda_1) = 3$$

since $((2,1,3),\lambda_1)$ and $((1,2,3),\lambda_1)$ are not available anymore due to the first routing. Obviously, SFR may either establish lightpath $((1,2),\lambda_2)$ or $((1,3,2),\lambda_1)$. However, $((1,2),\lambda_2)$S with cost 3 is clearly the better choice: Call σ_1 will expire soon (within the next ε time units), and then, routing lightpath $((1,3,2),\lambda_1)$ will also block the two lightpaths $((1,2,3),\lambda_1)$ and $((3,1,2),\lambda_1)$ until σ_2 expires. Furthermore, in this symmetric optical network, it is obviously better to use a single-edge lightpath instead of a lightpath with two edges if no other lightpaths are established.

There are similar examples in which SFR has only a unique cheapest routing lightpath which is obviously inferior to a more costly lightpath when durations of the calls are given.

The following variant of SFR, called *Anticipating* SFR (ASFR), aims at taking into account to which extent a possible routing choice blocks some lightpath during its whole existence. Again, let (G,Λ,W) be an optical network. Furthermore, let S be the current network status at arrival of a connection request σ_j whose start and expiration times are $t_j^{\mathtt{start}}$ and $t_j^{\mathtt{stop}}$.

As SFR before, ASFR aspires to measure for any two distinct nodes $s,t \in V$ the availability of edge-disjoint $[s,t]$-lightpaths in each wavelength $\lambda \in \Lambda$. Since this measure now also depends on $t_j^{\mathtt{start}}$ and $t_j^{\mathtt{stop}}$, it is denoted by $d_S^{(j)}(s,t,\lambda)$. It considers

those lightpaths which are newly blocked as well as those which would become available again before $t_j^{\mathtt{stop}}$ if no other connections were established. The latter lightpaths are taken into account according to their fraction of availability during $(t_j^{\mathtt{start}}, t_j^{\mathtt{stop}})$. To this end, a modified instance of the *maximum flow problem* is defined in which fractional capacities are assigned to those arcs on whose corresponding edges in G a considered wavelength is currently not available at $t_j^{\mathtt{start}}$ but would become available again before $t_j^{\mathtt{stop}}$. More precisely, for each edge $e \in E$ and each wavelength $\lambda \in \Lambda$ currently utilized on e, let $t^{\mathtt{free}}(e,\lambda)$ be the time when λ will become available again on e. Given a request σ_j with start and stop time $t_j^{\mathtt{start}}$ and $t_j^{\mathtt{stop}}$, respecitvely, we define $d_S^{(j)}(s,t,\lambda)$ as the maximum value of an (s,t)-flow in the digraph $D = (V,A)$ with arc capacities

$$\kappa_{S,j}^{(\lambda)}(a) := \begin{cases} 1, & \text{if } \lambda \text{ is available on } e(a) \text{ in status } S, \\ \max\{0, 1 - \frac{t^{\mathtt{free}}(e(a),\lambda) - t_j^{\mathtt{start}}}{t_j^{\mathtt{stop}} - t_j^{\mathtt{start}}}\}, & \text{if } \lambda \text{ is currently utilized on arc } e(a), \\ 0, & \text{if } \lambda \notin W(e(a)), \end{cases}$$

where the set of arcs A is defined as in 3.1, and for each arc $(u,v) \in A$, the corresponding edge in G is again denoted by $e((u,v)) := uv \in E$. Obviously,

$$0 \leq \kappa_{S,j}^{(\lambda)}(a) \leq 1$$

for each arc $a \in A$. Furthermore, given an arc $a \in A$ such that a wavelength $\lambda \in W(e(a))$ is currently utilized on $e(a)$, its capacity $\kappa_{S,j}^{(\lambda)}(a)$ is decreasing for increasing $t^{\mathtt{free}}(a,\lambda)$. Eventually, if $t^{\mathtt{free}}(e(a),\lambda) \geq t_j^{\mathtt{stop}}$, the capacity is zero. The idea is that the capacity of a is the greater, the longer λ is available on $e(a)$ in the interval $(t_j^{\mathtt{start}}, t_j^{\mathtt{stop}})$ unless further connections were realized.

Compared to SFR, note that $d_S^{(j)}(s,t,\lambda) \geq d_S(s,t,\lambda)$ since $\kappa_{S,j}^{(\lambda)}(a) \geq \kappa_S^{(\lambda)}(a)$ for each arc $a \in A$. As for SFR, we define with respect to each wavelength $\lambda \in \Lambda$ the current network fitness by

$$d_S^{(j)}(\lambda) := \sum_{\substack{s,t \in V: \\ s \neq t}} d_S^{(j)}(s,t,\lambda).$$

Hence, the total fitness function of ASFR is defined as

$$fit_{\mathsf{ASFR}}(S,j) := \sum_{\lambda \in \Lambda} d_S^{(j)}(\lambda)$$

yielding for each available routing lightpath (p,λ) the cost

$$\begin{aligned} c_{\mathsf{ASFR}}(S,j,(p,\lambda)) &= fit_{\mathsf{ASFR}}(S,j) - fit_{\mathsf{ASFR}}(S+(p,\lambda),j) \\ &= d_S^{(j)}(\lambda) - d_{S+(p,\lambda))}^{(j)}(\lambda). \end{aligned}$$

This cost function reflects that the routing decision depends on how many lightpaths become blocked at $t_j^{\mathtt{start}}$ and how much lightpaths are prevented from becoming available again later on.

Similar to SFR(T), we also consider a weighted version of ASFR, called ASFR(T), which takes into account the different average traffic demands for different connections. That is, we redefine $d_S(\lambda)$ for the network status S and a wavelength $\lambda \in \Lambda$ as:

$$\bar{d}_S^{(j)}(\lambda) := \sum_{\substack{s,t \in V: \\ s \neq t}} \text{demand}(s,t) \cdot d_S^{(j)}(s,t,\lambda).$$

where demand(s,t) denotes the average arrival frequency of calls which require connections between the nodes $s,t \in V$ in any period of time. The cost function of ASFR(T) is analogously defined to that of ASFR, but in terms of the values $\bar{d}_S^{(j)}(\lambda)$.

Let us look at the running time of ASFR and ASFR(T). Denote again by $n := |V|$ the total number of nodes in G. In order to determine the cost of an available routing lightpath, just as for SFR, one way is to solve $n \cdot (n-1)$ instances of the *maximum flow problem* as defined above. However, due to the arc capacities which are no longer either 0 or 1, solving each instance takes more time than in the integer case, namely it cannot be done in $O(n^2)$. Hence, we can take advantage in using the Gomory-Hu algorithm here, and obtain a total running time of $O(n^4)$.

Finally, let us reconsider Example 3.6 and see what decisions ASFR would make. For the first call σ_1, ASFR and SFR compute the same costs, since the network is still empty. Recall that $t_1^{\texttt{stop}} = t_2^{\texttt{start}} + \varepsilon$ and $t_i^{\texttt{stop}} - t_i^{\texttt{start}} = 1$ for $i = 1,2$. Upon arrival of σ_2, denote the current network status by S. The capacity of the arcs $(1,2)$ and $(2,1)$ in the corresponding digraph with respect to λ_1 now is

$$\kappa_{S,2}^{(\lambda_1)}((1,2)) = \kappa_{S,2}^{(\lambda_1)}((2,1)) = 1 - \frac{t_1^{\texttt{stop}} - t_2^{\texttt{start}}}{t_2^{\texttt{stop}} - t_2^{\texttt{start}}} = 1 - \varepsilon,$$

while all other arcs have capacity 1. At that time, we have $d_S^{(2)}(s,t,\lambda_1) = 2 - \varepsilon$ and $d_S^{(2)}(s,t,\lambda_2) = 2$ for any two distinct nodes $s,t \in V$. Denote by $L_1 := ((1,3,2),\lambda_1)$ and $L_2 := ((1,2),\lambda_2)$ the two lightpaths whose costs are identic for SFR. In Table 3.1 the cost defining values $d_{S+L_1}^{(2)}(s,t,\lambda_1)$, and $d_{S+L_2}^{(2)}(s,t,\lambda_2)$ are depicted for any pair of distinct nodes $s,t \in V$.

(s,t)	$d_{S+L_1}^{(2)}(s,t,\lambda_1)$	$d_{S+L_2}^{(2)}(s,t,\lambda_2)$
$(1,2)$	$1-\varepsilon$	1
$(1,3)$	0	1
$(2,3)$	0	1

Table 3.1: Computation of the cost c_{ASFR}.

Hence, the costs of the possible lightpaths are

$$c_{\text{ASFR}}(S,2,L_1) = (6 - 3 \cdot \varepsilon) - (1 - \varepsilon) = 5 - 2 \cdot \varepsilon$$

and

$$c_{\text{ASFR}}(S,2,L_2) = 6 - 3 = 3.$$

Obviously, the lightpath with minimum cost is L_2. Note also that a more general form of the cost of L_1 is

$$c_{\mathsf{ASFR}}(S,2,L_1) = 5 - 2 \cdot \frac{t_1^{\mathtt{stop}} - t_2^{\mathtt{start}}}{t_2^{\mathtt{stop}} - t_2^{\mathtt{start}}},$$

which shows that it increases with increasing duration of σ_2 and decreasing overlapping time of the two calls, as desired. On the other hand, the cost $c_{\mathsf{ASFR}}(S,2,L_2)$ remains constant.

3.2.4 The Algorithm NFR

In this section, we look at an approach which intends to define the fitness of an optical network as a number of currently available lightpaths such that any two are either edge-disjoint or use different wavelengths. That is, all counted lightpaths together satisfy the wavelength conflict constraint and can therefore be routed simultaneously in the current network status. By avoiding the problem of overlapping lightpaths, which occurred for the routing algorithms ALR and SFR, the resulting network fitness actually corresponds to a number lightpaths that could in principle be realized toghether. However, due to the uncertainty of the calls to come, the counted lightpaths can only correspond to one sample of future connection requests, which would very rarely occur. Again, the fitness value will be derived as the optimum solution of a flow problem.

Obviously, we cannot consider any pair of two distinct nodes in the network independently of the others, as done by SFR, since the fitness defining set is now allowed to contain in any wavelength edge-disjoint lightpaths having different end nodes. Hence, the current network fitness can not be obtained by computing single flows in the corresponding digraph. Instead, it must be represented by a set of flows between different pairs of source and sink nodes (*commodities*) in the optical network which compete for the network resources simultaneously. This leads to the idea to define the set of free edge-disjoint lightpaths via a *multicommodity flow*.

The input of the corresponding *maximum multicommodity flow problem* is a digraph $D = (V,A)$ with arc capacities $\kappa(a) \in \mathbb{R}_+$ for each $a \in A$ and a set $B \subseteq V \times V$ of pairs of nodes from V. The task is to find a vector $(x_b)_{b\in B}$, where x_b is an (s,t)-flow in D for each pair $b = (s,t) \in B$, such that in addition to the flow conservation constraints for each flow the *joint capacity constraints* $\sum_{b\in B} x_b(a) \leq \kappa(a)$ for all $a \in A$ hold, and the total flow value $\sum_{b\in B} \mathrm{val}(x_b)$ is maximum.

In our application, we will slightly modify the problem along with its parameters such that each feasible solution can be transformed into a set of lightpaths in the optical network. Based on the idea to compute the network fitness as the optimum value of a special *maximum multicommodity flow problem*, the proposed routing algorithm is called *network-flow-reduction* or NFR. As before, we are given an optical network (G,Λ,W) with network status S. In opposition to all algorithms presented so far, which used a decomposition of the problem in different wavelengths, NFR considers a coupled system. We first define for each wavelength $\lambda \in \Lambda$, one digraph as follows. Let V_λ be a copy of the set of nodes V, and denote for each node $v \in V$ the corresponding copied node in V_λ by v_λ and vice versa. Furthermore, define the set of arcs A_λ as

$$A_\lambda := \{(u_\lambda, v_\lambda) \in V_\lambda \times V_\lambda \mid uv \in E \text{ and } \lambda \in W(uv)\},$$

i.e., for each edge in G which is equipped with λ, set A_λ contains the two corresponding opposite directed arcs. For each $\lambda \in \Lambda$, let $D_\lambda := (V_\lambda, A_\lambda)$ be the corresponding digraph.

Moreover, similar to SFR, for each wavelength $\lambda \in \Lambda$, the arc capacities in D_λ with respect to network status S are defined by

$$\kappa_S((u_\lambda, v_\lambda)) := \begin{cases} 1, & \text{if } \lambda \text{ is available on edge } uv \text{ in status } S, \\ 0, & \text{otherwise.} \end{cases}$$

for each $(u_\lambda, v_\lambda) \in A_\lambda$.

Before we discuss the selection of the set of commodities B, let us define the structure of flows. Although we consider one joint flow system, for each $b = (s,t) \in B$, there is one (s,t)-flow $x_{b,\lambda}$ for each wavelength $\lambda \in \Lambda$. Since we would like to obtain undirected lightpaths but solve a directed flow problem, we require that for each flow also the corresponding so-called *backward* flow exists. That is, for each $b = (s,t) \in B$ and each wavelength $\lambda \in \Lambda$, a second (t,s)-flow $y_{b,\lambda}$ in D_λ is defined which must satisfy the condition

$$x_{b,\lambda}((u_\lambda, v_\lambda)) = y_{b,\lambda}((v_\lambda, u_\lambda)) \quad \text{for each arc } (u_\lambda, v_\lambda) \in A_\lambda. \tag{3.2}$$

In doing so, it is ensured that no two different flows share the capacities of two arcs $(u_\lambda, v_\lambda), (v_\lambda, u_\lambda) \in A_\lambda$, as long as the value of all flows on each arc is 0 or 1. Since for each backward flow the flow conservation constraints are obviously satisfied by the corresponding constraints for the *forward* flow, the backward flow must only be taken into account for the joint capacity constraints, i.e.,

$$\sum_{b \in B} x_{b,\lambda}(a) + y_{b,\lambda}(a) \leq \kappa_S(a) \quad \text{for each } \lambda \in \Lambda, a \in A_\lambda.$$

However, since the backward flow is defined in terms of the original one (3.2), the corresponding variables can be replaced. Together with the other flow constraints the following integer program is obtained:

$$\max \sum_{\lambda \in \Lambda} \sum_{b=(s,t) \in B} \left(\sum_{a \in \delta^+(s_\lambda)} x_{b,\lambda}(a) - \sum_{a \in \delta^-(s_\lambda)} x_{b,\lambda}(a) \right)$$

$$\sum_{a \in \delta^-(v_\lambda)} x_{b,\lambda}(a) = \sum_{a \in \delta^+(v_\lambda)} x_{b,\lambda}(a) \qquad \text{for each } \lambda \in \Lambda, b = (s,t) \in B, v_\lambda \in V_\lambda \setminus \{s_\lambda, t_\lambda\}$$

$$\sum_{b \in B} x_{b,\lambda}((u_\lambda, v_\lambda)) + x_{b,\lambda}((v_\lambda, u_\lambda)) \leq \kappa_S((u_\lambda, v_\lambda)) \qquad \text{for each } \lambda \in \Lambda, (u_\lambda, v_\lambda) \in A_\lambda$$

$$x_{b,\lambda}(a) \in \{0,1\} \qquad \text{for each } \lambda \in \Lambda, b \in B, a \in A_\lambda$$

Since this integer program depends on the network status S, we will denote it by $\mathtt{IP}_S$. Notice that $\mathtt{IP}_S$ indeed represents a special *maximum integral multicommodity flow problem*, since the objective function is the sum of the values of all flows $x_{b,\lambda}$ for $b \in B$ and $\lambda \in \Lambda$, and along with the joint capacity constraints also flow conservation constraints are established. Hence, the only modifications compared to the standard

problem are the implicit backward flows and the consideration of flows in $|\Lambda|$ digraphs. The number of binary variables of $\mathtt{IP}_S$ is

$$b \cdot \sum_{\lambda \in \Lambda} |A_\lambda| \leq b \cdot |\Lambda| \cdot 2|E|.$$

In the following, we will look at the relationship between solutions of $\mathtt{IP}_S$ and lightpaths in the optical network. Let L be a set of currently available lightpaths in the optical network any two of which are edge-disjoint, or have different wavelengths and whose end nodes correspond to a pair of nodes in B. Define for each $\lambda \in \Lambda$, $b = (s,t) \in B$, and each $(u_\lambda, v_\lambda) \in A_\lambda$ the value $x_{b,\lambda}((u_\lambda, v_\lambda))$ by

$$x_{b,\lambda}((u_\lambda, v_\lambda)) := \begin{cases} 1, & \text{if there is an } [s,t]\text{-lightpath } (p,\lambda) \in L \text{ with } uv \in E(p), \\ 0, & \text{otherwise.} \end{cases}$$

Then, this assignment of variables is obviously a feasible solution of $\mathtt{IP}_S$. The other way round, given a feasible solution of $\mathtt{IP}_S$, we can derive a set of lightpaths in the optical network with the same properties as those in L by decomposing for each $b \in B$ and each $\lambda \in \Lambda$ the flow $x_{b,\lambda}$ into its paths. Furthermore, for both transformations, the cardinality of the set of lightpaths equals the objective value of $\mathtt{IP}_S$ of the corresponding solution. Hence, we have proven the following.

