Local and global restoration of node and link failures in telecommunication networks

Diplomarbeit bei
Prof. Dr. M. Grötschel

vorgelegt von
Sebastian Orlowski

Fachbereich Mathematik der TU Berlin,
Studiengang Wirtschaftsmathematik

Berlin, den 3. Februar 2003

Abstract

This diploma thesis deals with the restoration problem in telecommunication networks. The goal is to find a cost minimal capacity capacity assignment on the edges and nodes of a network such that given demands can be satisfied even in case of the failure of an edge or node in the network. Moreover, restrictions on the routing paths (like length restrictions) and hardware constraints have to be satisfied.

A Mixed Integer Programming model is presented which takes into account restoration requirements as well as hardware constraints and which abstracts from a particular restoration protocol and failure situation. This abstraction provides new insight into the structure of the network restoration problem and shows that from a mathematical point of view, the commonly used restoration techniques Link Restoration, Path Restoration and Reservation are not as different as they seem to be from a practical point of view.

In addition, our model allows (but is not limited to) optimizing working capacity, intended for normal use, and spare capacity, intended for rerouting purposes in case of a failure, in one step. Furthermore, our formulation of capacity cost allows taking into account the effects of discrete, non-linear cost structures which are common in practice. Up to our knowledge, no publication in the existing literature covers all these aspects, let alone in one model, although they are of major practical interest.

The model has been implemented in a Branch and Cut framework. The theoretical background of the algorithmic procedure is presented in detail, including computational complexity investigations on the pricing problem.

The abstraction from a particular restoration protocol turns out to be useful both from a theoretical and computational point of view. In fact, our investigations suggest a distinction into *Local Restoration* and *Global Restoration* rather than into Link Restoration, Path Restoration, Reservation and mixtures of these concepts. In addition to the theoretical aspects of the algorithmic procedure, some implementational details are briefly discussed.

Our implementation has been tested on 14 real world instances, which is described in detail. One part of the computational results consists of a comparison of optimal network cost values using different restoration mechanisms, applied to securing either all single node failures, all single edge failures or both. In addition, the effects of a discrete cost structure are investigated, which has rarely been considered yet in literature. Furthermore, the cost difference between joint and successive working and spare capacity optimization is investigated. In the second part of the computational results, several heuristics for the network restoration problem are compared with respect to both solution quality and time.

Acknowledgments

Special thanks go to my advisor Roland Wessäly for always being reachable for questions and for providing me with the opportunity to use the existing framework of DISCNET as a basis for my implementation. I would also like to thank Andreas Bley for his advice in computational complexity questions, and Mathias Schulz and Alexander Kröller for proofreading various drafts of this document. Last but not least, thanks to Prof. Grötschel for the possibility to write this diploma thesis in connection with my work at ZIB.

Contents

Chapter 1

Introduction

Due to the liberalization of telecommunication markets in Europe, which has started a few years ago and is still running, the availability of a telecom network has become a major competitive edge for telecommunication providers. Many companies depend largely on the ability to access remote databases, for instance, banks and insurances. Such companies could easily be bankrupted by a failure of their computer network. Given that these computer networks often consist of several subnetworks in different buildings or even cities, connected with each other by fiber optic cables, it is crucial to these companies that the interconnecting network can quickly recover from a network failure.

Common types of network failures are cable cuts caused by dredgers or other external impacts, like earthquakes or sabotage. Another common reason for interrupted connections in a network are node failures caused by hardware failures or by an overloaded network, i.e., a node has to cope with more traffic than it is able to handle.

One example may illustrate the possible impact of such a network failure ([Hei02][1]): on June 15, 2002, saboteurs cut four principal optical fiber links of the Spanish telecommunications company Telefonica. As a result, about half of the Spanish population was affected in some way when using their telephone, accessing the Internet or regarding TV. Many cash dispensers did not work, and flights had to be delayed by a couple of hours on several airports.

Nowadays, many important network connections are made of fiber optic cables. Since these are quite expensive, optical fiber networks are often quite sparse. In addition, as such cables are able to carry a huge amount of data, even the impact of a single network component failure on the overall network can be quite important.

To allow traffic to be rerouted around a failing network component, some extra capacity reserved for this purpose has to be reserved in the network in addition to the capacity intended for normal use. The latter, the so-called *base* or *working capacity*, is the capacity for the *Normal Operating State (NOS)*, when the whole network is intact. The *spare capacity* is the additional capacity needed to enable traffic to be rerouted in so-called *failure states* when one or more network components are out of service.

Recovery from a network failure consists of three parts: First, providing enough capacity such that affected traffic can be rerouted around a failing network component. Second, actually rerouting the affected traffic until reparation is finished, and third, repairing the failing component. It is the first part that is going to be discussed in this document.

A network is said to be *survivable* (w.r.t. the given parameters) if it provides enough

[1] further examples can be found in [Wess00], among others.

working capacity to satisfy certain demands (i.e., forecasted amount of traffic between some pairs of nodes) in NOS, and enough spare capacity such that a specified part of affected traffic can be rerouted in case of certain component failures. Unfortunately, there is a tradeoff between the grade of survivability and capacity cost since by installing more capacities, the network can usually be made more resistant against component failures.

The objective of the *Network Restoration Problem* is to find a cost-minimal solution among all capacity assignments ensuring a given degree of survivability of the network. In our case, it can be stated as follows:

> **Given** a (potential or existing) network, demand values between its nodes, a list of installable capacities together with their cost and maybe some further restrictions and parameters,
> **find** a minimal cost assignment of working and spare capacities on the network
> **such that**
>
> - all demands can be satisfied in NOS using the working capacity,
> - in case an arbitrary node or link fails, the affected demands can be rerouted using the installed spare capacity,
> - all additional restrictions are satisfied.

Examples of additional restrictions are hardware requirements at the nodes and edges of the network or constraints on the employed routing paths.

In most cases, it is far from being evident how to find good solutions, as the total cost is influenced by many parameters like the cost structure of the installable capacity, the hardware at the nodes, the structure of the demands that have to be satisfied, the considered operating states, and many more.

A common approach to the network restoration problem is the application of heuristics in order to search for feasible capacity assignments. Many heuristics for survivable network design have been presented in literature, some of which effectively lead to good solutions in reasonable amount of time. However, it is usually very difficult to estimate the quality of a solution given by such a heuristic since in most cases, no usable lower bound on the cost of a feasible solution is known.

We address this problem by employing a Branch and Cut framework, which in many cases yields good solutions and lower bounds, and thus a provable quality of the solutions.

Another common approach is breaking down the problem into the *successive* planning of a network topology, a NOS routing and spare capacity. A major part of the literature on survivable network design deals with the cost-minimal assignment of *spare* capacity to the links of a network, given working capacities and a NOS routing (see Section 2.2).

This approach is certainly of interest for network planners whose aim is to install additional capacity to ensure survivability of an existing network. However, the cost savings of a *joint* optimization of working and spare capacity can be considerable (at the same level of survivability), as has been found by several authors and as we could confirm (see Sections 2.2 and 6.4.1 for details).

We developed a Mixed Integer Programming model for the network restoration problem which abstracts from a particular restoration technique or failure situation. As a result, the model applies to most commonly used restoration mechanisms (which will be outlined in Section 2.1) and allows a flexible handling of failure scenarios. In addition, the model allows

working and spare capacity to be optimized together as well as in separate steps. To the best of our knowledge, no model presented in literature yet provides an integrated approach to all these issues.

Our model considers node cost as well as edge cost and takes into account that edge and node capacity cost often exhibit highly non-linear behaviour. This allows us to reflect economies of scale and other particularities of the cost structure. Many publications assume continuous capacities; our computational results on several real world instances (see Chapter 6) show than this assumption can be largely misleading in practice.

We implemented the model as being part of the network planning software DISCNET (DImensioning Survivable Capacitated NETworks), originally developed by Roland Wessäly for his PhD thesis [Wess00] at ZIB[2]. It turned out that the abstraction from a particular restoration mechanism or failure state also greatly simplified the implementation of the model.

We then tested the implementation on several real world telecommunication networks. The algorithmic procedure which we used is described in detail in Chapter 4; this includes some computational complexity investigations for the commonly used restoration techniques.

Our computational results consist of two main parts:

1. An optimal cost comparison of small networks is given when these are secured against either all single edge failures simultaneously, all single node failures simultaneously or both, each of these scenarios being combined with the commonly used restoration mechanisms Link and Path Restoration. Moreover, the sometimes counterintuitive effects of a discrete cost structure of the installable capacities are investigated.

2. Several heuristics are compared on dense networks that could not be solved to optimality in all cases and where it is difficult to find good solutions at all. This includes a comparison of joint and successive NOS and spare capacity optimization.

This document is structured as follows: Chapter 2 gives an overview on widespread techniques in survivable network design, followed by a literature review on the topic. Our mathematical model will be explained in Chapter 3. Chapter 4 focuses on the theoretical issues of the employed algorithmic procedure, while the implementational issues are covered in Chapter 5. Chapter 6 presents our computational results. Finally, some conclusions are drawn in Chapter 7.

[2]Konrad-Zuse-Zentrum für Informationstechnik Berlin, `www.zib.de`

Chapter 2

Overview on survivable network design

This chapter discusses the most common survivability techniques to give the reader an idea of the topic. Afterwards, an outline of the existing literature on survivable network design is given.