Theorem 3.7. *Each optimum solution of the* $\mathtt{IP}_S$ *yields a set L of currently available lightpaths with the following properties: Any two are edge-disjoint, or have different wavelengths, their end nodes correspond to any commodity in B, and its cardinality* $|L|$ *is maximum, i.e., there is no such set that contains more elements.*

Proof. Assume there is a set L' of lightpaths which together satisfy the wavelength conflict constraint such that $|L'| > |L|$. Then, the objective value of $\mathtt{IP}_S$ of the feasible solution which corresponds to L' is greater than the optimum value. This yields a contradiction. □

By Theorem 3.7, the optimum value of $\mathtt{IP}_S$ represents an appropriate fitness function for the algorithm NFR. Therefore, the cost of a routing lightpath (p,λ) is defined as the difference of the optimum value of $\mathtt{IP}_S$ and the optimum value of $\mathtt{IP}_{S+(p,\lambda)}$. That is, NFR selects such a lightpath which maintains as many available lightpaths as can be routed simultaneously and whose end nodes correspond to a pair in B.

Unfortunately, due to the integrality constraints, the problem of solving the $\mathtt{IP}_S$ is NP-hard [GJ79]. Hence, we will work with its LP-relaxation which allows

$$x_{b,\lambda}(a) \geq 0 \quad \text{for each } \lambda \in \Lambda, b \in B, a \in A_\lambda.$$

Then, the resulting linear program can be solved efficiently in theory. Good in practice are standard LP-solver such as *CPLEX*. Note that the resulting solution cannot necessarily be transformed into a set of lightpaths. Nevertheless, using the relaxation which ignores flow integrality constraints might achieve good routing decisions if the solutions do not contain too many fractional flow values. Since we are only approximating network fitness the LP-relaxation might still yield useful information for the fitness function.

Selection of Commodities

Finally, we focus on the definition of the the set of commodities B which defines the source and sink nodes of the possible flows, or the end nodes of the corresponding allowed lightpaths, respectively. First, note that for any two distinct nodes $s,t \in V$ it suffices that either (s,t) or (t,s) is contained in B since due to the existence of forward and backward flows, flow is always established in both directions. In the following, let $<_V$ be any order on the set of nodes V. Choosing for B the set

$$B_1 := \{(u,v) \in V \times V \mid u <_V v\},$$

containing all distinct pairs of nodes in V will always yield an optimum value that is equal to the sum of all current edge availabilities. This is due to the fact that all free single-edge lightpaths in each wavelength are allowed, clearly pairwise edge-disjoint, and their number must be maximum. Hence, the cost of a routing lightpath is minimum if and only if it is a shortest lightpath, i.e., by this definition of B the algorithm NFR is equivalent to the greedy-type algorithms with total wavelength search up to special tie-breaking. Notice that it is a very interesting result that this very advanced fitness model of NFR, which was developed in order to overcome the drawbacks of ALR and SFR, again corresponds to the most simple idea for lightpath routing. However, this is only the case if the end nodes of given calls can be arbitrary and if all end nodes appear equally likely. In this case, the approach of NFR yields nothing new. Obviously, the set of single-edge lightpaths is also valid if B contains for each edge in G one pair of its end nodes, i.e., B equals

$$B_2 := \{(u,v) \in V \times V \mid uv \in E \text{ and } u <_V v\}.$$

Hence, the set of commodities B_2 has to be restricted such that the routing decisions of NFR differ from those of EXHAUSTIVE. Of course, in the case of a non-uniform traffic distribution, it seems reasonable to reduce B to the connection pairs with positive traffic demand, or even to such pairs whose traffic demand is not too small. Then, we can choose B as

$$B_3(\delta) := \{(u,v) \in V \times V \mid \text{demand}(u,v) > \delta \text{ and } u < v\},$$

where $\delta \geq 0$ is some threshold value for the demand. If the average demand between different nodes in the optical network varies, one would like to bound for two distinct nodes $u,v \in V$ the number of $[u,v]$-lightpaths in the computed solution from above if their traffic demand is relatively small. Furthermore, if there is a known upper bound on the number of lightpaths with the same end nodes that can ever be requested simultaneously, the solution should never contain more of them than this upper bound. However, choosing a smaller upper bound may also be reasonable if there is less network capacity left. To this end, we can add constraints of the form

$$\sum_{\lambda \in \Lambda} \left(\sum_{a \in \delta^+(s_\lambda)} x_{b,\lambda}(a) - \sum_{a \in \delta^-(s_\lambda)} x_{b,\lambda}(a) \right) \leq u(b) \quad \text{for each } b = (s,t) \in B$$

to the integer program which bound the total flow for each commodity $b = (s,t) \in B$ or the number of $[s,t]$-lightpaths in the corresponding set of lightpaths, respectively, by some value $u(b)$.

However, it is not easy to select upper bounds in a helping way. If they are too restrictive, the set of computed lightpaths, denote it by L, will not require the whole remaining network capacity. Hence, for many or even all routing lightpaths (p,λ), the network fitness after their realization will not decrease since all lightpaths in L are still available, which yields the cost $c_{\mathsf{NFR}}((p,\lambda)) = 0$. This is obviously bad since a lot of different routing lightpaths may be chosen, even those which are very long and require much capacity. On the contrary, if the flow value bounding constraints are not restrictive enough, there may be many commodities in B whose total flow value is large, and others having less or no corresponding flow at all. The latter will usually be those commodities $b = (s,t) \in B$ for which the distance between nodes s and t is large. Nevertheless, corresponding calls may often arrive, and therefore, it need to be taken into account whether free routing lightpaths for these calls exist or not.

Obviously, the upper bounds must depend on the current network status. The more network capacity is still available, the larger the bounds should be in order to avoid the first effect described above. And for increasing network load the bounds should decrease, otherwise the second problem will occur.

However, using these an type of these boundary constraints in the LP-relaxation requires much more computational effort. Therefore the version of NFR applied for the experimental studies in Chapter 5 does not use such additional constraints. We selected $B_3(0)$ to define the set of commodities, i.e., the pairs of nodes with positive average demands.

3.2.5 Reduction to k Shortest Lightpaths Routings

As mentioned befor, the number of all currently available lightpaths connecting the start node and the end node of a given connection request may be exponential in the number of edges of the graph. Hence, especially for large networks, the presented network fitness algorithms may require too much computational effort. Then we can restrict the set of evaluated lightpaths by taking into account only the k shortest lightpaths which are currently available in each wavelength for some constant $k \in \mathbb{N}$. Of course, those lightpaths has to be computed at arrival of each connection request depending on the current network status. This can be done by the algorithm which is presented in Chapter 4. Since the problem of finding the k-shortest paths in a graph or digraph can be solved efficiently, the reduction to k shortest lightpaths routings yields polynomial running times if the used network fitness algorithm can compute the cost for one lightpath in polynomial time. Note that this applies to SFR, ASFR, and SALR.

Notice that this proceeding need not to be disadvantageous. First, there are usually a lot of available long lightpaths which will never be chosen since they require very much network capacity, and thus make no sense to be considered at all. Therefore, only lightpaths which are short or of medium length provide reasonable routings. This tendency can also be observed by our routing algorithms which prefer shorter lightpaths. Hence, the routing decisions will not differ much if only a subset of shorter lightpaths is considered instead of all free lightpaths.

Chapter 4

An Algorithm for Finding the k Shortest Paths

We have seen that it is neither beneficial nor computationally efficient for the **DSCA**-algorithms presented in Section 3.2 to consider all possible routing choices. Therefore, especially when dealing with large networks, the set of evaluated lightpaths should be restricted to those more likely to be chosen, i.e., short lightpaths. To this end, we consider the problem of finding the k shortest paths between a pair of nodes, which is introduced in Section 4.1. In the following of this chapter, we present a revised full description of an algorithm that serves this purpose (Sections 4.2 and Section 4.3). It has been proposed by Martins, Pascoal, and Santos [MPS99]. Furthermore, we state a new proof of the correctness of the algorithms in Section 4.4 since the proof given by the authors turned out to be deficient. Finally, we answered the previous open question about the running time of the algorithm to be polynomial in the negative.

4.1 Introduction

An instance of the *k Shortest Walks Problem* (k-**SW**) is given by a digraph $D = (V,A)$ that is not necessarily simple, a length function $c : A \to \mathbb{R}$ such that no negative cycle exists, two nodes $s,t \in V$, and a number $k \in \mathbb{N}$. The task is to find a set of k (s,t)-walks $P = \{p_1, \ldots, p_k\}$ of total minimum length, or all such walks if fewer than k exist. We speak about the *k Shortest Paths Problem* (k-**SP**) if the found walks shall be paths, i.e., they must not contain cycles[1]. As usual, let $n := |V|$ be the number of nodes and $m := |A|$ be the number of arcs in the digraph.

The special case of $k = 1$ is the well-known *Shortest Path Problem*. Note that there is always a shortest walk which is a path if the digraph does not contain negative cycles. The algorithm of Moore, Bellman, and Ford (cf. [KV00, Chapter 7]) yields the

[1] In the literature, the two problems k-**SW** and k-**SP** are usually referred to as the *k shortest paths problem* and the *k shortest simple (or loopless) paths problem*. That is, our term walk corresponds to a path, and out term path corresponds to a simple or loopless path.

best known running time of $O(nm)$ for solving this problem. If all arc lengths are non-negative, the complexity can be reduced to $O(m+n\log n)$ using the implementation of Dijkstra's algorithm with Fibonacci heaps [KV00, Chapter 7]. Both algorithms do not only return a shortest (s,t)-path but a single source shortest path tree (Dijkstra's algorithm may terminate previously after the shortest (s,t)-path has been found). The root of this tree is s, and it yields for each node $v \in V$ that is reachable from s a shortest (s,v)-path. While k-**SW** can be solved efficiently in $O(m+n\log n+k)$ time by the algorithm of Eppstein [Epp98], k-**SP** seems to be more difficult. For directed graphs, the best known theoretical algorithms for this problem run in $O(kn(m+n\log n))$ time, see Yen [Yen71, Yen72] and Lawler [Law72]. For undirected graphs, Katoh, Ibaraki, and Mine [KIM82] improved the algorithm of Yen to $O(k(m+n\log n))$ running time.

In the following, we present another algorithm for k-**SP** proposed by Martins, Pascoal, and Santos [MPS99], called MPS in the sequel. Since their proof of correctness is based on simplifying assumptions which we can show to be false by a simple example, we state a more detailed proof here. Although MPS seems to outperform the above mentioned algorithms significantly in practice, as shown in [MPS99], its theoretical running time has previously been open. In the last section of this chapter, we show that its worst-case running time is exponential in the size of the digraph.

In our application, MPS is useful as a subroutine for the **DSCA**-algorithms which compute costs of lightpaths and realize a cheapest one. For larger networks it is computationally too expensive to evaluate all currently available lightpaths connecting the start node and the end node of the connection request. Since our algorithms tend to prefer short lightpaths and do rarely choose any of the currently available long lightpaths, their decisions should not differ much if only a subset of shorter lightpaths is considered instead of all free lightpaths.

MPS works in directed graphs but can also be applied to undirected graphs with non-negative edge lengths by the usual transformation into a digraph: Each edge is replaced by two opposite directed arcs. Since all edge lengths are non-negative, the resulting digraph cannot contain a negative cycle. Note that in our application, we have unit edge lengths, since the length of a lightpath is defined as its number of edges.

4.2 Preliminaries

Given an instance of k-**SP**, we may assume that there is a path from s to v and a path from v to t in D for each node $v \in V$. Otherwise, a node v without this property can be removed from D since it cannot be contained in any (s,t)-path. In the following, we denote by $V(p)$ and $A(p)$ the node and arc set of a given walk p, respectively. Furthermore, the concatenation of two paths p and q is denoted by pq, and for an (s,t)-walk p and each node $v \in V(p)$ that is reached on p before any node is reached twice, we denote by p_{sv} the starting subpath from s to v.

In an initial phase, MPS computes a *single destination shortest path tree* T with destination t. For each node $v \in V$, the path from v to t defined by this tree, denoted by p^v, is a shortest path. Note that such a tree corresponds to a single source shortest

path tree with source t in the digraph with the reversed arcs of D. Hence, it can be computed as described in the previous paragraph. In the special case that arcs have unit lengths, as in our application, the tree T can be determined more easily. Using breadth-first search, the computation takes only $O(n+m)$ time, cf. [KV00, Chapter 2].

Let $c(p) := \sum_{a \in A(p)} c(a)$ denote the length of walk p. Moreover, let $d(v) := c(p^v)$ be the distance from v to t for each node $v \in V$. MPS uses the following modified length function which was firstly applied in this context by Eppstein [Epp98]:

$$\bar{c}(a) := c(a) + d(head(a)) - d(tail(a)) \text{ for each arc } a \in A$$

Note that for each arc $(u,v) \in A$, the number $\bar{c}((u,v))$ is the difference between the lengths of the two (u,t)-walks $(u,(u,v),v)p^v$ and p^u. That is, $\bar{c}((u,v))$ measures the length of the detour which results from taking the walk along (u,v) followed by a shortest path from v to t, instead of taking directly a shortest path from u to t. As above, we define for each walk p its modified length $\bar{c}(p) := \sum_{a \in A(p)} \bar{c}(a)$.

Lemma 4.1. *For the length function $\bar{c}$, the following holds.*

a) $\bar{c}(p) = c(p) - d(s)$ *for each (s,t)-walk p.*

b) $\bar{c}((u,v)) \geq 0$ *for each arc $(u,v) \in A$.*

c) $\bar{c}((u,v)) = 0$ *for each arc $(u,v) \in T$.*

d) $\bar{c}(p^v) = 0$ *for each node $v \in V$.*

e) For two (s,t)-walks p and q it holds that $c(p) \leq c(q)$ if and only if $\bar{c}(p) \leq \bar{c}(q)$.

Proof. a) By using $d(t) = 0$, we obtain:

$$\begin{aligned}
\bar{c}(p) &= \sum_{a \in A(p)} \bar{c}(a) \\
&= \sum_{a \in A(p)} c(a) + d(head(a)) - d(tail(a)) \\
&= c(p) + \sum_{a \in A(p)} d(head(a)) - d(tail(a)) \\
&= c(p) + d(t) - d(s) \\
&= c(p) - d(s).
\end{aligned}$$

b) Clearly, $d(u) \leq c((u,v)) + d(v)$. Hence, $\bar{c}((u,v)) = c((u,v)) + d(v) - d(u) \geq 0$.

c) Since $(u,v) \in T$, the shortest (u,t)-path p^u is of the form $p^u := (u,(u,v),v)p^v$. Therefore, its length equals $c(p^u) = c((u,v)) + d(v)$. On the other hand, we have $c(p^u) = d(u)$. Hence, $\bar{c}((u,v)) = c((u,v)) + d(v) - d(u) = 0$.

d) Since $a \in T$ for each arc $a \in A(p^v)$, d) follows directly from c).

e) By a) we have $c(p) \leq c(q)$ if and only if $c(p) - d(s) \leq c(q) - d(s)$, which is equivalent to $\bar{c}(p) \leq \bar{c}(q)$.

□

Notice that Lemma 4.1 e) reveals that both length models are equivalent, in particular, an (s,t)-path is a shortest path with respect to c if and only if it is a shortest path with respect to $\bar{c}$. Hence, we may omit the specification of the length function.

4.3 The Algorithm

Since the algorithm MPS aims at solving k-**SP**, its input consists of a not necessarily simple digraph $D = (V,A)$, a length function $c : A \to \mathbb{R}$ such that no negative cycle in D exists, two nodes $s,t \in V$, and a number $k \in \mathbb{N}$. As mentioned before, it is assumed that for each node $v \in V$, v is reachable from s and t is reachable from v. Moreover, it is required that the digraph D is stored by adjacency lists. The output of MPS is a sequence $(p_1,\ldots,p_k)$ that contains the k shortest (s,t)-paths in D or all such paths if there are fewer than k, which indeed solves k-**SP**. Furthermore, the lengths of paths in this sequence is increasing, i.e., $c(p_1) \leq \ldots \leq c(p_k)$.

The strategy of MPS is as follows. While running the algorithm, it maintains a priority queue X that stores found walks. In each step of the algorithm, a walk p which is shortest among all walks in X is extracted, and new (s,t)-walks which share a subpath with p are determined and inserted into X. To this end, each walk p is assigned a special node d_p called *the deviation node* of the walk. By construction, the starting subpath p_{sd_p} of p to its deviation node will always exist. All walks newly inserted after the extraction of p share at least this subpath p_{sd_p}, but diverge afterwards as depicted in Figure 4.1.

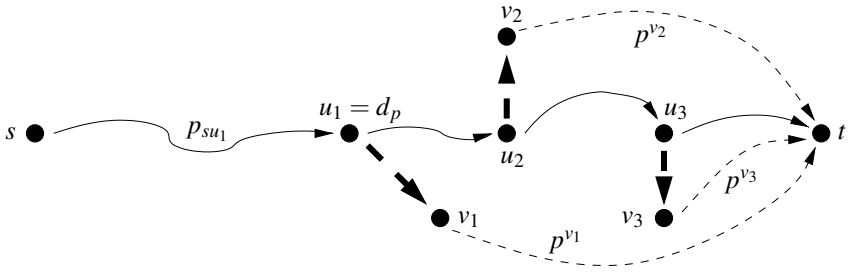

Figure 4.1: Shape of new walks which emerge by considering walk p (solid arrows). The deviation node of p is denoted by u_1, and the dashed thick arcs indicate where the new walks diverge from p. New walks are $p_{su_i}(u_i,(u_i,v_i),v_i)p^{v_i}$, for $i = 1,2,3$. Note that all walks are actually paths in this picture.

We wish to assert that the extracted walk p is not only shortest among the walks currently in X but also among all (s,t)-walks which have not yet been inserted into X. That is, only previously extracted walks may be shorter than p. Therefore, walks are extracted from X in order of increasing length. For that purpose, the algorithm makes use of the following sorting of arcs in the adjacency lists. For each node $v \in V \setminus \{t\}$, all arcs with tail node v are ordered by increasing modified length $\bar{c}$. Furthermore, the first arc in each order is contained in the single destination shortest path tree T. Hence, for the arcs $a_1^{(v)}, a_2^{(v)}, \ldots, a_{deg_v}^{(v)} \in \delta^+(v)$, it holds that

$$a_1^{(v)} \in T \text{ and } \bar{c}(a_1^{(v)}) \leq \bar{c}(a_2^{(v)}) \leq \ldots \leq \bar{c}(a_{deg_v}^{(v)}), \tag{4.1}$$

where $deg_v := |\delta^+(v)|$ is the out-degree of v. Since $\bar{c}(a) \geq 0$ for each arc $a \in A$ and $\bar{c}(a) = 0$ for each tree arc $a \in T$ by Lemma 4.1 b) and c), the tree arcs can indeed be the first elements in each order, respectively. For this reason, such orders exist.

New walks are constructed as follows. For walk $p := (s,(s,v_1),v_1,\ldots,v_h,(v_h,t),t)$ which has been extracted from X, let $1 \leq l \leq h$ be the maximum index such that the walk $(s,(s,v_1),v_1,\ldots,v_{l-1},(v_{l-1},v_l),v_l)$ is a path. The algorithm tries to construct one new walk for each node u located on p_{sv_l} between d_p and v_l (including the nodes d_p and v_l). Let $a^{(u)}_{i(p)} \in \delta^+(u) \cap A(p)$ be that arc of walk p whose tail node is u. MPS looks for the minimum index $j > i(p)$ such that $p_{su}(u,a^{(u)}_j,v_j)$ is a path, where $v_j := head(a^{(u)}_j)$. If such an index j exists, the concatenated walk $q := p_{su}(u,a^{(u)}_j,v_j)p^{v_j}$ with deviation node $d_q := u$ is inserted into X. Note that walk q still has a starting subpath from s to the immediate successor node of its deviation node. The construction of new walks, as described above, is depicted in Figure 4.2.

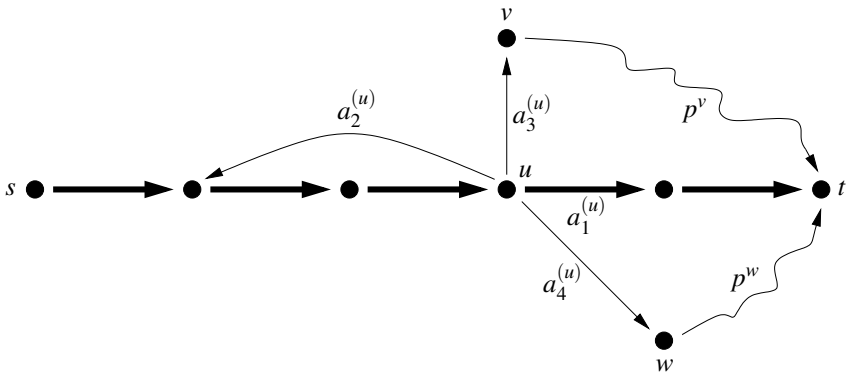

Figure 4.2: Construction of the next walk which diverges at node u from walk p (thick arcs). The arc of p which leaves u is $a^{(u)}_1$. Since p_{su} forms a cycle together with the arc $a^{(u)}_2$, the next walk which is inserted into X is $p_{su}(u,a^{(u)}_3,v)p^v$. Extracting this walk in turn in a subsequent step of the algorithm yields walk $p_{su}(u,a^{(u)}_4,w)p^w$. Actually, the three considered walks are again paths.

As mentioned above, we claim that MPS extracts walks from X in order of increasing lengths. Moreover, we will show that the set of constructed (s,t)-walks contains especially all paths from s to t. Hence, the output sequence of k shortest (s,t)-paths can be obtained as follows: Once a path is extracted from X, it becomes the next element in the sequence. After k paths have been extracted, the algorithm terminates. The code of MPS is shown in Algorithm 5.

Example 4.2. For an illustration of the algorithm, consider the digraph D with arc lengths c shown in Figure 4.3(a). The single destination shortest path tree T (indicated by thick arrows), the distances from each node to t, and the modified arc lengths $\bar{c}$ are depicted in Figure 4.3(b). In the given example, the order of the adjacency lists is

Input : A digraph (V,A); a length function $c : A \to \mathbb{R}$ such that no negative cycle exists; two nodes $s,t \in V$; a number $k \in \mathbb{N}$.

Output : A sequence $(p_1,\ldots,p_k)$, consisting of k shortest (s,t)-paths, or all paths from s to t if their number is less than k, respectively, with $c(p_i) \leq c(p_j)$ for $1 \leq i \leq j \leq k$.

Determine a single destination shortest path tree T with destination t;
Compute $\bar{c}(a) := c(a) + d(head(a)) - d(tail(a))$ for each $a \in A$;
For each node $v \in V \setminus \{t\}$, define $deg_v := |\delta^+(v)|$ to be the out-degree of v;
For each node $v \in V \setminus \{t\}$, let $a_1^{(v)}, a_2^{(v)}, \ldots, a_{deg_v}^{(v)} \in \delta^+(v)$ satisfy (4.1);
Let $X := \{p^s\}$ and define the deviation node of path p^s to be $d_{p^s} := s$;
Let $l := 0$; $\{l$ *indicates how many of the* k *paths have already been found*$\}$
1 **while** $X \neq \emptyset$ *and* $l < k$ **do**
  Choose $p \in X$ such that $\bar{c}(p) \leq \bar{c}(q)$ for each $q \in X$;
  $X := X \setminus p$;
  if p *is a path* **then**
    $l := l + 1$;
    $p_l := p$; $\{$*Next path in the sequence is found*$\}$
  end
  Let $u := d_p$ be the deviation node of walk p;
  Let $p^{start} := p_{su}$ be the subpath of p from s to u;
2 **repeat**
    $\{$*Detect new walk that diverges from walk* p *at node* u, *cf. Figure 4.2*$\}$
    Let $a_{i(p)}^{(u)} \in \delta^+(u) \cap A(p)$ be that arc of p whose tail node is u;
    if $i(p) < deg_u$ **then**
      Let $j := i(p) + 1$;
      while $j \leq deg_u$ *and* $a_j^{(u)}$ *forms a cycle with* p^{start} **do**
        $j := j + 1$;
      end
      if $j \leq deg_u$ **then**
        Let $v := head(a_j^{(u)})$;
        Let $q := p^{start}(u, a_j^{(u)}, v)p^v$; $\{$*New walk found*$\}$
        $d_q := u$;
        $X := X \cup \{q\}$;
      end
    end
    Let $v := head(a_{i(p)}^{(u)})$; $\{$*The next node on* p *is set to* u*$\}$
    $p^{start} := p^{start}(u, a_{i(p)}^{(u)}, v)$;
    $u := v$;
  until p^{start} *constains a cycle or* $u = t$;
end

Algorithm 5: MPS

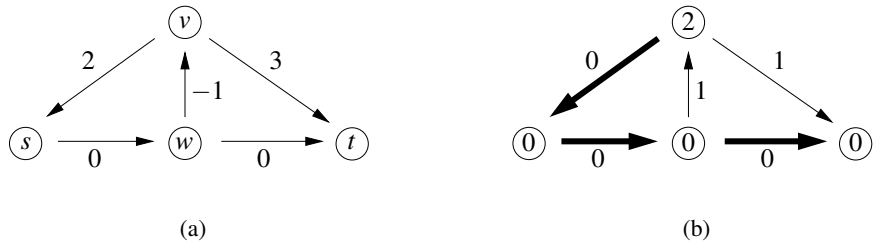

Figure 4.3: (a) Example digraph D with arc lengths c; (b) Shortest path tree (thick arcs), distances to t, and modified arc lengths $\bar{c}$ of D.

unique since all arcs leaving the same node have different lengths with respect to $\bar{c}$.

$$\begin{aligned} s:&\ \ a_1^{(s)} = (s,w) \\ w:&\ \ a_1^{(w)} = (w,t), a_2^{(w)} = (w,v) \\ v:&\ \ a_1^{(v)} = (v,s), a_2^{(v)} = (v,t) \end{aligned}$$

In order to simplify the notation, we will denote a walk by the sequence of its nodes, which is possible since D is simple. The shortest (s,t)-path given by T is $p^s = (s,w,t)$ with deviation node $d_{p^s} = s$. In the first iteration of the while-loop of MPS (Step 1), this path is extracted from X, and we have $p = p_1 = (s,w,t)$. The algorithm now tries to find new (s,t)-walks by processing the nodes of p (repeat-loop, Step 2). The first one of those is its deviation node $d_p = s$. But, except from (s,w), there are no further arcs emanating from s. Hence, there is no new walk which diverges from p at s. The next node which is processed by the algorithm is w. Besides arc (w,t) of p, (w,v) is the only other arc with tail node w. Furthermore, (w,v) succeeds (w,t) in the sorting of $\delta^+(w)$ and does not form a cycle together with the path (s,w). Therefore, the next (s,t)-walk $q_1 := (s,w)(w,v)(v,s,w,t) = (s,w,v,s,w,t)$ with deviation node $d_{q_1} = w$ is inserted into X. Afterwards, the first iteration of the while-loop in Step 1 is over since the successor node of w in the path is t. In the next iteration, the extracted walk is $p = (s,w,v,s,w,t)$. Since it contains a cycle, p_2 is not found yet. Processing the deviation node $d_p = w$, no new walk is found. But a second arc (v,t) is leaving node v and succeeds (v,s) is the adjacency list of v. This yields the path $q_2 = (s,w,v)(v,t)(t) = (s,w,v,t)$ with deviation node $d_{q_2} = v$. Path q_2 is inserted into X. Thereafter, the second iteration of the while-loop stops since s follows v, whereby the resulting walk (s,w,v,s) is a cycle. By extracting the only remaining walk in X, we obtain the next path $p_2 = (s,w,v,t)$. No further walk is found by processing its deviation node v, and its successor node is already t. MPS terminates having found all (s,t)-paths in D.

4.4 Proof of Correctness

As mentioned above, the proof of correctness for the algorithm MPS in [MPS99] is based on simplifying assumptions. Before we give a new proof, we show that one substantial assumption made by the authors is wrong.

Their proof uses induction on k and starts as follows. Obviously, MPS is correct for $k = 1$. Assume that it also determines correctly a sequence of $k-1$ shortest (s,t)-paths $(p_1, \ldots, p_{k-1})$. Let p be one candidate path for p_k. For each path in the sequence, consider the subpath which starts in s and follows p as long as possible. Let p' be one path in the sequence whose starting subpath is longest, and let $u \in V$ be the node where p diverges from p'. Here the authors claim that u must be the deviation node d_p of p. However, this is not necessarily true as Example 4.2 shows for $k = 2$. We have $p = (s,w,v,t)$ and $p' = (s,w,t)$. But w is not the deviation node d_p since p is found by processing the node v of walk (s,w,v,s,w,t). Hence, the deviation node of p is $d_p = v$.

In the following, we state a new proof of correctness. It is structured into two parts. In the first part, it is shown in Lemma 4.3 and Lemma 4.4 that MPS extracts walks from X in order of increasing length. Afterwards, we prove that all (s,t)-walks which are constructed by the algorithm are of a special form. Furthermore, if k is chosen sufficiently large, MPS finds all of those walks which comprise especially the (s,t)-paths. These results are obtained by Lemma 4.5. Together, Lemma 4.4 and Lemma 4.5 imply that MPS indeed returns the k shortest paths in D in order of increasing lengths.

Lemma 4.3. *In an arbitrary iteration of the while-loop of Algorithm 5, let p be the walk currently extracted from X. Then each diverging walk q which is detected by processing the nodes of p is at least as long as p.*

Proof. Let $u := d_p$ be the deviation node of walk p. Using the notation as in the algorithm but omitting the superscript, let $a_{i(p)} \in \delta^+(u) \cap A(p)$ be that arc of walk p which leaves its deviation node, and let $v := head(a_{i(p)})$ be the head node of this arc. Hence, the walk p is of the form $p = p_{su}(u, a_{i(p)}, v)p^v$. Since $\bar{c}(p^{(v)}) = 0$ by Lemma 4.1 d), it has the modified length $\bar{c}(p) = \bar{c}(p_{su}) + \bar{c}(a_{i(p)})$. By construction, each walk q which is generated from p either diverges at u or at a successor node of u. In the first case, $q = p_{su}(u, a_j, w)p^w$, where $a_j \in \delta^+(u) \cap A(p)$ with $j > i(p)$ and $w := head(a_j)$. Therefore, its modified length is $\bar{c}(q) = \bar{c}(p_{su}) + \bar{c}(a_j)$. Since $\bar{c}(a_{i(p)}) \leq \bar{c}(a_j)$, we have that $\bar{c}(p) \leq \bar{c}(q)$. In the second case, q shares with p the subpath $p_{su}(u, a_{i(p)}, v)$. By Lemma 4.1 b), $\bar{c}(a) \geq 0$ for each arc $a \in A$, and hence $\bar{c}(q) \geq \bar{c}(p_{su}) + \bar{c}(a_{i(p)}) = \bar{c}(p)$, yielding the claim for this case. □

Lemma 4.4. *At any time while performing Algorithm 5, let $r \in \mathbb{N}$ be the number of walks extracted from X so far, and for $l = 1, \ldots, r$, let q_l be the l-th extracted (s,t)-walk. Then, it holds for the lengths of these walks:*

$$c(q_l) \leq c(q_{l+1}) \text{ for } l = 1, \ldots, r-1.$$

That is, walks are extracted from X in order of increasing length.

Proof. For the proof we use induction on the number of extracted walks r.

$\underline{r = 1}$:
Nothing to show.