2.1 Survivability concepts

There are essentially two main strategies to achieve survivability of a network: *protection* and *restoration*. While this paper focuses on restoration, it is useful to know the basics of other concepts as well in order to know where this work ranges in w.r.t. to the available literature. Hence, we will give a short summary of widespread protection and restoration strategies together with their assets and drawbacks; for a discussion of these methods that is closer to hardware aspects, see [SWSLGF98].

Throughout this document, we are going to differentiate between various *operating states* of the network: the *Normal Operating State* (mostly referred to as *NOS*) where the whole network is intact, and different *failure states* where one or more network components (nodes or edges) are out of service. A *routing* is a network flow satisfying given demands; a routing is *feasible* if the corresponding flow does not exceed any edge or node capacity. In our case, a routing is given as a set of paths in the network together with a flow value on each of these paths.

A technique for ensuring survivability of a network has to take different objectives into account, which partially contradict each other in practice:

- It should minimize the damage caused by a failure (in terms of affected demand, in terms of time in which the network is out of service or in terms of demand that is definitely lost, according to the context).
- It should be cheap in terms of needed capacity.
- It should be easy to implement.
- It should be able to guarantee fast switching to a backup routing in case of a failure (if there are backup paths at all). This often goes together with ease of implementation.

2.1.1 Protection

Network planners usually employ the term *protection* when speaking of strategies that ensure survivability by conditions on the NOS routing while no or little reconfiguration of the network is required in case of a failure. In particular, if reconfiguration is required at all, it is done by *preassigned* or *dedicated* backup paths, such that no calculation is needed at runtime. Examples of common protection schemes are:

Topological considerations:

For instance, in order to be able to recover from any edge failure, the network has to be at least twice edge-connected, i.e., any two nodes have to be connected by at least two edge-disjoint paths. Usually, due to the high cost of protecting every network component, network planners will define a hierarchy of survivability requirements for the nodes and edges as a function of their importance to the network and choose an appropriate topological layout.

Path length restrictions:

These are also known as *hop limits*, indicating that any path used in an NOS routing must not contain more than a given number of edges. This number may be given globally or on a per-demand basis. These constraints reduce the number of nodes and edges per path and thus the probability that a given path is affected by the failure of a network component.

Diversification:

A *diversification parameter* $\delta \in [0, 1]$ may be given (globally or on a per-demand basis). Its meaning is that that for each demand with value d and diversification value δ, no more than δd will be routed through any network component. This ensures that as soon as $\delta < 1$, any single component failure will cause at most one path per demand to fail, corresponding to at most δd of the demand value. Thus, at least $(1 - \delta)d$ of the demand is still routable. The drawback of this method is that at least $\lceil 1/\delta \rceil$ paths are needed to route a given demand; usually, this leads to using relatively long paths, which in turn increases the probability that such a path is affected by a component failure. This technique is often combined with hop limits. Note that a diversification parameter $\delta = 1$ effectively means no diversification.

1+1 protection:

Using this protection scheme, every signal is routed on two edge- or node-disjoint paths simultaneously, and the target node for which the signal is dedicated chooses the signal of better quality. In particular, if one of the two paths fails, the signal is still completely routed on the other path. Despite the very high cost in terms of needed capacity (more than twice the minimum working capacity, as backup paths are often longer than the primary paths), this method is widespread in practice. This is in particular due to the simplicity of the concept and to the fact that in case of single component failures (which occur much more often than multiple component failures), no demand is lost or even interrupted for a short time. These advantages make the application of this method attractive for critical parts of a network, or when dealing with ring topologies.

Dedicated backup paths:

This technique is also known as *1:1 protection* and is basically like 1+1 protection, except that one of the two paths for each demand is used only when the other path fails. The precomputation of the backup paths ensures that in case of a failure, the switching can be accomplished very fast, in the order of 50 milliseconds.

Summarizing these protection methods, we see that they aim at different targets:

- Topological considerations serve as a basis for other survivability techniques.
- Path length restrictions and diversification only reduce the average amount of demand affected by a component failure and cannot always guarantee the reroutability of any demand.
- 1+1 protection or dedicated backup paths guarantee the routability of demands in some failure states, at the expense of much higher costs.

2.1.2 Restoration

As opposed to protection strategies, which impose constraints on the NOS flow, *restoration* methods ensure survivability by spreading spare capacity over the network such that every affected demand can be rerouted in case of failure; often, the actual routing is only determined at runtime when the failure occurs.

The level of restorable demand can be very high (up to 100%), as with 1+1 protection. At the same time, spare capacity can be shared among different backup paths, which yields potentially much lower capacity cost compared to protection schemes. The major drawbacks of restoration are that runtime reconfiguration in case of a failure takes more time than with protection, and that restoration usually leads to more paths per demand being used in NOS than with protection.

There are several restoration techniques, each of which is distinguished by the amount of flow that is rerouted in the case of failure. The most common ones are *Link Restoration* and *Path Restoration*; more rarely used techniques include *Reservation* and mixtures of Link and Path Restoration like the *Meta-mesh* concept. We will briefly review these methods and discuss their assets and drawbacks.

For the remaining of this section, we assume that a base network together with a NOS flow satisfying certain demands is given (see Figure 2.1). Each demand may be routed on several paths; we call a demand *affected* if at least one of its paths is interrupted by a node or edge failure.

In Figure 2.1, there are two demands: from node A to node E with value 2 and from node B to node F with value 4. The latter demand is routed on two paths in NOS. The capacities are not included in the picture and are assumed to be sufficient for all considered routings, just to illustrate the principles of restoration.

Reservation:

This is the technical term for a global reconfiguration in case of a failure, i.e., a completely new routing is computed for the remaining network, independent of which paths are affected. This restoration technique was the first one considered in literature (by Minoux [Min81]); the

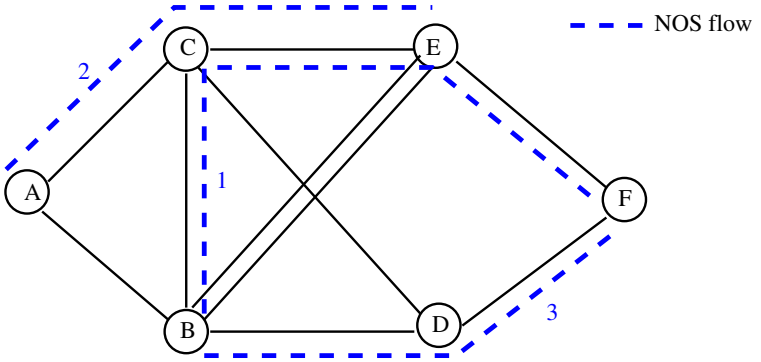

Figure 2.1: Base network with NOS flow

name *Reservation* reflects the fact that some extra capacity is reserved for failure states only. While this method is very cost-effective in terms of spare capacity, it prevents any traffic to be routed in the network for some seconds during the reconfiguration. As this is unacceptable for a network provider in most cases, the application of this technique is of rather theoretical interest. Its asset is that it provides a lower bound on the network cost using one of the other restoration techniques, as will be shown.

Path Restoration:

Using Path Restoration, only the interrupted NOS flow is rerouted from source to target of the interrupted paths. The unaffected NOS routing paths are not touched. As this is a special case of global rerouting (cost is minimized only over the routings where the unaffected paths are kept fixed, instead of minimizing cost over all routings), this method cannot yield a lower cost than Reservation.

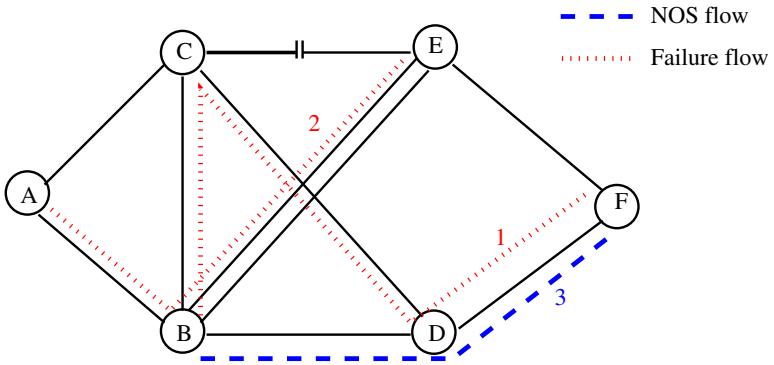

Figure 2.2: Path Restoration, failure of edge {C,E}

Figures 2.2 and 2.3 illustrate this method. Upon failure of node C or of edge {C,E}, the two affected NOS paths A → C → E and B → C → E → F (see Figure 2.1) are rerouted from source to target. The third NOS path is not affected by the failure and is not changed.

A variant of this technique is *Path Restoration with stub release*: prior to rerouting the affected demands, the capacity used by interrupted paths is released as it is not used any

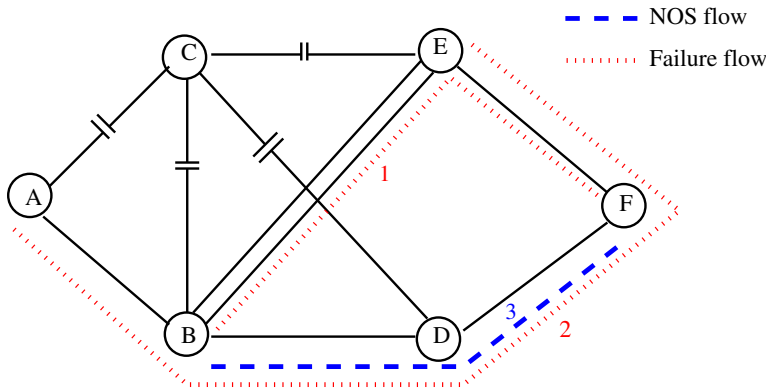

Figure 2.3: Path Restoration, failure of node C

more. This can reduce the needed spare capacity since existing working capacity can be reused.