$\underline{r \to r+1}$:
We only have to prove that $c(q_r) \leq c(q_{r+1})$. Let X_r be the priority queue X directly

before q_r was extracted. At that time, q_r was a shortest walk among all walks contained in X_r, i.e., $c(q_r) \leq c(q)$ for each walk $q \in X_r$. Moreover, all (s,t)-walks which are constructed from q_r and inserted into X are at least as long as q_r itself, as shown by Lemma 4.3. Therefore, each walk which is contained in X when the next walk q_{r+1} is extracted is at least as long as q_r. □

Lemma 4.5. *Given a node $v \in V$, let p' be an arbitrary (s,v)-path that satisfies the following conditions:*

a) t is not an interior node of p', i.e., $t \notin V(p') \setminus \{v\}$.

b) If $p' \neq (s)$, the last arc of p' is not contained in the shortest path tree T.

If k is sufficiently large, MPS *determines the (s,t)-walk $p = p'p^v$ exactly once.*

Proof. Let $r \geq 0$ be the number of arcs in p', i.e., the (s,v)-path p' is of the form $p' = (s, b_1, v_1, \ldots, v_{r-1}, b_r, v)$, where $b_1, \ldots, b_r \in A$, $b_r \notin T$ and $v_1, \ldots, v_{r-1} \in V \setminus \{t\}$. By construction, when $p = p'p^v$ is inserted into X, its deviation node is set to $d_p = v_{r-1}$ if $r \geq 1$ ($v_0 := s$). We prove by induction on the number of arcs r of p' that the walk $p = p'p^v$ is determined exactly once by MPS.

$\underline{r = 0}$:
Obviously, since p' is the empty path, $p = p^s$. This path is determined when the shortest path tree T is constructed, and it is extracted from X in the first iteration of the while-loop. Afterwards, all walks in X contain at least one arc which is not in T. Hence, the shortest (s,t)-path p^s is not found again.

$\underline{r \to r+1}$:
Given the starting subpath $p' = (s, b_1, v_1, \ldots, v_r, b_{r+1}, v)$ of p, where $b_1, \ldots, b_{r+1} \in A$, $b_{r+1} \notin T$ and $v_1, \ldots, v_r \in V \setminus \{t\}$, we define the set $A(v_r)$ as follows:

$$A(v_r) := \{a \in \delta^+(v_r) \setminus T \mid (s, b_1, v_1, \ldots, b_r, v_r)(v_r, a, head(a)) \text{ is a path}\}.$$

It is now shown by an inner induction on $|A(v_r)|$ that for each arc $a \in A(v_r)$, the concatenated walk $(s, b_1, v_1, \ldots, b_r, v_r)(v_r, a, head(a))p^{head(a)}$ is determined exactly once. Since $b_{r+1} \in A(v_r)$, this proves the inductive step from r to $r+1$. Let $h := |A(v_r)|$ be the number of arcs in $A(v_r)$, and again omitting the superscripts, let $a_{i_1}, \ldots, a_{i_h}$ be their order with respect to the modified length $\bar{c}$, i.e., $i_1 < \ldots < i_h$ and $\bar{c}(a_{i_1}) \leq \ldots \leq \bar{c}(a_{i_h})$. Moreover, for $j = 1, \ldots, h$, let $q_j := (s, b_1, v_1, \ldots, b_r, v_r)(v_r, a_{i_j}, head(a_{i_j}))p^{head(a_{i_j})}$ be the (s,t)-walk which contains arc a_{i_j}. When q_j is inserted into X, its deviation node is set to v_r. Due to the order of the arcs $a_{i_1}, \ldots, a_{i_h}$ in the adjacency list of v_r, processing the node v_r in the repeat-loop of MPS (Step 2) after q_j has been extracted from X yields the walk q_{j+1} for $j = 1, \ldots, h-1$.

First, we prove the claim for walk q_1. Let b_i be the last arc in $(s, b_1, v_1, \ldots, b_r, v_r)$ which is not an element of the shortest path tree T, if such an arc exists. By the outer induction's hypothesis, the walk

$$q := \begin{cases} (s, b_1, v_1, \ldots, b_i, v_i)p^{v_i}, & \text{if } \{b_1, \ldots, b_r\} \cap (A \setminus T) \neq \emptyset, \\ p^s, & \text{otherwise} \end{cases}$$

is determined exactly once by MPS. Its deviation node is

$$d_q = \begin{cases} v_{i-1}, & \text{if } i > 1, \\ s, & \text{if } i = 1 \text{ or } q = p^s. \end{cases}$$

Furthermore, it holds by definition of the arc b_i that q contains the starting subpath $(s, b_1, v_1, \ldots, b_r, v_r)$, since $b_{i+1}, \ldots, b_r \in T$ if $q \neq p_s$, and $b_1, \ldots, b_r \in T$ if $q = p_s$. Hence, v_r is a node of q. Since this node is particularly a successor of d_q on the walk q and reached without yielding a cycle, v_r is processed in the repeat-loop (Step 2) of the algorithm after q has been extracted from X. In doing so, the walk q_1 is detected since a_{i_1} is the first arc in the order of the adjacency list of node v_r that succeeds the tree arc and whose attachment to path $(s, b_1, v_1, \ldots, b_r, v_r)$ does not yield a cycle.

It remains to show that q_1 cannot be constructed again. Let $\bar{q}$ be any (s,t)-walk from which the algorithm constructs q_1 in its repeat-loop (Step 2). We will now show that $\bar{q} = q$. Note that q_1 and $\bar{q}$ share the same starting subpath up to the deviation node of q_1. Furthermore, the deviation node of a constructed walk is always the tail node of the last arc on the walk that is not contained in the shortest path tree T (if all arcs are elements of T, the deviation node is s). That arc is a_{i_1} for the walk q_1, yielding the deviation node $d_{q_1} = v_r$. Hence, $\bar{q}$ has the starting subpath $(s, b_1, v_1, \ldots, b_r, v_r)$. Moreover, the deviation node $d_{\bar{q}}$ of $\bar{q}$ must lie on this subpath, or v_r will not be processed after extracting $\bar{q}$. As mentioned above, q_1 cannot be constructed from any walk q_j for $j = 1, \ldots, h-1$ (as q_{j+1} is constructed from q_j). Note that this also holds for $j = h$. Therefore, we have $d_{\bar{q}} \neq v_r$ since otherwise $\bar{q} = q_j$ for some $1 \leq j \leq h$. Furthermore, the arc of $\bar{q}$ which is emanating from $d_{\bar{q}}$ cannot be contained in the shortest path tree T (unless $d_{\bar{q}} = s$), but all succeeding arcs on $\bar{q}$ must be elements of T. This implies that b_i is the arc that emanates from $d_{\bar{q}}$, which yields $\bar{q} = q$.

Assume that the claim has been proven for the walk q_j for $j < h$. We show next that it also holds for q_{j+1}. As mentioned before, this walk is immediately constructed after q_j was extracted from X. Due to the way the algorithm selects the diverging arc at a processed node, q_{j+1} can only be determined via q_j. Since q_j is detected exactly once, the same applies to q_{j+1}. □

As mentioned above, from Lemma 4.4 and Lemma 4.5 follows the correctness of the algorithm MPS.

Theorem 4.6. *After a finite number of steps, the algorithm* MPS *correctly determines a sequence $(p_1, \ldots, p_k)$ containing the k shortest paths from s to t, or all such paths if there are fewer than k. In addition, it holds that $c(p_1) \leq \ldots \leq c(p_k)$.*

Proof. By Lemma 4.5, especially each (s,t)-paths is determined exactly once. It only remains to be shown that the running time of MPS is finite. Note that this fact is assured since Lemma 4.5 can be applied to each walk which is constructed by the algorithm. □

4.5 Runnning Time Complexity

Finally, we are interested in the running time complexity of MPS. Example 4.7 proves that its worst-case running time is indeed exponential. However, experimental results in [MPS99] reveal that MPS is very fast in practice. E.g., in an undirected 100×25 grid network for $k = 1000$, MPS runs 500 to 1000 times faster than the algorithm of Yen [Yen71] and that of Katoh, Ibaraki, and Mine [KIM82]. Moreover, these two algorithms are, to the knowledge of the authors of MPS, the most efficient previously known.

By the following example, we prove the worst-case running time of MPS to be exponential.

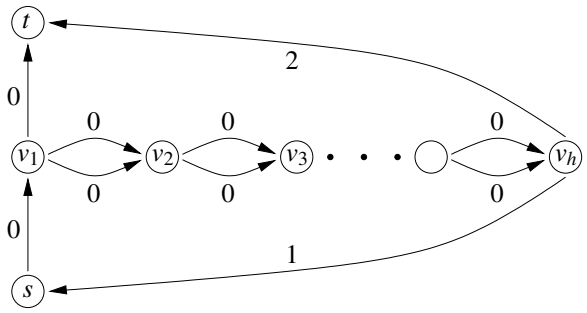

Figure 4.4: Worst-case example for MPS.

Example 4.7. Consider the task to find the $k = 2$ shortest (s,t)-paths in the digraph shown in Figure 4.4, where $h \geq 2$. Since there are two parallel arcs from v_1 to v_2, from v_2 to v_3, ..., and from v_{h-1} to v_h, the number of (v_1, v_h)-paths in the digraph is 2^{h-1}. Obviously, the shortest (s,t)-path in the digraph is $p_1 = (s,(s,v_1),v_1,(v_1,t),t)$. All further walks which are ever contained in the set X during the performance of the algorithm are either of the form

$$p = (s,(s,v_1),v_1)p'(v_h,(v_h,s),s)p_1 \text{ with length } c(p) = 1, \tag{4.2}$$

or

$$q = (s,(s,v_1),v_1)p'(v_h,(v_h,t),t) \text{ with length } c(q) = 2, \tag{4.3}$$

where p' is a (v_1, v_h)-path. Since walks are extracted from X in order of increasing length (Lemma 4.4), all 2^{h-1} walks of form (4.2) are extracted from X before the first path q of form (4.3), which yields our second path p_2 whereupon the algorithm terminates. Furthermore, since the number of arcs is $m := |A| = 2h + 2$, we have $h = m/2 - 1$. Hence, the running time of MPS is $\Omega((\sqrt{2})^m)$, which is exponential in m.

Note that Example 4.7 can easily be adapted to prove an exponential running time for MPS also for simple digraphs. The example still works if for $i = 1, \ldots, h-1$ one new node w_i is added and one of the two parallel arcs (v_i, v_{i+1}) is replaced by two successive arcs (v_i, w_i) and (w_i, v_{i+1}) with lengths $c((v_i, w_i)) = c((w_i, v_{i+1})) = 0$. However, note that digraphs with such successive arcs might be very similar to digraphs

having parallel arcs. For many optimization problems in digraphs, a node v of degree two that is incident to arcs (u,v) and (v,w) can be removed by replacing the two arcs by one arc (u,w). Such a transformation may be applied to shortest paths computations unless v is an end node of paths to find: The length of (u,w) is defined as the sum of the lengths of (u,v) and (v,w). Figure 4.5 depicts a digraph without nodes of degree two, in which MPS also needs exponential running time to compute the two shortest (s,t)-paths. At least 2^{h-1} walks containing cycles are extracted from X before the second path p_2. Since $m = 6h-2$, the number 2^{h-1} is still exponential in m.

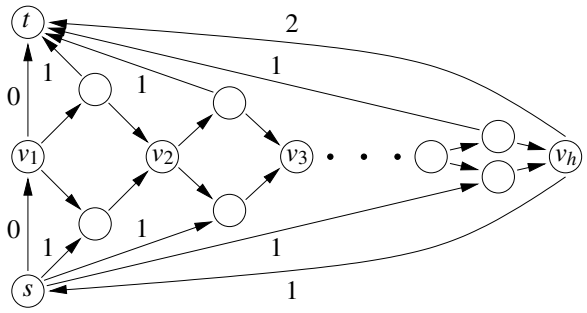

Figure 4.5: Worst-case example for MPS, where the digraph is simple and contains no node of degree two. All arcs without labels have length 0.

Chapter 5

Experimental Results

In this chapter we report on an extensive experimental study of the **DSCA**-algorithms presented in Chapter 3. These studies were carried using our self-developed simulation tool *CARWA* whose features are shortly described in the appendix. We compare the blocking probabilities of these algorithms in relation to the offered traffic load in four optical networks based on two topologies. These networks and the way their dimensioning is determined are introduced in Section 5.1. The random model used to generate request sequences is presented in Section 5.2. Section 5.3 describes the actual settings in the used simulations. Finally, we present and analyze the obtained results in Section 5.4.

5.1 Four Real-World Optical Networks

For our experimental studies, we took into account four different optical networks that are based on two topologies, the 17-*nodes topology* and the 14-*nodes topology*. The dimensionings of the considered optical networks are based on given static traffic demands shown in Table 5.1 and Table 5.2. These static demands were generated with respect to US-American and German population data. Partitioning the population into different regions leads to the 14-nodes and 17-nodes topologies (cf. []Poensgen+etal:ONDM03). Due to the methods the dimensionings are constructed, we call them *shortest path dimensioning* and *low cost dimensioning*.

Given a graph $G = (V,E)$ and a static traffic demand matrix, the *shortest path dimensioning* is obtained as follows. For each unit of static traffic between nodes $u,v \in V$, a shortest $[u,v]$-path is computed. Let $p(e)$ be the number of those paths which contain edge $e \in E$. Then, the set of wavelengths in the optical network is defined as $\Lambda := \{\lambda_1,\ldots,\lambda_\chi\}$, where $\chi := \max_{e\in E} p(e)$ is the maximum number of paths which have an edge in common. Moreover, each edge $e \in E$ is equipped with the set of wavelengths $W(e) := \{\lambda_1,\ldots,\lambda_{p(e)}\}$. That is, every edge contained in a path yields one wavelength on that edge in the optical network. Note that this dimensioning does not imply that there is a set of conflict-free lightpaths using the computed paths (such

Node	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	1	1	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-
6	0	0	3	3	2	-	-	-	-	-	-	-	-	-	-	-	-
7	0	1	1	1	1	2	-	-	-	-	-	-	-	-	-	-	-
8	0	0	0	0	0	0	1	-	-	-	-	-	-	-	-	-	-
9	0	0	0	0	0	0	2	0	-	-	-	-	-	-	-	-	-
10	0	1	4	4	2	3	3	1	0	-	-	-	-	-	-	-	-
11	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-
12	0	1	5	3	1	3	1	0	2	5	1	-	-	-	-	-	-
13	0	1	1	0	0	1	1	0	0	1	1	2	-	-	-	-	-
14	0	0	0	0	0	0	0	0	0	0	1	1	1	-	-	-	-
15	0	0	0	0	0	1	0	0	0	0	0	2	1	0	-	-	-
16	0	0	0	0	1	1	1	0	0	0	0	2	1	1	1	-	-
17	0	0	0	0	1	1	0	0	0	2	0	1	1	0	0	1	-

Table 5.1: Static demand matrix for the German 17-nodes network.

Node	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	13	-	-	-	-	-	-	-	-	-	-	-	-	-
3	2	3	-	-	-	-	-	-	-	-	-	-	-	-
4	4	6	1	-	-	-	-	-	-	-	-	-	-	-
5	6	9	2	4	-	-	-	-	-	-	-	-	-	-
6	3	4	1	2	3	-	-	-	-	-	-	-	-	-
7	5	7	2	3	6	2	-	-	-	-	-	-	-	-
8	1	1	0	1	1	0	1	-	-	-	-	-	-	-
9	4	6	1	3	4	3	4	1	-	-	-	-	-	-
10	9	13	2	6	10	5	8	1	14	-	-	-	-	-
11	6	9	2	4	6	4	5	1	13	16	-	-	-	-
12	11	16	4	7	11	4	8	2	7	15	10	-	-	-
13	1	2	0	1	1	0	1	0	1	2	1	2	-	-
14	3	5	1	2	3	1	2	0	2	4	3	5	1	-

Table 5.2: Static demand matrix for the US 14-nodes network.

conflict-free lightpaths would exist if the network provided arbitrary wavelength conversion in its nodes). Hence, it might be impossible to satisfy all given static demands together.

In contrast, the *low cost dimension* yields a network that is always capable of satisfying all static demands. The low cost dimensionings were computed using a method of [KWZ03]. This method focuses on cost minimization while providing backup capacities for failure situations. The four optical networks which result for the 17-nodes and 14-nodes topology with respect to the given static demand matrices are shown in Figure 5.1 and Figure 5.2.

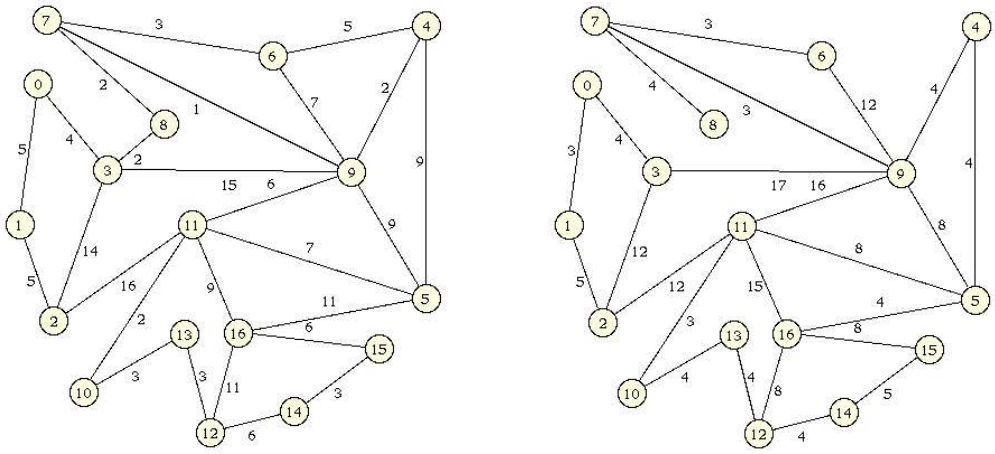

Figure 5.1: The 17-nodes network with shortest path (left) and with low cost dimensioning (right).

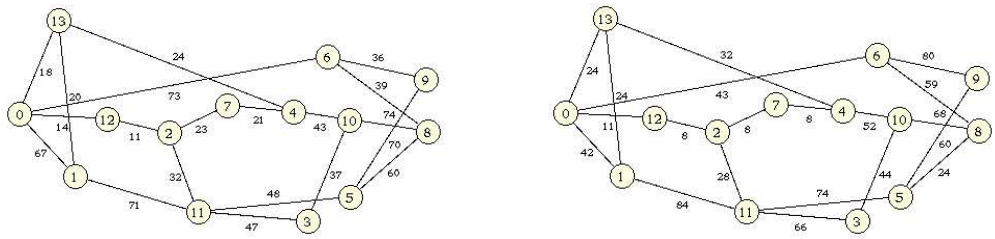

Figure 5.2: The 14-nodes network with shortest path (left) and with low cost dimensioning (right).

5.2 Traffic Model and Request Sequences

Recall that each connection request σ_j in the problem **DSCA** is of the form:

$$\sigma_j = (u_j, v_j, b_j, t_j^{\texttt{start}}, t_j^{\texttt{stop}}, p_j),$$

where $d_j := t_j^{\texttt{stop}} - t_j^{\texttt{start}}$ is the duration of the call. In our simulation studies, we consider a restrict version in which we assume several parameters to be constant. Each

connection request requires one lightpath for a constant duration of 1 hour and yields a profit of 1 if it is accepted. That is, $b_j = p_j = d_j = 1$. Hence, the specification of call σ_j can be reduced to:

$$\sigma_j = (u_j, v_j, t_j^{\texttt{start}}).$$

The used model of call arrivals depends on a given traffic demand matrix, and an integer number m, called the *multiplex factor* that serves to control the offered load. Let (G, Λ, W) be the considered optical network, where $G = (V, E)$. For each unit of static demand between nodes $u, v \in V$, m sources generate calls for connections between u and v according to a modified *Poisson arrival process*. More precisely, the inter arrival times between two calls generated by a single source are determined as the sum of a constant and an exponentially distributed random value. The constant is chosen to be 1. In doing so, it is ensured that no two calls of the same source can overlap, since the constant duration of each call equals 1, too. This models that no customer requires two connections simultaneously. The mean of the exponential distribution is chosen equal to 11. That is, each source generates on average one call of duration 1 in a time span of 12 hours. This reflects the observation of a network provider that a customer who has a permanent connection uses it only for about $1/12$ of the time.

At this point the impact of the multiplex factor becomes clear: A multiplex factor of 1 yields dynamic traffic that corresponds to the traffic actually incurred on permanent connections. But as connections can be set up and taken down on demand, additional network capacities are temporarily available, and further demands could be satisfied. In principle, 12 calls with duration 1 could be accepted in 12 hours using the same lightpath. This would correspond to a permanent connection that is used continuously. At a multiplex factor of 12, exactly this number of calls are generated on average. Therefore, multiplex factor 12 corresponds to 100% offered load.

5.3 Simulation Model

For each of the four optical networks, we compare the blocking probabilities of the considered algorithms on request sequences which are generated for multiplex factors from 1 to 12. That is, we investigate the dependence of an algorithm's blocking probability on the multiplex factor. For both 17-nodes networks, we generated for each multiplex factor 21 batches of 5000 requests each, where the first batch serves as an onset for the sequence. In doing so, we achieve a balanced state before the actual simulation starts with the remaining 20 batches. From the blocking probability values obtained for the calls in these batches, i.e., the ratio of rejected and generated requests, we determine a 95% confidence interval (cf. [LK00]). For the 14-nodes networks, the number of calls per batch is reduced to 1000 since the processing of calls in these optical networks requires more computational effort due to its larger dimensioning. Unfortunately, the resulting confidence intervals for the blocking probabilities in the 14-nodes networks are quite large. Hence, we must evaluate the corresponding results with more cautious. Since blocking probabilities greater than 5% are considered unacceptable for the customer according to network providers the graphics are plotted within a logarithmic scale to emphasize the smaller ranges. However, providers desire a threshold value of 0.5% for blocking probabilities.

5.4 Results

We consider the results of our experiments for each class of algorithms separately and then present overview tables with a selection of the best algorithms.

5.4.1 Greedy-Type Algorithms

First, we consider the greedy-type algorithms with partial wavelength search (cf. Section 3.1.1). Their blocking probabilities in the different optical networks are depicted in Figures 5.3–5.6.

For the 17-nodes networks (Figure 5.3 and Figure 5.4), the results are very similar and show clearly which wavelength selection strategies are advantageous and which not.

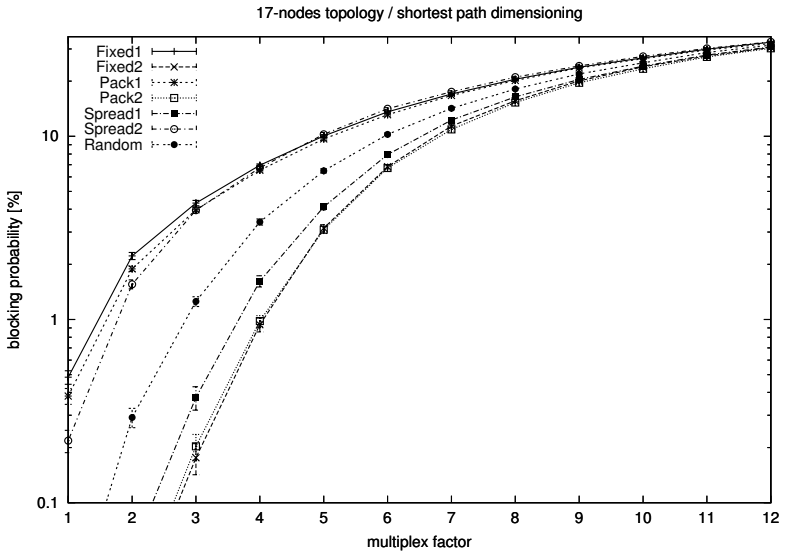

Figure 5.3: Blocking probabilities (including confidence intervals) of the greedy-type algorithms with partial wavelength search in the 17-nodes network with shortest path dimensioning.

Note that RANDOM shows medium performance in both cases for all traffic loads. This is not a surprise: good and bad selections are both made similarly frequently. Preferable wavelength search orders are those of FIXED2 and PACK2, which yield nearly the same blocking probabilities and perform best in both networks, whereas the opposite search orders used by FIXED1 and SPREAD2 are apparently inferior: At a multiplex factor of 3, the latter achieve a blocking probability which is approximately 10 times higher than that of FIXED2 and PACK2 in the 17-nodes network with shortest path dimensioning. The other way round, at a blocking probability of 1% FIXED2 and PACK2 can handle up a multiplex factor of 4, which corresponds to an offered load of about 30%, whereas FIXED1 and SPREAD2 can not even cope with a multiplex factor of 2, i.e., half the offered load.

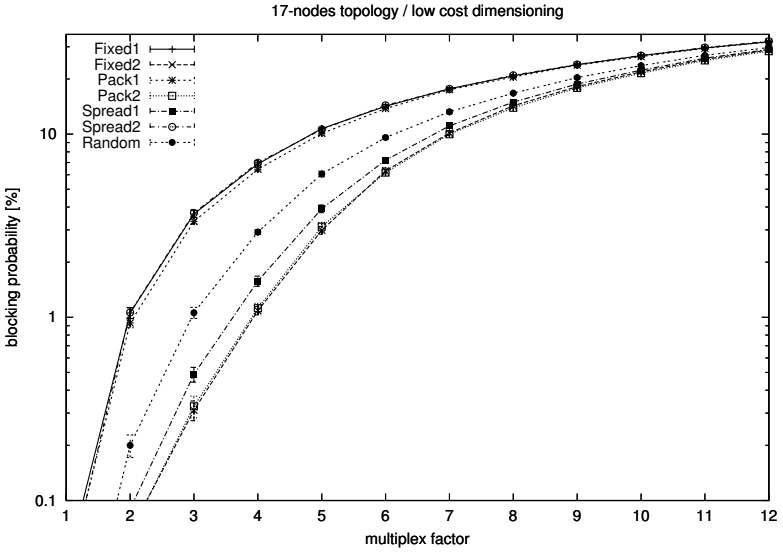

Figure 5.4: Blocking probabilities (including confidence intervals) of the greedy-type algorithms with partial wavelength search in the 17-nodes network with low cost dimensioning.

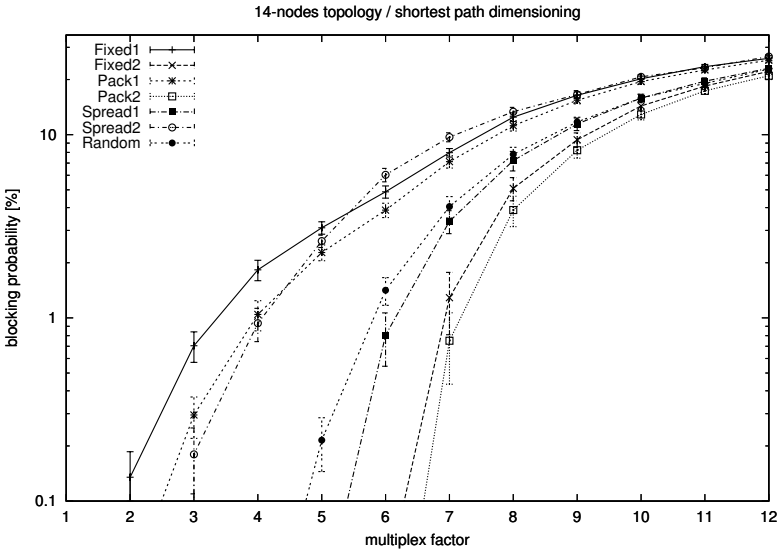

Figure 5.5: Blocking probabilities (including confidence intervals) of the greedy-type algorithms with partial wavelength search in the 14-nodes network with shortest path dimensioning.

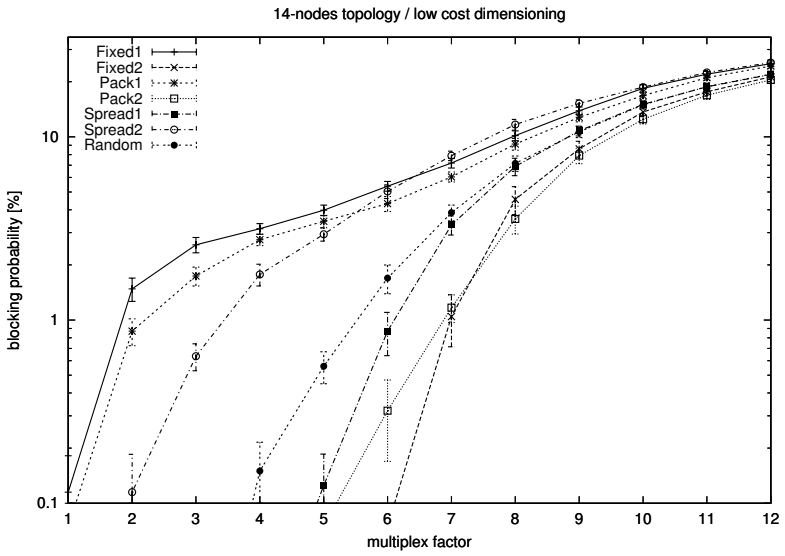

Figure 5.6: Blocking probabilities (including confidence intervals) of the greedy-type algorithms with partial wavelength search in the 14-nodes network with low cost dimensioning.

Similar, but more differentiated trends, are apparent the 14-nodes networks (Figure 5.5 and Figure 5.6). For the shortest path dimensioning, PACK2 performs best, followed by FIXED2. For the low cost dimensioning, however, FIXED2 is better up to an offered load of about 60% (multiplex factor 7). The 14-nodes networks show the huge difference in offered load the algorithms can handle at small blocking probability even more obviously: PACK2 and FIXED2 deal with more than 50% offered load at a blocking probabily of 1%, FIXED1 and SPREAD2 do not even manage 20% for the low cost dimensioning. Finally, let us consider the difference between PACK1 and PACK2. Obviously, defining the wavelength order with respect to increasing edge availabilities (PACK2) is superior to using decreasing availabilities (PACK1). For the version of SPREAD, however, SPREAD1 outperforms SPREAD2 since rarely installed wavelengths have usually small edge utilizations. Therefore, the wavelength search order of SPREAD1 tends slightly to those of FIXED2 and PACK2.

Figure 5.7–5.10 show the results for the greedy-type algorithms with total wavelength search (cf. Section 3.1.2). For each network, the graphic looks very similar to the corresponding graphic for the greedy-type algorithms with partial wavelength search: Using a certain wavelength search order for breaking ties in an EXHAUSTIVE version seems to be as effective as the order is for the wavelength selection in the corresponding greedy-type algorithm with partial wavelength search. Moreover, any EXHAUSTIVE version is superior to the corresponding greedy-type algorithm with partial wavelength search which uses the same wavelength order. To give an example for the supremacy of EXHAUSTIVE$_{f2}$ and EXHAUSTIVE$_{p2}$, consider the 14-nodes network with low cost dimensioning. These algorithms can handle circa 60% offered load (multiplex factor 7) at a blocking probability of 0.54%, while EXHAUSTIVE$_{f1}$ and EXHAUSTIVE$_{p2}$ can not even manage 25% load (multiplex factor 3) for that blocking

probability.

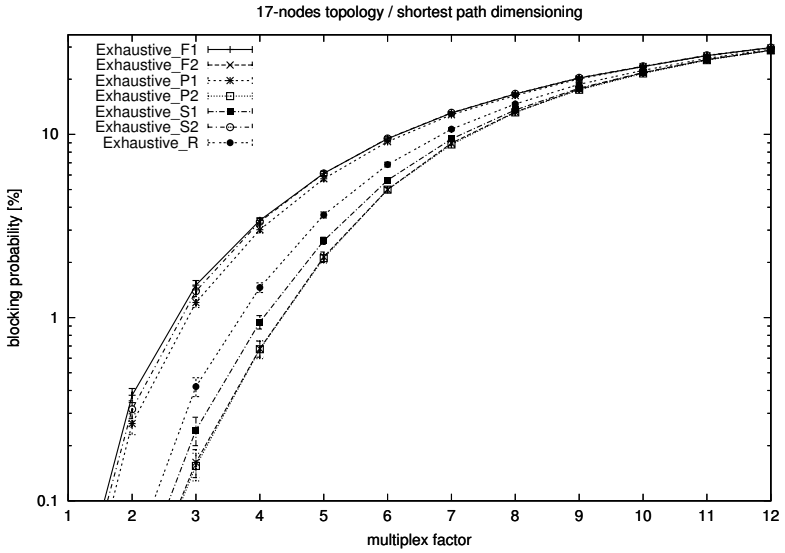

Figure 5.7: Blocking probabilities (including confidence intervals) of the greedy-type algorithms with total wavelength search in the 17-nodes network with shortest path dimensioning.