Link Restoration:

This local rerouting scheme is also called *Span Restoration*, has been introduced by Wu [Wu92] and was initially designed for edge failures only.

Upon failure of an edge $e = \{u, v\}$, every affected path is rerouted between the two end nodes u and v of e. In other words, a new demand is created between u and v, with a demand value equal to the total NOS flow on e. Since this demand is routed between the end nodes of the failing edge, the resulting (local) paths can be used to restore the interrupted (global) paths by concatenation of the local failure path with the parts of the NOS paths that did not fail.

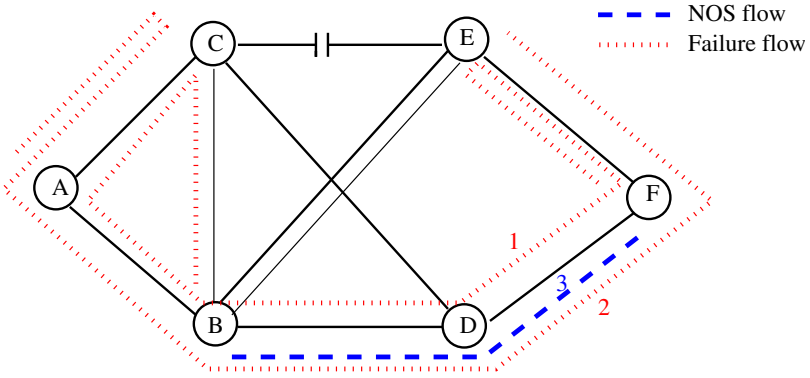

Figure 2.4: Link Restoration, failure of edge {C,E}

This principle is illustrated in Figure 2.4. A failure of edge {C,E} affects two NOS demands (see Figure 2.1). A local demand is created from C to E with value 3 (= total NOS flow on {C,E}). This demand can be routed over the (local) failure path C → A → B → D → F → E. This failure path is concatenated with the parts of the affected NOS paths that did not fail (A → C for the first demand, B → C and E → F for the second one). This leads to the

backup paths A → C → A → B → D → F → E and B → C → A → B → D → F → E → F, respectively. The unaffected NOS path B → D → F is not rerouted.

As can be seen in Figure 2.4, Link Restoration often suffers from the so called *backhauling* problem: a backup path contains loops, which leads to unnecessary additional capacity requirements (for instance, edge {A,C} would not need any spare capacity if these loops were eliminated). The reason for backhauling is that upon calculation of the failure paths, no information about the affected NOS paths is used, except that they were routed over the failing edge. In theory, is it of course possible to eliminate these loops in a postprocessing step, but in practice, this is not always desired. For instance, the sender-chooser concept as described in [SNH90] actually leads to backup paths with loops in practice.

Upon failure of a node v, a new demand is created between every pair of neighbour nodes u, w of v where there is some NOS flow on the subpath $u \to v \to w$. The value of this NOS flow is the demand value of the new demand.

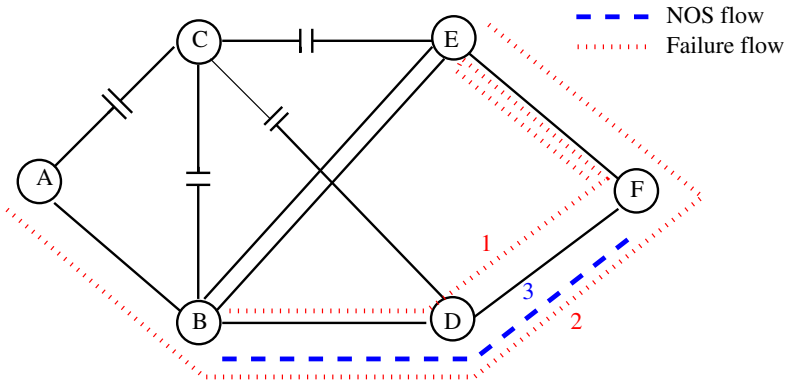

Figure 2.5: Link Restoration, failure of node C

In Figure 2.5, upon failure of node C, a new (local) demand from A to E and a new demand from B to E are created for the two affected NOS flow paths. These demands (with values 2 and 1) are routed over the paths A → B → D → F → E and B → D → F → E, respectively. Afterwards, these failure paths are concatenated with the parts of the NOS paths that are still usable (the empty path for one demand, E → F for the other one). Again, the unaffected path B → D → F is not touched. As can be seen in Figure 2.5, the backhauling problem also appears with node failures.

Observe that every feasible Link Restoration routing is also a feasible routing for Path Restoration, but not vice versa (in both cases, only the affected paths are rerouted, but with Link Restoration, the backup paths are fixed to the corresponding NOS paths on the intact part of the network). This directly implies that Link Restoration (with routing paths considered from end to end of a demand) is as least as expensive as Path Restoration. Note that when capacity is reserved not from end to end of a demand but only for the NOS paths and the *local* failure paths (as it is usually done in the literature), Link Restoration is still as least as expensive as Path Restoration with stub release but may be cheaper than Path Restoration without stub release. As our computational tests showed, this actually occurs in practice.

Note: Since we also consider the extension of Link Restoration to node failures in this paper, the name *Local Restoration* (as opposed to *Global Restoration* schemes like Reservation or Path Restoration) could seem more appropriate in our case. However, as the designation *Link Restoration* is more common in literature, we will use the this term for the described restoration mechanism.

There is another reason for doing this: many of the issues discussed in this thesis suggest the following classification of restoration techniques:

- a *Global Restoration scheme* reroutes affected paths from end to end (like Path Reservation or Reservation), whereas
- a *Local Restoration scheme* reroutes only parts of the affected paths (like Link Restoration).

Hence, we are going to reserve the terms *Local* and *Global* Restoration to these classes of restoration mechanisms.

Meta-mesh:

This is in fact a modification of Link Restoration (introduced in [DG01b]). The basic idea is to combine degree-2-chains (subpaths whose nodes are all of degree 2) to a single edge and to apply Link Restoration only between nodes of degree 3 or more. The effect is that the backhauling problem is avoided to a great extent which in turn reduces the needed spare capacity on degree-2-chains. This technique is mentioned here as a representative for various restoration mechanisms that work neither as locally as Link Restoration nor as globally as Path Restoration. However, given that affected paths are rerouted locally rather than end-to-end, Meta-Mesh is a Local Restoration scheme.

Comparison of Link and Path Restoration

As already mentioned, Link and Path Restoration are the most widely used restoration techniques. Given that this work focuses on these methods as well, we take a closer look at them now.

The major drawback of Link Restoration is that it may lead to relatively high network cost, especially if an NOS routing is already given and only spare capacity cost is minimized. Apart from the backhauling problem (see Page 16, this is mainly due to the fact that with Link Restoration, the choice of failure paths is quite limited compared to Path Restoration since a major part of the restoration paths is already fixed.

However, Link Restoration has some advantages:

- In case of a single link failure, only one failure demand is created with Link Restoration, so one has to deal with a single commodity flow problem. In contrast, Path Restoration causes the creation of one failure demand per NOS demand even upon a single link failure, which leads to a multicommodity flow problem and is more difficult to handle. Upon node failures, both methods lead to multicommodity flow problems, as explained above.
- The fact that only local information is required for Link Restoration makes the restoration process simpler and faster.

As we found out in our computational tests, the cost savings of Path Restoration compared to Link Restoration are often not as important as expected when only few installable capacities per edge are available and when economies of scale are considered. Details can be found in Chapter 6.

Note that Link Restoration in the presented form is limited to single failures: otherwise, imagine a path $u \to v \to w$ where both the edges $\{u, v\}$ and $\{v, w\}$ fail at the same time. Which failure demands should be created? If there is no other path from u to v, the Link Restoration problem has no solution since the failure demand from u to v cannot be satisfied. On the other hand, it would make sense to create a failure demand directly from u to w, slightly modifying the Link Restoration concept. In any case, the concept of Link Restoration would have to be adapted when considering multi-failures.

2.2 Related work

2.2.1 Problem variants

During the last decade, a lot of effort has been spent on the field of dimensioning survivable networks. As a consequence, many variants of the network dimensioning problem have emerged, for instance:

- Capacities can be continuous or discrete. Continuous capacities avoid the inherent difficulties associated with integer variables and facilitate solving the problems in adequate time despite the large scale of the resulting LPs.

  With discrete capacities, there are several variants:

  - capacities may be allowed to take any integer value,
  - they may be assumed to be all multiples of some single base capacity (with linear cost),
  - they may be assumed to be *divisible* (see below),
  - a finite set of installable capacity values can be given for each edge, each with a given cost.

  The divisibility assumption states that after ordering the available capacities increasingly, each capacity divides the next one. This assumption is valid in many (but not all) cases in practice. For instance, common link capacity types are STM-1, STM-4, STM-16 and STM-64 (where one STM-N channel provides a capacity of $N \times 155$ Mbps) and OC-1, OC-3, OC-12, OC-24 and OC-48 (where one OC-N channel corresponds to $N \times 51.84$ Mbps).

  Also the base capacity assumption is of course feasible from a computational point of view by declaring the greatest common divisor of all capacities to be the base capacity.

  However, as with continuous capacities, the first three discrete approaches assume the cost to be a linear function of the capacity; unfortunately, capacity cost is often far from being linear in practice, which can only be reflected by the last approach. Very often, the cost per capacity unit decreases with increasing capacity; these economies of scale can be quite considerable in practice (see also Chapter 6).

- Different failure states can be investigated. Most publications consider single edge failures, some also single node failures, very few consider multiple component failures.

- The base network and an NOS flow may be given, in which case the problem reduces to determining the spare capacities, or the base and the spare capacities may be determined at the same time. While the latter approach needs much more calculation efforts than the former, it is obviously more cost effective. As the research done by Iraschko et al. [IMG98] and other authors suggests, the cost-savings of a joint optimization of working and spare capacity can be quite important in real world problems. This could also be verified for the networks investigated in this work (see Chapter 6).

- Additional constraints may be imposed, like hop limits, restrictions on the number of paths to be reconfigured in case of failure or on the maximum number of paths per demand, diversification, and many more. These restrictions are often applied together with Path Restoration as they fit well into the existing LPs in this case, or with no restoration at all to ensure a minimum level of survivability.

- Some recent studies (for instance, [LOVG99] and [RA00]) investigate stochastic demands, i.e., a probability distribution on the demands is given instead of fixed demand values. This seems to be a useful approach, provided that the assumptions on the probability distribution are at least as realistic as those on the demand values (or even better, to compensate the additional effort for the more complicated models).

2.2.2 Literature on restoration of single failures

The vast majority of publications on survivable network design considers single component failures only. There are two main reasons for this: First, this approach often simplifies models and implementation to a great extent, and second, it is considered unlikely in practice that more than one network component at a time fails.

Essentially, there are two popular ways of modeling the network dimensioning problem with single network component failures: using a Cut LP or a Path LP.

A Cut LP contains an inequality for every cut in the network, stating that the flow on a cut must not exceed its capacity. While the number of cuts is exponential in the number of nodes, these cut inequalities may quickly be separated using a MaxFlow algorithm. The number of variables in a Cut LP is usually bounded by a polynomial in the number of nodes and edges. Cut LPs are often used in investigations of Link Restoration with single link failures.

On the contrary, a Path LP approach leads to a relatively small number of constraints but exponentially many variables (at least one for each path in the network), making it necessary to employ column generation. The major advantage of a Path LP is that it allows more convenient and flexible formulations of Path Restoration with additional constraints (such as length restrictions or diversification) than a Cut LP. Consequently, Path LPs are often used to model Path Restoration or Reservation or when considering node failures.

Table 2.1 lists some existing papers on restoration of single component failures together with the investigated problem variants and the methods employed by the authors. Given the huge amount of literature on restoration, only a small part of it can be shown in this overview, but it should nevertheless give the reader an idea of the existing literature on this topic.

Paper	Failures		Restoration			NOS given?		Model, methods, special features
	link	node	LR	PR	Res.	yes	no	
[AGW96]	✓	✓	-	-	✓	-	✓	Path MIP, hop limits, diversification, CPA, heuristics
[AGW97]	✓	✓	-	-	✓	-	✓	Path MIP, hop limits, divisible capacities, diversification, CPA, B&B and heuristics
[DS98]	✓	✓	-	-	✓	-	✓	LP with all metric inequalities, CPA, heuristics
[IMG98]	✓	-	✓	✓	-	✓	✓	Path MIP, integer flows
[Xiong98]	✓	✓	-	✓	-	✓	✓	Path MIP, hop limits, one base capacity
[PD98]	✓	✓	✓	✓	-	✓	-	Path LP, continuous capacities
[Wess00]	✓	✓	-	✓	✓	-	✓	Path MIP, CPA, B&C, diversification, hop limits, heuristics, polyhedral study
[XM97]	✓	-	✓	✓	✓	-	✓	Path LP, continuous capacities, hop limits, heuristics
[BMSW98]	✓	-	✓	-	-	✓	-	Cut MIP, one base capacity, CPA, B&B
[BMSW00]	✓	-	✓	-	-	✓	-	Cut MIP, two heuristics
[MW97]	✓	-	✓	-	-	✓	-	Cut MIP, polyhedral study, no computational results
[SNH90]	✓	-	✓	✓	-	✓	-	Cut LP + rounding, NETRATS algorithm
[Wang01]	✓	-	✓	-	-	-	✓	Cut MIP, B&C

Table 2.1: Literature on restoration of single component failures

Abbreviations: LR means *Link Restoration*, PR means *Path Restoration*, Res. means *Reservation*, CPA means *Cutting plane algorithm*, B&B means *Branch and Bound*, B&C means *Branch and Cut*.

The large-scale Mixed Integer Programs appearing in survivable network design problems are inherently difficult to solve. Due to this fact, in addition to exact solution methods (using Branch and Bound, for instance), a plethora of heuristics exists in the literature on this subject, from solving the LP relaxation and rounding up the resulting values (e.g., [SNH90]) to more sophisticated ones (e.g., [AGW97], [BMSW00], [DS98]). In some cases, these heuristics are employed as the only solution method, but in most cases they support a Branch and Bound or Branch and Cut algorithm.

2.2.3 On other aspects of network design

While the papers presented in Table 2.1 focus on the solution of single component failure problems, there are a lot of other interesting aspects of network design, like polyhedral studies (which in turn can be used in the solution process), dual failure analysis, different restoration protocols, etc. Some papers dealing with these subjects will be presented now; in addition, Soriano et al. [SWSLGF98] give a good literature survey and propose a useful introduction to optical network terminology to non-expert readers.

Polyhedral studies:

There are quite a lot of polyhedral studies of the network design problem, some of them considering only the NOS, others considering various restoration settings. Here are some of these studies:

Atamtürk [Ata00] investigates cut-set inequalities for the network design problem with single and multiple facilities and commodities and completely describes the cutset polyhedron for the single facility / single commodity case (where only one commodity is considered and only one type of capacity is installable).

Dahl and Stoer [DS93] discuss many valid inequalities for the network design problem, including metric inequalities, (strengthened) band inequalities, partition inequalities, lifted 2-Cover-Inequalities and Q-Subset-Inequalities, partially with facet proofs.

Grötschel, Monma and Stoer [GMS95] address the survivability problem by connectivity requirements and present polyhedral results, structural properties and heuristics for the problem. They investigate some polynomially solvable special cases and give computational results on random and real world problems.

Bienstock and Muratore [BM97] present valid inequalities for link and node failures using an abstract but simple model that arises in many contexts of network planning.

Magnanti and Wang [MW97] investigate polyhedral properties of the network design problem with a single base capacity, present a facet proof for the Q-Subset-Inequalities and completely describe the polytope of the parallel path network restoration problem, which arises naturally as a subproblem in many network design problems.

Wessäly [Wess00] introduces extensions of (strengthened) metric inequalities, band, cut and partition inequalities for use with diversification in different restoration models. Furthermore, another extension of band inequalities is introduced, the k-band inequalities. For some of these inequalities, facet proofs are given.

Restoration protocols:

Sakauchi, Nishimura and Hasegawa [SNH90] describe the NETRATS[1] algorithm for Link and Path Restoration in networks with DCS[2] nodes and introduce the sender-chooser concept for local restoration of interrupted demands.

Chow, McCaughey and Syed [CMS93] propose a survey on different restoration protocols. They compare these protocols in terms of restoration speed and implementation complexity and give extensive numerical results.

Comparisons of different restoration techniques:

Xiong and Mason [XM97] compare spare capacity assignment with given NOS using Link Restoration, Path Restoration and Reservation. They consider continuous capacities and apply heuristics to Path Restoration.

Iraschko, MacGregor and Grover [IMG98] treat single edge failure situations using Link Restoration or Path Restoration (with or without stub release), all with or without given NOS capacities and flow, and compare the total network cost induced by these models. They allow capacities to be arbitrary positive integers, the cost being proportional to the amount of installed capacity. Unlike most authors, they also require the flow to be integer.

Poppe and Demeester [PD98] investigate a Path LP (with continuous capacities) with given NOS flow and present a multicommodity flow formulation for link and node failures using Link or Path Restoration. They provide a detailed comparison of spare capacity cost using either of these techniques for different failure scenarios (only single link failures, only single node failures or all single component failures). One surprising result is that the total spare capacity needed to restore demands upon single edge failures often significantly exceeds the one needed to restore demands upon single node failures. This is probably due to the fact that in case of a node failure, the demand emanating from the failing node is lost and no more considered in the corresponding failure state; this effect seems to be quite important.

Doucette and Grover introduce the Meta-Mesh concept [DG01b] and compare different protection and restoration schemes with respect to their dependency on graph connectivity [DG01a].

Wessäly [Wess00] proposes models for Reservation and Path Restoration with arbitrary discrete or divisible capacities and with diversification and length restrictions. No NOS routing nor NOS capacities are assumed to be given.

Dual edge failures:

Clouqueur and Grover [CG01] analyze the resistance with respect to dual edge failures of networks designed for single edge failures using Path Restoration. Their results indicate that in many cases, a considerable amount of flow can be recovered even if two links fail at the same time (provided that the network is sufficiently dense).

Lumetta and Médard [LM01] introduce dual link failure vulnerability measures and classify the possible dual failure states with respect to their frequency and their impact on restorability. Different protection and restoration schemes are compared with respect to these measures in networks of different density.