5.4.2 Versions of ALR

For the algorithm ALR, we considered the versions CCC, TCC, CCE, TCE, CEC, TEC, CEE, and TEE of Section 3.2.2. Recall that the first parameter considers whether given traffic demands T are taking into account or not C, a second parameter of E specifies that end nodes with currently small connectivity are protected, and E for the last parameter specifies that shorter lightpaths are additionally protected.

For the 17-nodes networks only four selected versions are plotted in Figure 5.11 and Figure 5.12 since the blocking probabilities of the eight variants are very close to each other. For both dimensionings the ranking of the four selected versions is the same: CEE achieves the smallest blocking probabilities, followed by CCC and TCC, while TEC performs worst. For the network with low cost dimensioning, the obtained blocking probabilities are closer to each other as for the other dimensioned network. We guess that the reason for this is that the routing decisions made by the algorithms in the network with low cost dimensioning are more similar to each other. Let us mention that the blocking probabilities of all remaining variants whose results are not plotted lie between the values of CEE and TEC for both 17-nodes networks.

Note that this also holds for the 14-node networks (Figure 5.13 and Figure 5.14), where the results for all eight versions of ALR are depicted. Be aware that we used another scale in Figure 5.13 and Figure 5.14, starting with multiplex factor 3. Also in these networks, CEE is superior to the other algorithms (note that the line styles changed). Compared with the results in the 17-nodes networks, the curves lie less

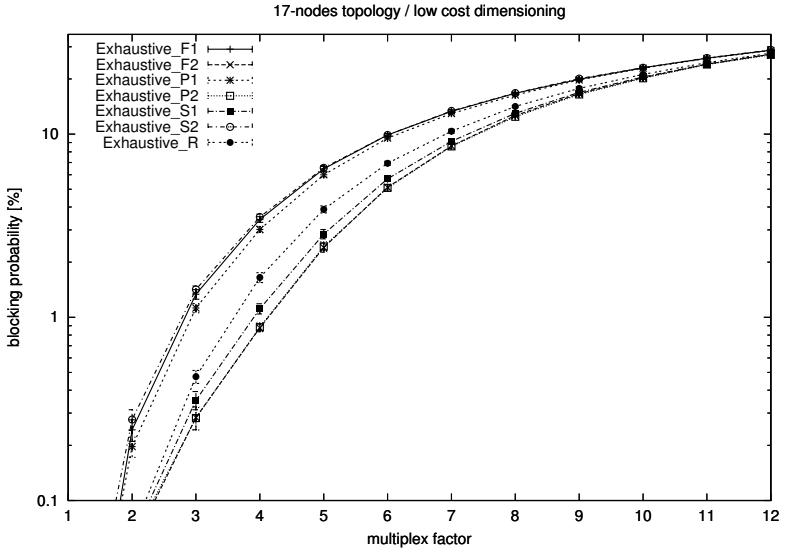

Figure 5.8: Blocking probabilities (including confidence intervals) of the greedy-type algorithms with total wavelength search in the 17-nodes network with low cost dimensioning.

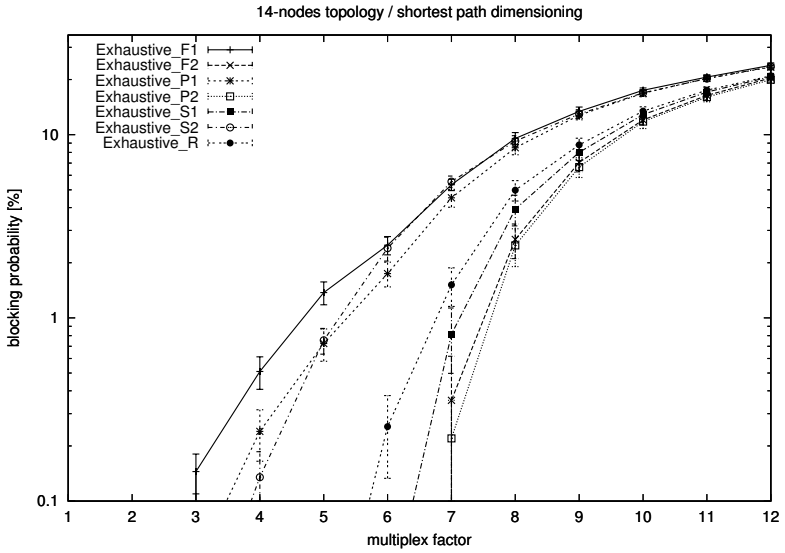

Figure 5.9: Blocking probabilities (including confidence intervals) of the greedy-type algorithms with total wavelength search in the 14-nodes network with shortest path dimensioning.

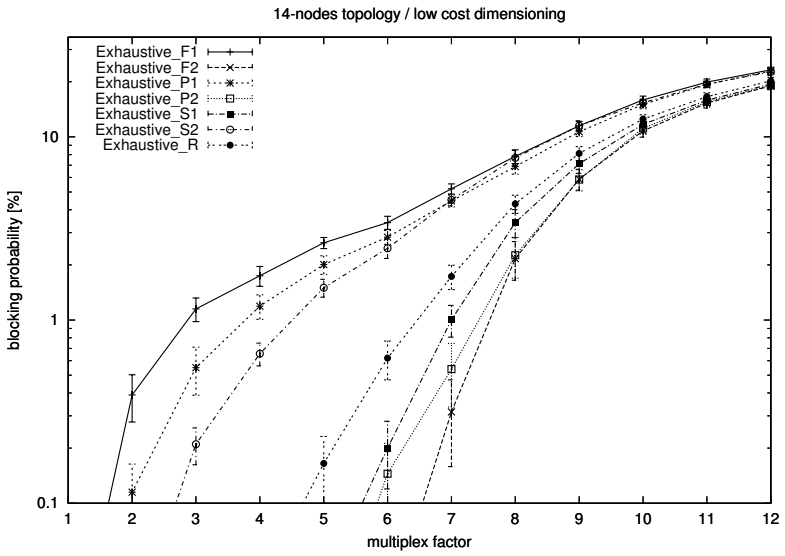

Figure 5.10: Blocking probabilities (including confidence intervals) of the greedy-type algorithms with total wavelength search in the 14-nodes network with low cost dimensioning.

close to each other. This trend, which also applies to the greedy-type algorithms, lets us suppose that the quality of a **DSCA**-algorithm becomes more apparent for optical networks with larger dimensionings which provide more different routing possibilities. That is, for such large networks, the choice of the algorithm is more important. Furthermore, notice that the versions which take into account the given average traffic demands using the function as defined by (T) perform worse than their unweighted counterparts with function (C).

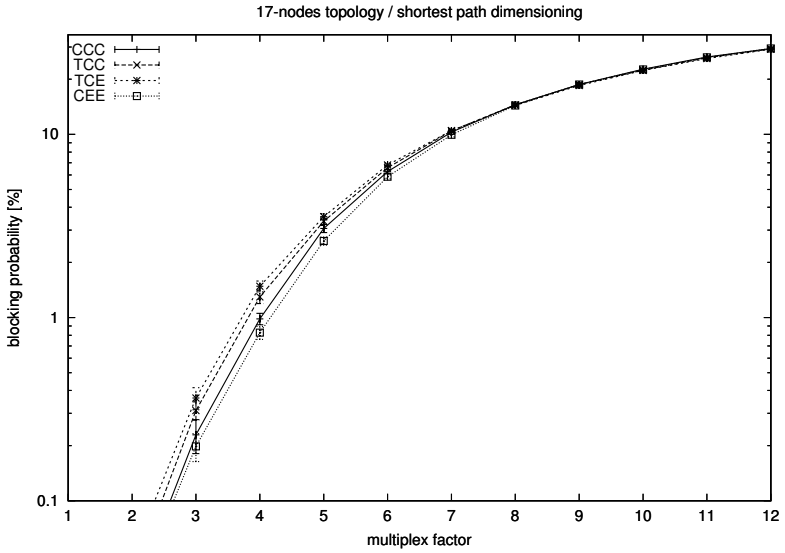

Figure 5.11: Blocking probabilities (including confidence intervals) of selected versions of ALR in the 17-nodes network with shortest path dimensioning.

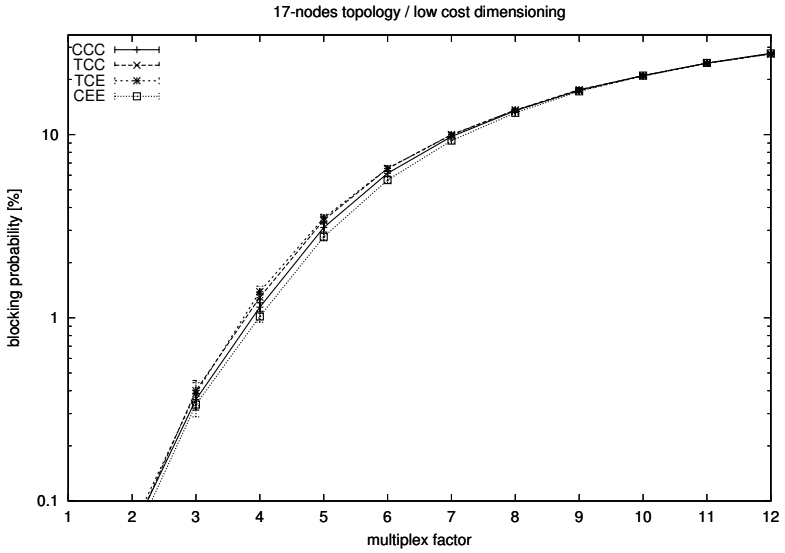

Figure 5.12: Blocking probabilities (including confidence intervals) of selected versions of ALR in the 17-nodes network with low cost dimensioning.

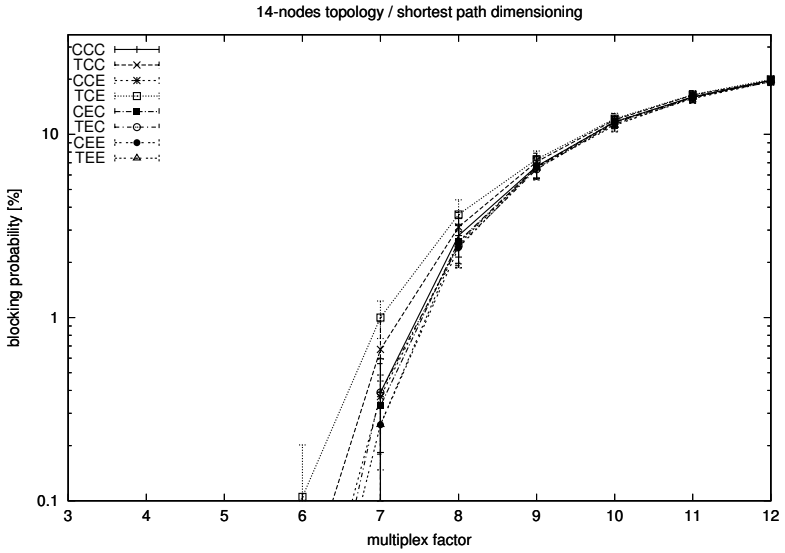

Figure 5.13: Blocking probabilities (including confidence intervals) of selected versions of ALR in the 14-nodes network with shortest path dimensioning.

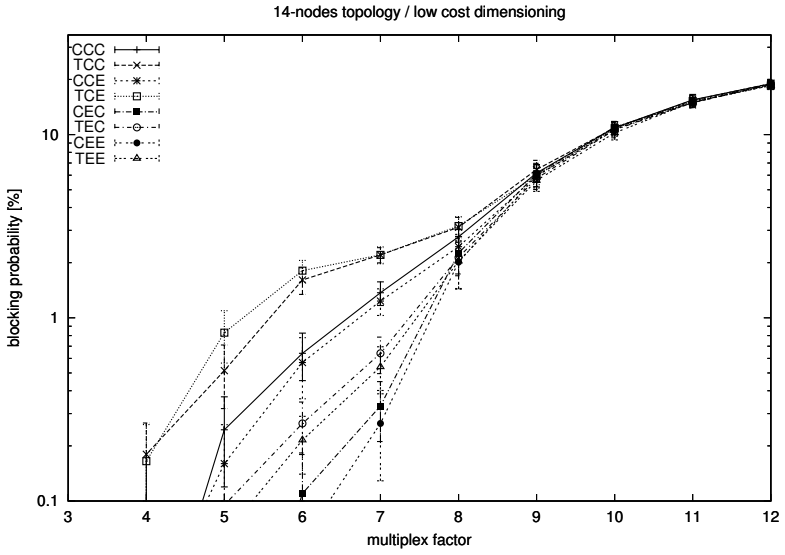

Figure 5.14: Blocking probabilities (including confidence intervals) of selected versions of ALR in the 14-nodes network with low cost dimensioning.

5.4.3 Versions of SFR

As a last group, we consider the versions based on SFR, namely, the original SFR itself, SFR(T), ASFR, and ASFR(T), (cf. Section 3.2.3). The results for the 17-nodes networks in Figure 5.15 and Figure 5.16 reveal that also for this class of **DSCA**-algorithms, the versions which incorporate average traffic demands are inferior to their counterparts, particularly for the shortest path dimsioning. In the interesting range up to blocking probability 5%, ASFR slightly outperforms the basic version SFR. Also similar to the versions of ALR, we see that the results for the network with low cost dimensioning are closer to each other than those for the shortest path dimensioned network.

Slightly different are the observations for the 14-nodes network. For the shortest path dimensioning, see Figure 5.17, ASFR(T) performs best for an offered load larger than 75%, which corresponds to a multiplex factor of 9. But since the corresponding blocking probability at this point exceeds 5%, this region of offered load is hardly of interest. In the 14-nodes network with the low cost dimensioning, however, ASFR outperforms the other versions significantly in the whole relevant range of offered load, as depicted in Figure 5.18. Also for the versions of SFR, the blocking probability values in the 14-nodes networks are further spread than those in the 17-nodes networks.

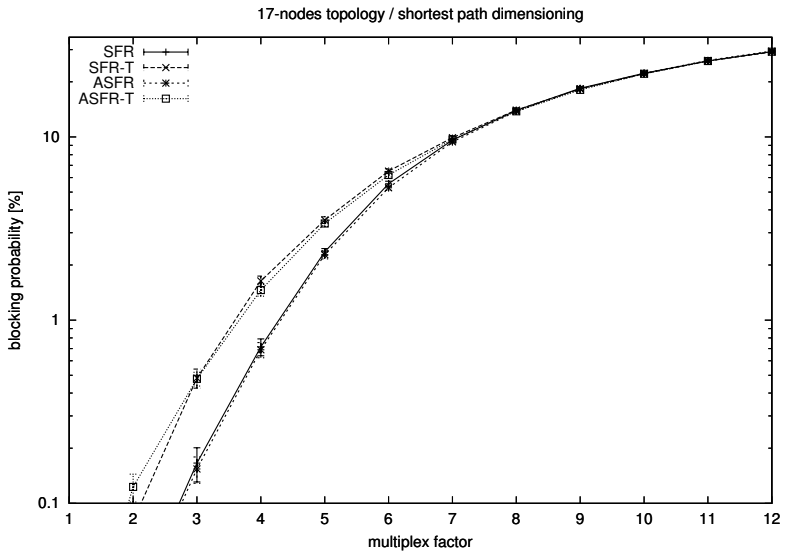

Figure 5.15: Blocking probabilities (including confidence intervals) of the variants for SFR in the 17-nodes network with shortest path dimensioning.

5.4.4 Best Algorithms Revisited

In order to determine which **DSCA**-algorithms perform globally best, we compare the results of the best algorithms from each group. The selected algorithms are PACK2 for the greedy-type algorithms with partial wavelength search, EXHAUSTIVE$_{p2}$ for those

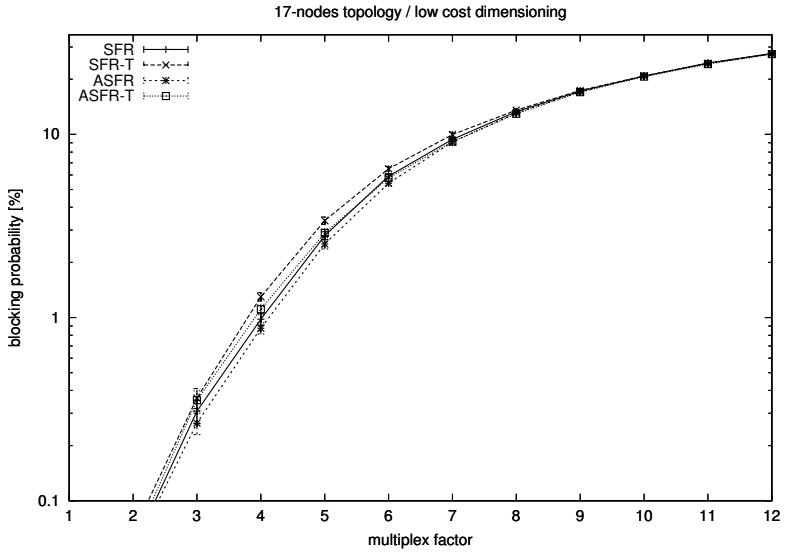

Figure 5.16: Blocking probabilities (including confidence intervals) of the variants for SFR in the 17-nodes network with low cost dimensioning.