[1] NETwork Restoration Algorithm for Telecommunication Systems
[2] Digital Cross-connect Switch

2.2.4 Computational complexity

The network restoration problem is already $\mathcal{NP}$-hard in very special cases. For instance, Chopra et al. [CGS98] showed that already the capacity assignment problem for the NOS without restoration requirements or hop limits, with two installable capacities on each edge and one demand is $\mathcal{NP}$-hard.

The same result yields that also the spare capacity assignment problem with given NOS capacities and discrete capacities is $\mathcal{NP}$-hard, since it decomposes into a sequence of problems, one for each failure state and very similar to the corresponding problem for the NOS. With path flow variables, the resulting LP contains a demand constraint for each demand and a capacity constraint for each edge (see also Section 3.3.6); with edge flow variables, the demand constraints are substituted by flow conservation constraints.

Assume that NOS capacities and a NOS flow are given and that spare capacity is to be minimized under the following assumptions:

- On every edge, we may install either nothing or exactly one capacity unit as spare capacity (at cost 1). The hardware at the nodes does not induce any restrictions or cost.
- The NOS flow and installed NOS capacity on every edge are 1.
- Only single edge failures with Link Restoration are considered.

As Balakrishnan et al. [BMSW98] showed by reduction to the Hamilton path problem, the decision problem corresponding to this network restoration problem is strongly $\mathcal{NP}$-complete.

2.2.5 How this work adds to the existing literature

To the best of our knowledge, no paper has been published yet that covers all common restoration techniques in one mathematical model while jointly optimizing working and spare capacity with arbitrary capacities. In nearly every paper, completely different models are applied to Link and Path Restoration. In this thesis, we will show that all the restoration techniques explained in Section 2.1.2 can be modeled in a common way, the difference between them being hidden in the definition of some commodities and in one coefficient of capacity constraints. Another asset of our model is that it also covers hardware requirements at the nodes, which is considered in very few papers.

Especially two papers served as a basis for our mathematical model and for the issues investigated in this thesis:

- In [PD98], a multicommodity flow formulation for single network component failures is presented that abstracts from node or link failures and applies to Link Restoration as well as to Path Restoration with or without stub release. This served as a basis for the formulation of failure flows in this thesis. However, the authors made two simplifying assumptions: they assumed continuous capacities, which allowed them to solve linear programs instead of mixed integer programs, and they minimized spare capacity cost based on a given NOS routing, which allowed them to consider each operating state separately.
- In [Wess00], capacities are given as a finite set of installable capacities for each edge together with an arbitrary capacity cost. This cost model has been adopted in this

thesis. The NOS and failure states are optimized in one step, using Path Restoration or Reservation. This approach is combined with the protection mechanisms diversification and hop limits.

In our computational results, we compare optimal values of some real world networks for Link Restoration and Path Restoration (with or without stub release) with different failure scenarios.

Though several such comparisons between various restoration techniques are available in literature, all such comparisons known to us are either only aimed at spare capacity minimization with given NOS, are limited to single link failures, assume continuous capacities, or some heuristic values are compared with each other instead of optimal values.

Given that the quality of heuristic solutions is rarely known, the latter approach cannot compare inherent properties of the investigated networks but only the quality of different heuristics (since usually different heuristics are applied to different restoration techniques).

Furthermore, we investigate the effects of a discrete cost structure on real world networks. As shown in Chapter 6, these effects can lead to quite counterintuitive results.

For large or dense networks where we could not at all or only in special cases find an optimal solution, a comparison of several heuristics is given w.r.t. the obtained solutions and the required solution time. The same heuristics were applied to all restoration scenarios. As a side result, the cost savings of a joint optimization of NOS and spare capacity compared to successive optimization are investigated.

Chapter 3

Mathematical model

In this chapter, our mathematical model for the restoration problem will be introduced. The model consists of two main parts, linked by the edge capacities: a linear problem (HLP) modeling the hardware aspects of the problem (hence the 'H'), and a linear program (RLP) covering the restoration aspects (hence the 'R'). In addition, there are integrality constraints for some variables in (HLP) which effectively turn it into a mixed integer program. These parts will be presented after some necessary preliminaries and introduction of notational conventions. Figure 3.2 shows the whole model at a glance.

3.1 Preliminaries

3.1.1 Underlying graphs

We consider two (directed or undirected) graphs with the same node set V:

The telecommunication network: The considered telecom network will be denoted by $G = (V, E)$ and will often simply be referenced by *the network*. For instance, the nodes can be cities, buildings or computers in a network, whereas the edges represent connections between these nodes by fiber cables or radio contact. For each edge, a certain set of installable capacities is given (see Section 3.2 for details).

The demand graph: The demand graph $H = (V, D)$ contains the same nodes as G. Two nodes are connected by an edge in H if and only if there is a communication demand with positive demand value between them.

3.1.2 Preconditions

There are a some assumptions on the input of the model which have to be ensured by preprocessing:

1. Although our model allows arbitrary operating states, the network must be sufficiently connected to allow routings in all considered operating states for the problem to be feasible. For instance, to allow restoration of every single network component failure, the (potential) network has to be at least twice node-connected. In other words, every

pair of nodes must be connected by at least two node-disjoint (and thus edge-disjoint) paths.

Given that real telecommunication graphs are often not biconnected, an appropriate subset of operating states has to be specified that is considered for restoration.

2. Either the network, the demand graph, paths and network flows are all undirected, or all of them are directed. If everything is undirected, flows in different directions on the same edge do not interfere. In other words, the direction of the flow does not really matter, and an edge with capacity 3 may for instance accommodate one unit of flow in one direction and two units in the other direction. This assumption is often valid in practice.

3. Without loss of generality, all installable capacities are assumed to have nonnegative integer values (this is nearly always fulfilled in practice). This will be used for some rounding procedures when considering valid inequalities.

4. Once capacity is installed, there is no additional routing cost.

5. A demand may be routed on more than one path.

6. Demands from a node to itself are not allowed. We also disallow looped routing paths since a path containing loops may always be substituted by a path without loops between the same end nodes. The path without loops will never be more expensive in terms of required edge capacity than the looped path.

7. The network is allowed to contain parallel edges. We cannot aggregate parallel edges to a single edge, as in case of failure of one of these edges, we can still use the other edge to restore the interrupted flow. Real life networks sometimes actually contain parallel edges.

8. The demand graph is assumed to be simple, i.e., parallel demands are not allowed. This can be assumed without loss of generality by aggregating parallel demands into a single demand since a demand may be routed on several paths.

3.1.3 Notation

Given that notation varies widely in literature when dealing with graphs, this section shortly introduces some notational conventions used in this work. In most cases, the meaning of a notation will be clear from the context; as a short reference, see also the table of symbols on Page 87.

Edges: The notation (u, v) may refer to an ordered pair of nodes (which need not be connected by an edge) in the network G. More often, it will denote the directed edge from u to v (if there is only one of them) or the whole set of directed edges from u to v if there are several of them and it does not matter *which* edge from u to v is currently considered. The actual meaning will always be clear from the context.

In a similar way, the notation $\{u, v\}$ refers to an unordered pair of nodes or to the set of undirected edges between nodes u and v.

Path: A *path* in a network is considered to be a set of edges (directed if and only if the network is directed). However, for notational convenience, we will sometimes also write $v \in P$ for some node $v \in V$, with the obvious meaning. A path without node repetitions (and thus without edge repetitions) is called a *simple path*. The three-node-path $s \to w \to t$ refers to the undirected path $\{\{s, w\}, \{w, t\}\}$ if the network is undirected.[1]

Capacity: Our model does not distinguish between working and spare capacity. The *capacity* of an edge is always the sum of the working and the spare capacity.

Demands: For ease of notation, a demand from node u to v will in most cases be denoted by uv. For an undirected network, uv is a shortcut for the undirected demand $\{u, v\}$, whereas for directed networks, uv stands for the directed demand (u, v).

3.2 The hardware model

This section explains the used MIP formulating the hardware requirements, independent of any routing or restoration conditions. This MIP is a special case of a more general hardware model developed by Kröller [Krö03].

3.2.1 Parameters

Several hardware components have to be taken into account: node designs, edge designs, and modules, which are connected by each other via slots and interfaces. We will shortly explain how these parts work together; Figure 3.1 illustrates the connection between these hardware components.

A node in the network may be equipped by a *node design*, providing a maximum capacity which the node is able to switch between the adjacent edges. A node design offers some *slots* which can be equipped with *modules* (e.g., cards which can be plugged in). Each module may occupy one or more of these slots. A module provides some *interfaces* which in turn are needed to accommodate *edge designs*. An edge design can be installed on an edge and provides a capacity which is an upper bound for the flow on this edge.

The following hardware parameters are assumed to be given:

Node designs A node $v \in V$ can be equipped with at most one node design out of a (finite) set $\mathcal{D}(v)$ of node designs installable at v. With a node design $d \in \mathcal{D}(v)$ are associated:

- the switching capacity $C^d \in \mathbb{N}$ of the node design,
- the number of provided slots $S^d \in \mathbb{N}$,
- a set of installable modules $\mathcal{M}(d)$,
- for every module $m \in \mathcal{M}(d)$, a number $M_d^m \in \mathbb{N}$ specifying the maximum number of modules of type m that can be supported by node design d, independent of slot restrictions,
- and a node design cost $c_v^d \geq 0$.

[1] Actually, $s \to w \to t$ may refer to a whole set of paths if there are parallel edges, but they may all be considered equivalent in this context.