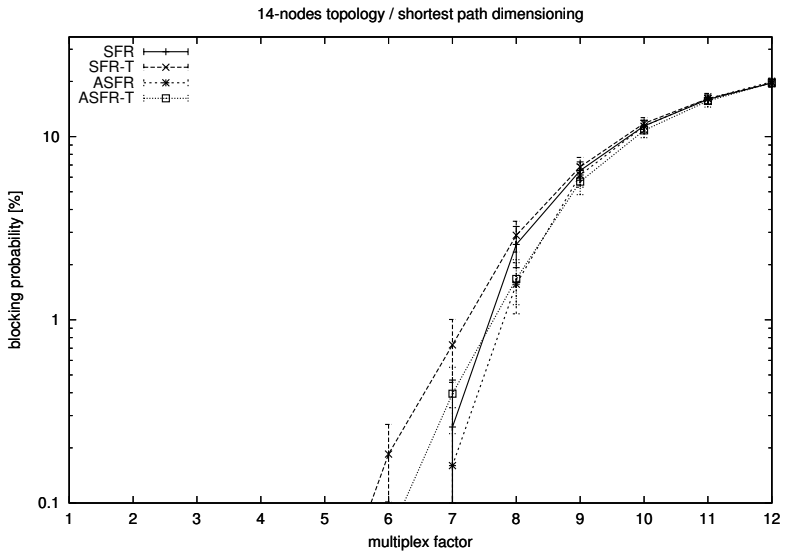

Figure 5.17: Blocking probabilities (including confidence intervals) of the variants for SFR in the 14-nodes network with shortest path dimensioning.

with total wavelength search, CEE for the variants of ALR, and ASFR for the algorithms based on SFR. Furthermore, first simulation results for the algorithm NFR which was developed last have been obtained and added. The runs in the 14-nodes network with low cost dimensioning have not finished until now. Therefore, we can only report on its results for the 17-nodes networks and the 14-nodes network with shortest path dimensioning.

The blocking probabilities of the selected algorithms for the 17-nodes networks are plotted in Figure 5.19 and Figure 5.20. For the network with shortest path dimensioning, EXHAUSTIVE$_{p2}$ performs best, but ASFR is equal to EXHAUSTIVE$_{p2}$ up to a multiplex factor of 4 (about 33% offered load), where their blocking probabilities reach a value of 0.7%. The algorithm CEE and PACK2 perform a bit worse with a similar progression in both networks. A very different curve yields NFR. It starts very bad for small offered load, comes close to the others at multiplex factor 5, and performs very well for high offered load. To give one example on this network, the blocking probabilities for multiplex factor 4 are nearly 0.7% for EXHAUSTIVE$_{p2}$ and ASFR, 0.8% for CEE, 1.0% for PACK2, and 1.3% for NFR.

For the low cost dimensioning, NFR yields exactly the same blocking probabilities as EXHAUSTIVE$_{p2}$, which are best from multiplex factor 4 on. If the offered load is smaller, ASFR is slightly superior to them. The results for CEE and PACK2 again perform a bit inferior to EXHAUSTIVE$_{p2}$ and ASFR.

The results for the 14-nodes network are depicted using a different scale, see Figure 5.21 and Figure 5.22. For both dimensionings, the algorithm ASFR clearly outperforms the others, and PACK2 yields the highest blocking probabilities by far. For the network with shortest path dimensioning, NFR is second, followed by EXHAUSTIVE$_{p2}$, and CEE. For example, at a multiplex factor of 7, the achieved blocking probabilities are 0.16%, 0.18%, 0.22%, 0.26%, and 0.75% for ASFR, NFR, EXHAUSTIVE$_{p2}$ CEE, and PACK2, respectively. In contrast, to the network with shortest path dimensioning, CEE is superior to EXHAUSTIVE$_{p2}$ for the low cost dimensioning. For multiplex factor 7, the blocking probabilities for ASFR, CEE, EXHAUSTIVE$_{p2}$, and PACK1 are 0.20%, 0.27%, 0.54%, and 1.17%, respectively.

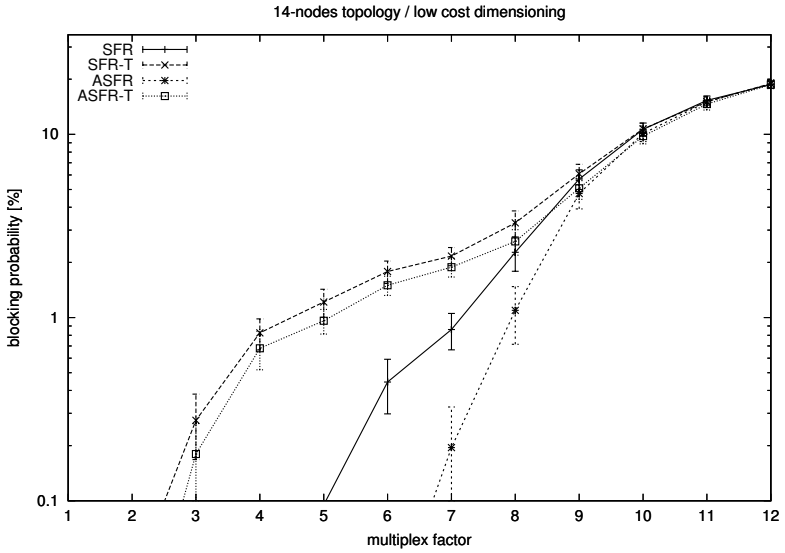

Figure 5.18: Blocking probabilities (including confidence intervals) of the variants for SFR in the 14-nodes network with low cost dimensioning.

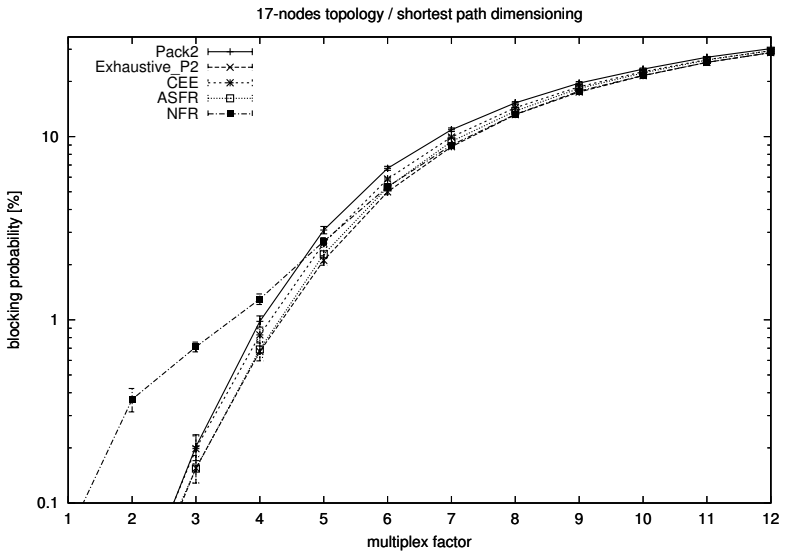

Figure 5.19: Blocking probabilities (including confidence intervals) of selected algorithms in the 17-nodes network with shortest path dimensioning.

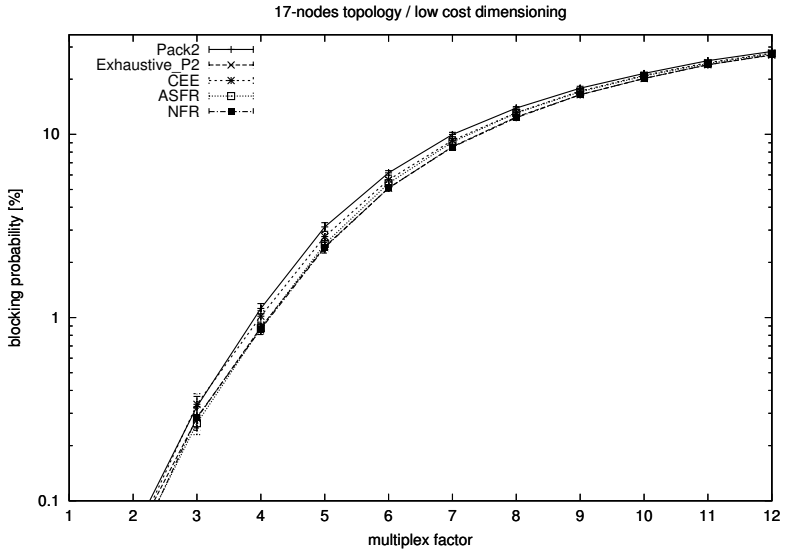

Figure 5.20: Blocking probabilities (including confidence intervals) of selected algorithms in the 17-nodes network with low cost dimensioning.

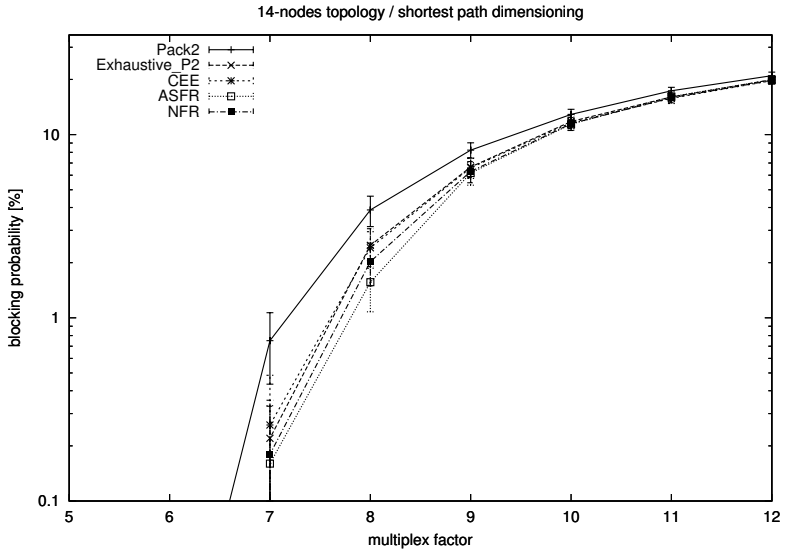

Figure 5.21: Blocking probabilities (including confidence intervals) of selected algorithms in the 14-nodes network with shortest path dimensioning.

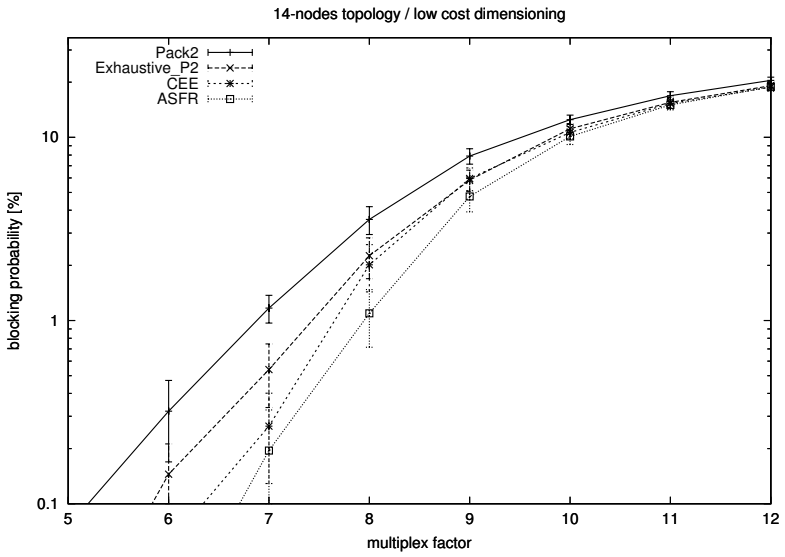

Figure 5.22: Blocking probabilities (including confidence intervals) of selected algorithms in the 14-nodes network with low cost dimensioning.

Chapter 6

Conclusions and Outlook

In this diploma thesis, we focused on the design and evaluation of online algorithms for **DSCA**, a problem of call admission and routing and wavelength assignment in optical networks. Experimental results showed that the blocking probabilities of **DSCA**-algorithms may differ substantially, even for algorithms which only differ in their tie-breaking rules.

For the greedy-type algorithms that are based on routing along shortest lightpaths in some wavelength (the choice of which defines the version), we obtained the following results. Each EXHAUSTIVE variant that using one of the considered orders on the set of wavelengths for breaking ties outperforms the greedy-type algorithm with partial wavelength search that selects a shortest lightpath in the first possible wavelength in that order. Furthermore, we have seen that for both classes of greedy-type algorithms the selection of this order substancially affects the performance. This is particularly interesting for the EXHAUSTIVE versions since they only differ in their strategies to break ties. That is, globally shortest routing lightpaths are often available in many wavelengths. The studies suggest that less available wavelengths should be preferred. The availability of a wavelength can be either defined as the total number of edges on which the wavelength is available with respect to the current network status, or as the number of edges on which the wavelength is installed. Both versions yield similar blocking probabilities. That is, among the greedy-type algorithms, $\mathsf{EXHAUSTIVE}_{p2}$ and $\mathsf{EXHAUSTIVE}_{f2}$ perform best.

Moreover, we observe that the network fitness algorithms based on ALR, which route in such a way that the total number of newly blocked lightpaths in the network is minimized, could not benefit from incorporating informations about the average traffic demands. However, the way some versions of ALR protect lightpaths with end nodes whose connectivity is currently small and lightpaths which are short proves to be advantageous. We believe this to be the reason that the version CEE is superior to the others.

The studies about the variants of SFR which are based on the idea of considering only edge-disjoint lightpaths between each pair of nodes yield the following results. Also for these algorithms considering average traffic demands is disadvantageous in most cases. However, if the durations of calls are also given at the moment of their

arrival, the version ASFR which also account for currently blocked lightpaths performs better than the other variants.

As an overall result in the evaluation of algorithms, it turned out that no algorithm is superior to the best greedy-type algorithm in the 17-nodes networks. This also applies to the algorithm NFR which is based on solving a multicommodity flow problem in order to compute a set of edge-disjoint lightpaths. However, in the relevant range up to a blocking probability of 5%, the algorithm ASFR achieves only marginally different blocking probabilities from those of the best greedy-type algorithms. In the 14-nodes networks, which have larger dimensionings, the differences between the algorithms are much more apparent: ASFR performs best, in particular for the low cost dimensioning. For this dimensioning also CEE outperforms the greedy-type algorithms.

Throughout the simulation experiments, we observed that the achieved blocking probabilities of the different algorithms in the 14-nodes networks diverge more from each other than those in the 17-nodes networks. We suppose the reason for this to be, among others, the size of network's dimensioning: For larger dimensionings, the algorithms have more freedom of decision, whereby the impact of good or bad routings is strengthened. We believe that the network fitness algorithms could be superior to the best greedy-type algorithm in larger networks in general. It has been observed in other fields of optimization that greedy strategies might degrade for approaches for more inhomogeneous problem data. That is, the greedy-type algorithms could lead to worse results if the parameters of the connection requests are more irregular, e.g., changing demands and durations.

For the practicability of the network fitness algorithms, it must be mentioned that they require great computational effort, especially in comparison with the greedy-type algorithms. However, the running time of the new algorithms can be reduced significantly by taking into account only a fixed number of possible routing lightpaths. In doing so, the versions based on SFR can handle each call in polynomial time. This also holds for SALR, a variant of ALR which we did not take into account in the experiments.

For further research, the following points are of interest. In order to obtain more evidence on the qualities of the algorithms, additional studies based on other networks are necessary. These should also include real-world traffic data instead of the random call generation due to Poisson arrivals applied in this work. Moreover, more inhomogeneous call parameters like changing demands and durations should be considered. After these, the additional parameters of **DMCA** should be taken into account: different customer classes and different service classes. Also suitable handling of failure situation is important. Note that this problem has more offline than the call processing in the standard operating state of the network, since a lot of information gets known at the same time. That is, a possibly large set suspended connections must be rerouted at the same time.

Another point is that of rejection criteria. Note that all presented algorithms completely omit the call admission part of **DSCA**. Possibly, the option to reject calls improves performance of an algorithm. In our opinion, this could apply to scenarios with high offered traffic. Note that the profit function is highly associated with rejection criteria since particularly calls with profits that are small in comparison to their required service should be rejected.

Appendix A

Table of Notations

G	denotes a graph
D	denotes a digraph
n	total number of nodes in a graph or digraph
m	total number of edges or arcs in a graph or digraph, respectively
$\delta^+(v)$	set of all arcs in a digraph with start node v
$\delta^-(v)$	set of all arcs in a digraph with end node v
p, q	usually denote paths or walks
$V(p)$	set of nodes of a walk p in a graph or digraph
$E(p)$	set of edges of a walk p in a graph
$A(p)$	set of arcs of a walk p in a digraph
pq	denotes the concatenation of two walks p and q
(G, Λ, W)	denotes an optical network
L	set of all lightpaths in an optical network
$L(u,v)$	set of all $[u,v]$-lightpaths in an optical network
S	denotes a network status
$M_S = (m_{ij}^{(S)})$	denotes a link-status matrix in status S
L_S	set of all lightpaths in a network that are available in status S
$L_S(u,v)$	set of all $[u,v]$-lightpaths in a network that are available in status S
σ	denotes a request sequence
σ_j	denotes a call
u_j, v_j	denote the end nodes of the connection for call σ_j
b_j	denotes the demand (number of lightpaths) of call σ_j
$t_j^{\mathtt{arr}}$	denotes the arrival time of call σ_j
$t_j^{\mathtt{ans}}$	denotes the latest answer time of call σ_j
$t_j^{\mathtt{start}}$	denotes the start time of the connection for call σ_j
$t_j^{\mathtt{stop}}$	denotes the expiration time of the connection for call σ_j
d_j	denotes the duration of the connection for call σ_j
c_j	denotes the customer class of call σ_j
q_j	denotes the customer class of call σ_j
p_j	denotes the profit of call σ_j

Appendix B

Basic Definitions

In the following, we introduce the graph theoretical notations that are used in this thesis. For a more detailed look at graph theory, cf. [Jun99].

Definition B.1. A *graph* G is a pair $G = (V, E)$ consisting of finite set $V \neq \emptyset$ of nodes and a set $E \subseteq V^{(2)}$ of edges, where $V^{(2)}$ denotes the set of unordered pairs of elements in V. An edge with nodes $u, v \in V$ is denoted by uv. The nodes u and v are said to be the *end nodes* of the edge uv. Moreover, we say that the nodes u and v are *incident* with the edge uv and that u and v are *adjacent* or *neighbors*. G is called to be *simple* if there is at most one edge between each pair of nodes and if there is no edge connecting a node with itself.

Definition B.2. A *walk* in a graph $G = (V, E)$ is a sequence $(v_0, e_1, v_1, \ldots, e_k, v_k)$ consisting of nodes and edges of G such that $e_i = v_{i-1}v_i$ for $1 \leq i \leq k$. The nodes v_0 and v_k are called the *end nodes* of the walk, which is also said to be a $[v_0, v_k]$-*walk*. If, in addition, the nodes of a walk are pairwise distinct, the walk is a *path*. If $v_1, \ldots, v_{k-1}$ are pairwise distinct and $v_0 = v_k$, the walk is called a *cycle*.

Definition B.3. A *digraph* D is a pair $D = (V, A)$ consisting of finite set $V \neq \emptyset$ of nodes and a set $A \subseteq V \times V$ of arcs. An arc from node u to v is denoted by (u, v). Node u is said to be the *tail node* of (u, v), and v is said to be the *head node* of (u, v). Moreover, we say that the nodes u and v are *incident* with the arc (u, v). D is called to be *simple* if there is at most one arc from u to v for each pair of distinct nodes $u, v \in V$ edge between each pair of nodes and if there is no edge connecting a node with itself.

Definition B.4. A *walk* in a digraph $D = (V, A)$ is a sequence $(v_0, a_1, v_1, \ldots, a_k, v_k)$ consisting of nodes and arcs of D such that $a_i = (v_{i-1}, v_i)$ for $1 \leq i \leq k$. v_0 is called the *start node* and v_k is called the *end node* of the walk, which is also said to be a (v_0, v_k)-*walk*. If, in addition, the nodes of a walk are pairwise distinct, the walk is a *path*. If $v_1, \ldots, v_{k-1}$ are pairwise distinct and $v_0 = v_k$, the walk is called a *cycle*.

Appendix C

Deutsche Zusammenfassung

In dieser Diplomarbeit werden Online-Algorithmen zur Bearbeitung von Verbindungsanfragen in optischen Telekommunikationsnetzen untersucht.

Für ein gegebenes optisches Telekommunikationsnetz ist die Entscheidung zu treffen, eingehende Verbindungsanfragen entweder anzunehmen und die geforderte Verbindung zur Verfügung zu stellen oder diese abzulehnen. Für dieses Problem werden bereits bekannte und neue Algorithmen vorgestellt und diese mit Hilfe von Simulationsstudien bewertet.

Ein optisches Netz besteht aus Knoten, zwischen denen Leitungen verlaufen. Jede Leitung enthält optische Fasern, auf denen Daten als optische Signale übermittelt werden können. Die Knoten des Netzes sind mit Schaltern ausgestattet, die optische Signale weiterleiten können, ohne diese zwischenzeitlich in ihre digitale Information umwandeln zu müssen. Dadurch können optische Kanäle entlang mehrerer Fasern realisiert werden. Diese optischen Kanäle werden auch als Lichtwege bezeichnet. Mit Hilfe der Technik Wavelength Division Multiplexing (WDM) können optische Signale mehrere sich nicht gegenseitig beeinflussende Wellenlängen verwenden. Das bedeutet, dass mehrere Lichtwege dieselbe Faser gleichzeitig benutzen können, sofern sie dabei verschiedene Wellenlängen verwenden. In dieser Arbeit werden keine Wellenlängenkonverter betrachtet, d.h. ein Lichtweg muss dieselbe Wellenlänge auf allen durchlaufenden Fasern verwenden.

Jede eingehende Anfrage fordert eine Verbindung zwischen zwei Knoten im Netz für eine bestimmte Dauer, die vorher meist nicht bekannt ist. Die Verbindung erfordert eine bestimmte Zahl von Lichtwegen. (Im Rahmen der in dieser Arbeit enthaltenen Simulationen werden nur Anfragen berücksichtigt, die genau einen Lichtweg benötigen.) Für jede angenommene Anfrage müssen entsprechende Lichtwege zur Verfügung gestellt werden. Diese Dienstleistung liefert einen bestimmen Profit. Das Ziel ist die Maximinierung des Gesamtprofits. Bei den durchgeführten Untersuchungen werden konstante Profite angenommen. In diesem Fall ist das Ziel der Profitmaximierung äquivalent zur Minimierung der Wahrscheinlichkeit einer Anfrageablehnung, der sogenannten Blockierungswahrscheinlichkeit. Der wesentliche Charakter des Problems besteht darin, dass zukünftige Anfragen bis zum Zeitpunkt ihrer Ankunft

unbekannt bleiben, jede Anfrage also ohne Kenntnis der zu erwartenden weiteren Anfragen angenommen oder abgelehnt werden muss.

Ein Algorithmus für dieses Problem muss also für eine gegebene Anfrage die Entscheidung treffen, ob und auf welchen Lichtwegen die geforderte Verbindung realisiert oder ob die Anfrage abgelehnt werden soll. Zu den untersuchten Algorithmen gehören die sogenannten Greedy-Algorithmen sowie neu entwickelte Algorithmen, die auf dem Konzept der Netzfitness beruhen. Die Greedy-Algorithmen verwenden stets kürzeste Lichtwege (die Länge eines Lichtwegs ist die Anzahl der von ihm verwendeten Fasern), und unterscheiden sich nur in der Wahl der Wellenlängen. Es gibt zwei Klasse dieser Algorithmen. Die Greedy-Algorithmen mit partieller Wellenlängensuche durchsuchen alle im Netz verfügbaren Wellenlängen gemäß einer speziellen Ordnung (diese definiert die verschiedenen Varianten) und wählen einen kürzesten Lichtweg in der ersten Wellenlänge dieser Ordnung, in der es überhaupt einen Lichtweg für die aktuelle Anfrage gibt. Die Greedy-Algorithmen mit totaler Wellenlängensuche hingegen bestimmen in jeder Wellenlänge einen kürzesten Lichtweg und wählen unter diesen den insgesamt kürzesten aus. Sie unterscheiden sich in der Wahl der Lichtwege, wenn es in mehreren Wellenlängen kürzeste Lichtwege gibt.

Die neu entwickelten Netzfitness-Algorithmen versuchen eine sinnvolle Funktion zu finden, die einem aktuellen Status des Netzes, der sich über die augenblicklich realisierten Lichtwege definiert, den sogenannten Fitnesswert zuweist. Dieser Wert soll die Fähigkeit des Netzes widerspiegeln, zukünftige Anfragen annehmen zu können. Zur Realisierung von Verbindungen für ankommende Anfragen werden diejenigen Lichtwege gewählt, für die der resultierende Fitnesswert maximiert wird.

Der erste Algorithmus *available-lightpaths-reduction* (ALR) definiert den Fitnesswert als die Anzahl Lichtwege, die augenblicklich verwendet werden könnten. Es wird also jeweils der Lichtweg gewählt, der eine minimale Abnahme an verfügbaren Lichtwegen bewirkt. Für diesen Algorithmus gibt es noch einige Varianten, die ermöglichen, wichtige Lichtwege höher zu gewichten.

Ein zweiter Algorithmus *single-flow-reduction* (SFR) betrachtet zwischen jedem Knotenpaar in jeder Wellenlänge nur noch Kanten-disjunkte Lichtwege. Dadurch soll verhindert werden, dass zu viele unwichtige, nie verwendete Lichtwege, berücksichtigt werden.

Der letzte betrachtete Netzfitness-Algorithmus *network-flow-reduction* (NFR) soll in jeder Wellenlänge nur Systeme von Kanten-disjunkten Lichtwegen berücksichtigen. Diese Lichtwege könnten theoretisch alle gleichzeitig im optischen Netz realisiert werden.

Für die Simulationen wurden vier Netze untersucht, die auf zwei verschiedenen Topologien basieren. Die Simulationsergebnisse für die Greedy-Algorithmen zeigen deutlich, dass die Wahl der Wellenlängenordnung jeweils einen entscheidenen Einfluss auf die erreichten Blockierungswahrscheinlichkeiten haben. Am besten schneiden die Varianten ab, die einen über alle Wellenlängen kürzesten Lichtweg wählen und dabei falls dieser nicht eindeutig ist, denjenigen auswählen, dessen Wellenlänge im Augenblick auf am wenigsten Fasern im Netz verfügbar ist.

Fast genauso gut ist in diesen Netzen eine Variante von SFR. Für die Netze, die über mehr Kapazitäten verfügen, erweist sich dieser jedoch als deutlich besser.

List of Algorithms

1 FFC 22
2 Generic greedy-type algorithm with partial wavelengths search. 27
3 Generic greedy-type algorithm with total wavelengths search. 30
4 Generic network fitness algorithm. 34
5 MPS 61

Bibliography

[AAF+96] B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén, *On-line competitive algorithms for call admission in optical networks*, Proceedings of the 4th Annual European Symposium on Algorithms, 1996, pp. 431–444.

[AAP93] B. Awerbuch, Y. Azar, and S. Plotkin, *Throughput-competitive on-line routing*, Proceedings of the 34th Annual IEEE Symposium on the Foundations of Computer Science, 1993, pp. 32–40.

[ABFR94] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén, *Competitive, non-preemptive call control*, Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, 1994, pp. 312–320.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Networks flows*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[BEY98] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, 1998.

[BFL96] Y. Bartal, A. Fiat, and S. Leonardi, *Lower bounds for on-line graph problems with applications to on-line circuit and optimal routing*, Proceedings of the 28th Annual ACM Symposium on the Theory of Computing, 1996, pp. 531–540.

[BK95] A. Birman and A. Kershenbaum, *Routing and wavelength assignment methods in single-hop all-optical networks with blocking*, Proceedings of INFOCOM '95, 1995, pp. 431–438.

[BSSB95] K. Bala, T. Stern, K. Simchi, and K. Bala, *Routing in a linear lightwave networks*, IEEE/ACM Transactions on Networking **3** (1995), 459–469.

[CGK92] I. Chlamtac, A. Ganz, and G. Karmi, *Lightpath communications: An approach to high bandwidth optical WAN's*, IEEE Transactions on Communications **40** (1992), 1171–1182.

[Epp98] D. Eppstein, *Finding the k shortest paths*, SIAM Journal on Computing **28** (1998), no. 2, 652–673.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and intractability (a guide to the theory of NP-completeness)*, W.H. Freeman and Company, New York, 1979.

[HJK+03] R. Hülsermann, M. Jäger, S. O. Krumke, D. Poensgen, J. Rambau, and A. Tuchscherer, *Experimental study of routing algorithms in optical networks*, Proceedings of the 7th IFIP Working Conference on Optical Network Design & Modelling, Kluwer Academic Press, 2003, To appear.

[Jun99] Dieter Jungnickel, *Graphs, networks and algorithms*, 5 ed., Springer, 1999.

[KIM82] N. Katoh, T. Ibaraki, and H. Mine, *An efficient algorithm for k shortest simple paths*, Networks **12** (1982), no. 2, 411–427.

[KP02] S. O. Krumke and D. Poensgen, *Online call admission in optical networks with larger demands*, Proceedings of the 28th International Workshop on Graph-Theoretic Concepts in Computer Science, vol. 2573, Springer, 2002, pp. 333–344.

[KT95] J. Kleinberg and E. Tardos, *Disjoint paths in densely embedded graphs*, Proceedings of the 36th Annual IEEE Symposium on the Foundations of Computer Science, 1995, pp. 531–540.

[KV00] Bernhard Korte and Jens Vygen, *Combinatorial optimization: Theory and algorithms*, 1 ed., Springer, 2000.

[KWZ03] A. M. C. A. Koster, R. Wessäly, and A. Zymolka, *Transparent optical network design with sparse wavelength conversion*, Proceedings of the 7th IFIP Working Conference on Optical Network Design & Modelling, Kluwer Academic Press, 2003, To appear.

[Law72] E. L. Lawler, *A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem*, Management Science **18** (1972), 401–405.

[LK00] A. M. Law and W. D. Kelton, *Simulation modeling and analysis*, McGraw-Hill, Boston, 2000.

[LMSPR98] S. Leonardi, A. Marchetti-Spaccamela, A. Presciutti, and A. Rosén, *Online randomized call-control revisited*, Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, 1998, pp. 323–332.

[MA98] A. Mokhtar and M. Azizoglu, *Adaptive wavelength routing in all-optical networks*, IEEE/ACM Transactions on Networking **6** (1998), no. 2, 197–206.

[MPS99] E. Q. V. Martins, M. M. B. Pascoal, and J. L. E. Santos, *An algorithm for ranking loopless paths*, Tech. report, CISUC, 1999.

[Muk97] B. Mukherjee, *Optical communication networks*, McGraw-Hill, New York, 1997.

[RS98] R. Ramaswami and K. N. Sivarajan, *Optical networks: A practical perspective*, Morgan Kaufmann Publishers, Inc., ISBN 1-55860-445-6, 1998.

[SB99] T. E. Stern and K. Bala, *Multiwavelength optical networks: A layered approach*, Addison Wesley Longman, Inc., ISBN 0-201-30967-X, 1999.

[Val79] L. G. Valiant, *The complexity of enumeration and reliability problems*, SIAM Journal on Computing **8** (1979), no. 3, 410–421.

[Yen71] J. Y. Yen, *Finding the k shortest loopless paths in a network*, Management Science **17** (1971), 712–716.

[Yen72] ———, *Another algorithm for finding the k shortest loopless network paths*, Proceedings of 41st National Meeting of the Operations Research Society of America, vol. 20, 1972.

[ZA95] Z. Zhang and A. Acampora, *A heuristic wavelength assignment algorithm for multi-hop WDM networks with wavelength routing and wavelength re-use*, IEEE/ACM Transactions on Networking **3** (1995), 281–288.