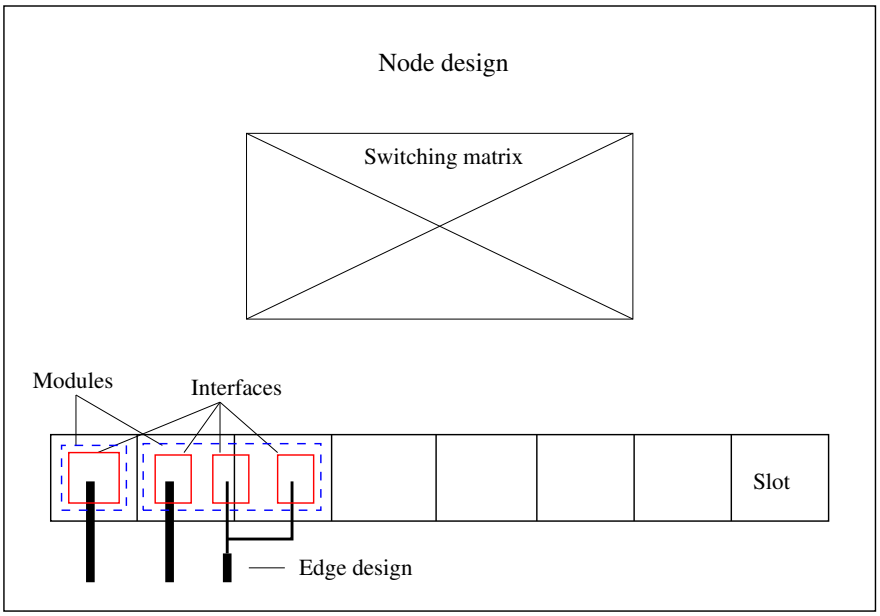

Figure 3.1: Hardware configuration

Modules The parameters associated with a module $m \in \mathcal{M}(d)$ are

- the number of occupied slots $S^m \in \mathbb{N}$,
- a (finite) set of interfaces $\mathcal{I}(m)$ provided by m,
- the number $I_m^i \in \mathbb{N}$ of provided interfaces of type $i \in \mathcal{I}(m)$,
- and a module cost $c^m \geq 0$.

For ease of notation, we will denote by $\mathcal{M}(v) := \bigcup_{d \in \mathcal{D}(v)} \mathcal{M}(d)$ the set of modules installable at a node $v \in V$. Similarly, $\mathcal{I}(v) := \bigcup_{m \in \mathcal{M}(v)} \mathcal{I}(m)$ denotes the set of interfaces potentially available at a node $v \in V$.

Edge designs For each edge $e \in E$, a (finite) set of installable edge designs $\mathcal{D}(e)$ is given, out of which at most one may be chosen for installation on the e. Associated with each edge design $d \in \mathcal{D}(e)$ are

- the routing capacity $C^d \in \mathbb{N}$ of the edge design,
- a (finite) set of needed interfaces $\mathcal{I}(d)$,
- the number $I_d^i \in \mathbb{N}$ of interfaces of type $i \in \mathcal{I}(d)$ needed by d,
- and an edge design cost $c_e^d \geq 0$. This cost may for instance be composed of a fixed part and a length dependent part.

3.2.2 Variables

There are binary variables for the node and edge designs, and integer variables for the modules:

- Node design variables: For each $v \in V$, $d \in \mathcal{D}(v)$, a binary variable x_v^d is introduced, indicating whether node design d is installed on node v or not.
- Edge design variables: Similarly, a binary variable x_e^d indicates whether edge design $d \in \mathcal{D}(e)$ is installed on edge e or not. Since at most one edge design may be installed on a link (i.e., at most one of the x_e^d has to be 1 and all others have to be 0), the capacity of edge e is
$$y_e := \sum_{d \in \mathcal{D}(e)} C^d x_e^d.$$
- The nonnegative integer variable $x_v^m \in \mathbb{N}$ gives the number of modules of type m installed at node v, for every node $v \in V$ and every module $m \in \mathcal{M}(v)$.

3.2.3 Inequalities

Given the parameters and variables, we are now in a position to formulate the necessary side constraints that must be satisfied by every feasible hardware configuration.

1. *GUB constraints on the node and edge designs:*[2] At most one node design must be chosen on each node; the same applies to edge designs:
$$\sum_{d \in \mathcal{D}(v)} x_v^d \leq 1 \quad \forall\, v \in V \tag{3.1}$$
$$\sum_{d \in \mathcal{D}(e)} x_e^d \leq 1 \quad \forall\, e \in E \tag{3.2}$$

2. *Capacity constraints:* The capacity of a node must be sufficient to switch all the capacity of its incident edges:
$$\sum_{d \in \mathcal{D}(v)} C^d x_v^d \geq \sum_{e \in \delta(v)} y_e \quad \forall\, v \in V. \tag{3.3}$$

3. *Slots restrictions:* The number of slots provided by a node design must not be exceeded by the slot requirements of the installed modules:
$$\sum_{d \in \mathcal{D}(v)} S^d x_v^d \geq \sum_{m \in \mathcal{M}(v)} S^m x_v^m \quad \forall\, v \in V. \tag{3.4}$$

4. *Interface restrictions:* The number of interfaces of type i provided by modules installed at a node v must suffice to accommodate the number of interfaces of type i required by the incident edges:
$$\sum_{m \in \mathcal{M}(v)} \sum_{i \in \mathcal{I}(m)} I_m^i x_v^m \geq \sum_{e \in \delta(v)} \sum_{d \in \mathcal{D}(e)} I_d^i x_e^d \quad \forall\, v \in V,\ \forall\, i \in \mathcal{I}(v). \tag{3.5}$$

[2] *GUB* stands for *generalized upper bound* and describes a constraint of the form $\sum_{i \in I} x_i \leq k$ with binary variables x_i, stating that at most k items in I may be chosen.

5. *Module bounds:* Independently of slot restrictions, the maximum number of allowed modules of type m must not be exceeded at any node:

$$\sum_{d \in \mathcal{D}(v)} M_d^m x_v^d \geq x_v^m \quad \forall\, v \in V,\ \forall\, m \in \mathcal{M}(v). \tag{3.6}$$

3.2.4 Objective function

The objective is minimizing total installation cost, i.e.,

$$\min \quad \sum_{v \in V} \Big(\sum_{d \in \mathcal{D}(v)} c_v^d x_v^d + \sum_{m \in \mathcal{M}(v)} c^m x_v^m \Big) + \sum_{e \in E} \sum_{d \in \mathcal{D}(e)} c_e^d x_e^d \tag{3.7}$$

Observe that without further constraints, it is perfectly feasible to install nothing, yielding zero cost. However, further requirements will be given by the conditions on the routing in the next section.

3.3 The restoration model

This section introduces the constraints and variables formulating the restoration aspects. This part of the model ensures that the NOS demands are satisfied and that all restoration requirements are met by the edge capacities. In fact, this LP acts as a feasibility tester for edge capacities and as a separator for metric inequalities, as will be explained in detail in Chapter 4.

Notational conventions: While our notation will be explained when used, some notational conventions have been adopted throughout this thesis which may give the reader some orientation. The variable g will nearly always refer to a failing edge, whereas w will be reserved for a failing node. The nodes s and t are usually adjacent to a failing node w (with Link Restoration), while u and v refer to the end nodes of a path or of a NOS demand.

3.3.1 Parameters

Operating states: S is the set of considered operating states. This set has to be chosen appropriately according to several parameters:

- The connectivity of the network: for instance, the failure of an articulation node or a bridge (i.e., a node or edge whose removal would disconnect the network) causes the restoration problem to be infeasible.
- The desired level of security against failures. This level may be different for various parts of the network as some nodes ore edges may be more important than others.
- The type of restoration; with Link Restoration in the presented form, only single component failures can be considered.

A failure state can be characterized by its failing components (not including secondary failures, for instance the adjacent edges of a failing node), so we will for example simply denote by $w \in S$ the operating state in which only node w fails (inducing the failure of its adjacent edges). The NOS is usually denoted by 0.

Demand values: $d_{uv} > 0$ is the value of demand $uv \in D$. If the network is undirected, $d_{uv} = d_{vu} > 0$ refers to the value of the undirected demand between u and v.

Restoration level: $\rho^s_{uv} \in [0, 1]$ indicates the fraction of interrupted NOS flow for demand $uv \in D$ that should be restored in failure state $s \in S \setminus \{0\}$. In particular, restoration of all interrupted flow corresponds to $\rho^s_{uv} = 1$ for all demands $uv \in D$.

Hop limits: ℓ_{uv} gives the maximum allowed number of edges of a uv-path P used in a NOS routing.

3.3.2 Notation

Usable Edges: $V^s \subseteq V$ denotes the set of non-failing nodes in operating state $s \in S$. Similarly, $E^s \subseteq E$ denotes the set of usable edges in operating state $s \in S$. An edge is usable if neither the edge itself nor one of its end nodes fail.

Failure commodities: Recall that with Link Restoration, a local failure commodity is created between the end nodes of the failing edges or between each pair of neighbours of a failing node, respectively. With Path Restoration, a failure commodity is created for each affected NOS demand. In either case, some set C^s of failure commodities is newly created in each operating state $s \in S$. These failure commodities are node-to-node demands uv (directed if and only if the network is directed).[3] See Section 3.3.4 for more explanation and examples.

Paths:

- $\mathcal{P}$ denotes the set of all nonempty simple paths in $G = (V, E)$. In particular, we disallow loops for the routing paths (see Section 3.3.6 for a note on this restriction). The set of uv-paths is denoted by $\mathcal{P}_{uv} \subseteq \mathcal{P}$.

- By $\mathcal{P}^s_{uv} \subseteq \mathcal{P}$ we refer to the set of uv-paths valid in operating state $s \in S$. A path is *valid* if it consists only of usable edges and satisfies all other side constraints, like hop limits ($|P| \leq \ell_{uv}$ for all $P \in \mathcal{P}^0_{uv}$) and simplicity.

  Similarly, $\mathcal{P}^s_c$ is the set of paths between the end nodes of $c \in C^s$ that are usable in operating state $s \in S$.

  We assume that any valid NOS path is also valid in every other operating state as long as all of its edges or nodes are usable. In other words, we exclude side constraints like length restrictions in failure states but not in NOS. In addition, we assumed in our computational tests that there are no hop limits in failure states.

- We will write $\mathcal{IP}^s_c$ for the set of NOS paths for whose restoration in operating state s the failure commodity $c \in C^s$ is created. One may think of $\mathcal{IP}$ as *interrupted paths*; usually, $\mathcal{IP}^s_c$ consists of some paths interrupted by a failure (at the exception of Reservation). Section 3.3.4 gives some examples.

[3] We write these demands as commodities instead of demands in order to clearly distinguish them notationally from the NOS demands and to abstract from the actual end nodes of such a commodity, which vary among the different restoration techniques.

3.3.3 Variables

Capacity: $y_e \in \mathbb{N}$ again denotes the capacity of $e \in E$. This variable is not actually contained in the LP but serves only to simplify notation.

NOS flow: $f_{uv}(P) \geq 0$ is the NOS flow on path $P \in \mathcal{P}^0_{uv}$ for demand $uv \in D$. Thus, the total NOS flow on edge $e \in E$ is

$$\sum_{uv \in D} \sum_{\substack{P \in \mathcal{P}^0_{uv} \\ e \in P}} f_{uv}(P).$$

Failure flow: $f^s_c(P) \geq 0$ is the extra flow on path $P \in \mathcal{P}^s_c$ in failure state $s \in S$, for failure commodity $c \in C^s$. Note that with Link Restoration, the failure paths P are defined between the end nodes of a failing edge, for instance, and not between the end nodes of some NOS demands.

3.3.4 Some important special cases

To clarify some of the notations used above, some examples will be given now for important special cases, namely NOS, single link failures and single node failures.

NOS

In the NOS (denoted by 0), all edges are usable, which means $E^0 = E$. The set $\mathcal{P}^0_{uv}$ consists of all simple uv-paths satisfying the given length restrictions (and maybe other side constraints), and $C^0 = \emptyset$.

Single link failures

Suppose that edge $g \in E$ fails. In this case, all edges except for g are still usable, as well as all paths not containing g. Thus,

$$E^g = E \backslash \{g\}, \quad \text{and} \quad \mathcal{P}^g_{uv} = \{P \in \mathcal{P}_{uv} \mid g \notin P\} \quad \forall\, uv \in D.$$

Using *Link Restoration*, exactly one failure commodity (or demand) is created between the end nodes of g, whose value is the part of the total NOS flow on g which should be restored. Otherwise stated,

$$C^g_{LR} = \Big\{(s,t,d) \;\Big|\; g = st,\ d = \sum_{uv \in D} \sum_{\substack{P \in \mathcal{P}^0_{uv} \\ g \in P}} \rho^g_{uv} f_{uv}(P)\Big\}.$$

Note that C^g contains only one element, so we have a single commodity flow problem here. The set $\mathcal{IP}^g_c$ for this single commodity c consists of all paths containing the failing edge.

Using *Path Restoration*, a new commodity is created for each affected NOS demand between its end nodes, the new commodity value being the part of the affected NOS flow value for this demand which should be restored. Thus, this case translates to

$$C^g_{PR} = \Big\{(u,v,d) \;\Big|\; \mid uv \in D,\ d = \sum_{\substack{P \in \mathcal{P}^0_{uv} \\ g \in P}} \rho^g_{uv} f_{uv}(P)\Big\}.$$

As the newly created commodities are a subset of the original demands in this case, C^g_{PR} contains at most $|D| \leq |V|(|V|-1)/2$ elements.

Using *Reservation*, the old routing is completely ignored and everything is rerouted. Thus, there is a one-to-one correspondence between the newly created failure commodities and the original NOS demands. Otherwise stated,

$$\mathcal{P}^g_{uv} = \emptyset \quad \forall\, uv \in D, \qquad \mathcal{P}^g_c = \mathcal{P}^0_{uv} \quad \forall\, c = uv \in C^g,$$
$$C^g_{Res} = \Big\{(u,v,d) \;\Big|\; uv \in D,\ d = \sum_{P \in \mathcal{P}^0_{uv}} \rho^g_{uv} f_{uv}(P)\Big\}$$

Single node failures

Suppose that node $w \in V$ fails, inducing the failure of all its incident edges. The still usable paths are those not containing any of the failing edges. Otherwise stated,

$$E^w = E \backslash \delta(w) \quad \text{and} \quad \mathcal{P}^w_{uv} = \{P \in \mathcal{P}_{uv} \mid w \notin P\}$$

Using *Link Restoration*, for each pair of adjacent nodes s, t of w with $s \neq t$ such that the subpath $s \to w \to t$ exists and is contained in some NOS routing path, a new commodity c is created between s and t. Its demand value is the part of the total NOS flow on $s \to w \to t$ which should be restored.

The set $\mathcal{IP}^w_{st}$ is the set of all paths containing $s \to w \to t$. Let $\Gamma(v)$ denote the set of adjacent nodes of a node $v \in V$ in G. Then this can be formulated as

$$\mathcal{IP}^w_{st} = \{P \in \mathcal{P}^0_{uv} \mid uv \in D,\ s \to w \to t \subseteq P\},$$
$$C^w_{LR} = \Big\{(s,t,d) \;\Big|\; s,t \in \Gamma(w),\ s \neq t,\ d = \sum_{uv \in D} \sum_{\substack{P \in \mathcal{P}^0_{uv} \\ s \to w \to t \subseteq P}} \rho^w_{uv} f_{uv}(P)\Big\}.$$

Using *Path Restoration*, the approach is nearly the same as for link failures: a new commodity is created between the end nodes of each affected NOS demand unless the failing node is source or target of the NOS demand; in this case, the demand is lost. The demand value of the new commodity is the part of the affected NOS flow value for this demand which should be restored.

The set $\mathcal{IP}^w_{uv}$ is the set of all NOS routing paths between u and v containing the failing node as an *inner* node. Observe that under the assumption of loopless paths, the paths containing w as an inner node are exactly the paths satisfying $|P \cap \delta(w)| = 2$. Consequently,

$$\mathcal{IP}^w_{uv} = \{P \in \mathcal{P}^0_{uv} \mid u,v \neq w,\ w \in P\},$$
$$C^w_{PR} = \Big\{(u,v,d) \;\Big|\; uv \in D,\ d = \sum_{\substack{P \in \mathcal{P}^0_{uv} \\ |P \cap \delta(w)| = 2}} \rho^w_{uv} f_{uv}(P)\Big\}.$$

Note that both $|C^w_{LR}|$ and $|C^w_{PR}|$ are bounded by the number of node pairs in the graph, i.e., by $|V|(|V|-1)/2$.

For *Reservation*, the definitions are practically the same as stated above for single link failures, as everything is rerouted, independently of the specific type of failure.

3.3.5 Inequalities

We are now in a state to be be able to write the whole restoration model down in terms of three types of constraints:

1. *Demand constraints*, stating that the demands must be satisfied by the NOS flow:

$$\sum_{P \in \mathcal{P}^0_{uv}} f_{uv}(P) = d_{uv} \qquad \forall\, uv \in D. \tag{3.8}$$

2. *Flow saving constraints*: In case of a failure, the interrupted NOS flow on paths that cause the failure commodity $c \in C^s$ to exist must be restored by the additional flow created for c:

$$\sum_{uv \in D} \sum_{P \in \mathcal{P}^0_{uv} \cap \mathcal{I}\mathcal{P}^s_c} \rho^s_{uv} f_{uv}(P) \leq \sum_{P \in \mathcal{P}^s_c} f^s_c(P) \qquad \forall\, s \in S \backslash \{0\},\ \forall\, c \in C^s. \tag{3.9}$$

3. *Capacity constraints:* In any operating state, the NOS flow on an edge $e \in E^s$ plus the additional flow on e induced by the failure must not exceed the capacity of e:

$$\sum_{uv \in D} \sum_{\substack{P \in \mathcal{P}^0_{uv} \\ e \in P}} \gamma^s(P) f_{uv}(P) + \sum_{c \in C^s} \sum_{\substack{P \in \mathcal{P}^s_c \\ e \in P}} f^s_c(P) \leq y_e \qquad \forall\, s \in S,\ \forall\, e \in E^s. \tag{3.10}$$

The value $\gamma^s(P)$ depends on the used restoration model and is calculated as follows: Let $\delta^s(P) = 1$ if P is valid in operating state s (i.e., $P \in \mathcal{P}^s_{uv}$) and 0 otherwise. Then

$$\gamma^s(P) := \begin{cases} 1 & \text{for Local Restoration,} \\ 1 & \text{for Global Restoration without stub release,} \\ \delta^s(P) & \text{for Global Restoration with stub release.} \end{cases}$$

To motivate this definition, consider the following restoration mechanisms and operating states:

- In NOS, whatever restoration technique may be used, there is no failure flow and $\delta^0(P) = 1$ for all paths P since no path fails in NOS. Hence, $\gamma^0(P) = 1$ for all paths P and the inequality yields the usual capacity restrictions for the NOS flow.
- With Link Restoration (or any other Local Restoration scheme) or with a global restoration mechanism (in particular, Path Restoration or Reservation) without stub release, capacity has to be reserved for all NOS paths (regardless of whether they are failing or not) and all failure paths. This is done by setting $\gamma^s(P) = 1$ for every NOS path.
- With stub release, the NOS flow used by failing paths is released prior to rerouting. Otherwise stated, capacity on an edge e has to be reserved for all failure paths containing e and for those NOS paths containing e which are usable in operating state s. This is exactly what is done by setting $\gamma^s(P) = \delta^s(P)$.

3.3.6 Some notes on the model

Areas of application

We are now going to present a short summary of which aspects of network restoration are covered by the model and which are not.

- Although the examples often refer to single failures for the sake of simplicity, the model uses an *arbitrary* set of operating states and an *arbitrary* restoration mechanism; the only differences between the restoration techniques are the actual definition of the failure commodities and the coefficient of the NOS flow variables in the capacity constraints.

  It turned out that the implementation of the model was in major parts independent of the used set of operating states or the restoration model as well.

  Note, however, that with Reservation (with stub release, i.e., the existing routing is completely discarded and replanned), the routings in the different operating states are completely unrelated with each other. Thus, from a computational point of view, it is advisable to consider each operating state individually and independently of the other operating states. By doing so, the problem reduces to solving a sequence of small LPs, which probably leads to much smaller computation times than applying the presented model and solving one big LP. As a result, the applicability of the model to Reservation is more of theoretical than of practical interest.

- The model applies to both undirected and directed networks. Upon implementation of the model, only the set of interrupted paths and the set of failure commodities have to be adapted to the graph type; these changes are straightforward.

- Suppose that an NOS routing is already given and the problem consists only of determinating the failure routings and corresponding capacities. In this case, one can simply substitute the variables $f_{uv}(P)$ by their value in the given NOS routing and hide them in the right hand side of the constraints.

  In this case, the demand constraints (3.8) become completely obsolete as they only influence the NOS routing. The former flow saving constraints (3.9) transform into some sort of demand constraints for each failure state in this case, stating that the flow used for one commodity must be at least of a certain constant demand value. The capacity constraints (3.10) state that the failure flow on a given edge must not exceed a certain capacity. Moreover, as we know in advance from the NOS routing which failure commodities have to be generated, the size of the resulting LP reduces dramatically.

  Note as well that if an NOS routing is given, the restoration LP nicely decomposes into the different failure states, and each operating state can be handled separately. This dramatically reduces the size of the restoration LP (and thus the required time for the feasibility test, as described in the next chapter).

- The model assumes a bifurcated routing. In other words, it is *not* possible to route each demand on one path only.

- Integrality of the flow can be formulated in the restoration model. However, our solution method (see Chapter 4) would be inappropriate. In fact, the restoration LP is solved very often as a capacity feasibility test, which is impractical with a restoration MIP.

- Although the model contains no diversification, it should not be hard to be extended appropriately. Also our employed algorithmic procedure can be used in this case (see [Wess00] for details on how to adapt the solution process to diversification).

Paths with or without loops?

We excluded paths with loops in the model, which at first glance seems to raise problems with Link Restoration: as seen in Section 2.1.2, the backhauling effect may cause Link Restoration paths to contain loops.

However, this is no contradiction to the model: the failure routing paths used in the model are paths connecting the end nodes of a failing edge, for instance, and we may safely disallow loops for them. The loops can only occur upon concatenation of such a failure path with the parts of the corresponding NOS paths which did not fail. But the paths resulting from this concatenation are not part of the model. The model only contains loopless NOS routing paths and loopless failure routing paths, whereas the paths resulting from concatenation have to be dealt with in a postprocessing of the solution.

Aggregation of commodities

In our model, all NOS and failure commodities consist of exactly one demand. Some authors use an aggregated form of commodities: a vertex cover in the demand graph is calculated, and all demands covered by the same node are are aggregated into one commodity. This approach allows the formulation of the demand constraints, in terms of commodities instead of demands. Since the number of commodities built in this way is $O(|V|)$ as opposed to $O(|V|^2)$ demands, this often leads to a significantly reduced size of the resulting linear programs if there are many NOS demands.

In our model, no such aggregation was possible since in the aggregated version, no proper control of the used NOS paths for a given demand is possible. However, this is necessary to formulate hop limits for NOS paths on a per-demand basis and for the flow saving constraints (3.9).

The size of the resulting LP is in fact the major drawback of our model. In practice, demand graphs are often quite dense, so the number of failure commodities and thus of flow saving constraints (3.9) can be quite high especially with Path Restoration (note that there is one such constraint for each failure commodity in each operating state; thus, with Path Restoration, there are $|D|(|S|-1)$ such constraints). With Link Restoration, the number of failure commodities is usually tolerable. Single link failures pose no problem anyway (they lead to one failure commodity per failure state), and the number of failure commodities upon failure of a node with degree d is bounded by $d(d-1)/2$. Given that telecommunication graphs often have an average nodal degree between 2 and 6, this number is manageable.

In our implementation, the number of constraints that are actually contained in the LP could drastically be reduced by a good choice of the initial path variables and delayed construction of constraints (see Section 5.1.1 for details). As a result, the calculation times could be reduced to reasonable values in many cases.

3.4 The whole model at a glance

Figure 3.2 shows the whole model at a glance, with all hardware and restoration requirements and variable domains. Note that the two parts of the model, namely the hardware MIP and the restoration LP, are linked with each other only by the edge capacities y_e. Thus, the routing conditions affect primarily the edge capacities, which in turn affect the installed hardware at the nodes.

$$\min \quad \sum_{v\in V}\Big(\sum_{d\in\mathcal{D}(v)} c_v^d x_v^d + \sum_{m\in\mathcal{M}(v)} c^m x_v^m\Big) + \sum_{e\in E}\sum_{d\in\mathcal{D}(e)} c_e^d x_e^d$$

(RLP)

$$\begin{aligned}
\sum_{P\in\mathcal{P}_{uv}^0} f_{uv}(P) &= d_{uv} && \forall\, uv\in D\\
\sum_{uv\in D}\sum_{P\in\mathcal{P}_{uv}^0\cap\mathcal{IP}_c^s} \rho_{uv}^s f_{uv}(P) - \sum_{P\in\mathcal{P}_c^s} f_c^s(P) &\le 0 && \forall\, s\in S\backslash\{0\},\ \forall\, c\in C^s\\
\sum_{uv\in D}\sum_{\substack{P\in\mathcal{P}_{uv}^0\\ e\in P}} \gamma^s(P) f_{uv}(P) + \sum_{c\in C^s}\sum_{\substack{P\in\mathcal{P}_c^s\\ e\in P}} f_c^s(P) &\le y_e && \forall\, s\in S,\ \forall\, e\in E^s
\end{aligned}$$

(HLP)

$$\begin{aligned}
\sum_{d\in\mathcal{D}(e)} x_e^d &\le 1 && \forall\, e\in E\\
\sum_{d\in\mathcal{D}(v)} x_v^d &\le 1 && \forall\, v\in V\\
\sum_{d\in\mathcal{D}(e)} C^d x_e^d &= y_e && \forall\, e\in E\\
\sum_{e\in\delta(v)} y_e - \sum_{d\in\mathcal{D}(v)} C^d x_v^d &\le 0 && \forall\, v\in V\\
\sum_{m\in\mathcal{M}(v)} S^m x_v^m - \sum_{d\in\mathcal{D}(v)} S^d x_v^d &\le 0 && \forall\, v\in V\\
\sum_{e\in\delta(v)}\sum_{d\in\mathcal{D}(e)} \mathcal{I}(d) x_e^d - \sum_{m\in\mathcal{M}(v)}\sum_{i\in\mathcal{I}(m)} I_m^i x_v^m &\le 0 && \forall\, v\in V,\ \forall\, i\in\mathcal{I}(v)\\
x_v^m - \sum_{d\in\mathcal{D}(v)} M_d^m x_v^d &\le 0 && \forall\, v\in V,\ \forall\, m\in\mathcal{M}(v)
\end{aligned}$$

$$\begin{aligned}
x_e^d &\in \{0,1\} && \forall\, e\in E,\ \forall\, d\in\mathcal{D}(e)\\
x_v^d &\in \{0,1\} && \forall\, v\in V,\ \forall\, d\in\mathcal{D}(v)\\
x_v^m &\in \mathbb{N} && \forall\, v\in V,\ \forall\, m\in\mathcal{M}(v)\\
y_e &\in \mathbb{N} && \forall\, e\in E\\
f_{uv}(P) &\ge 0 && \forall\, uv\in D,\ \forall\, P\in\mathcal{P}_{uv}^0\\
f_c^s(P) &\ge 0 && \forall\, s\in S\backslash\{0\},\ \forall\, c\in C^s,\ \forall\, P\in\mathcal{P}_c^s
\end{aligned}$$

Figure 3.2: The whole MIP including hardware and routing constraints

Chapter 4

Algorithmic approach

This chapter describes the algorithmic procedure used to solve the test problems in this thesis. In addition to an outline of the algorithm as a whole, this chapter focuses on the theoretical issues of the capacity feasibility problem, the separation of metric inequalities and the pricing problem for various restoration techniques (including its computational complexity). In contrast, the implementational issues of the algorithm will be discussed in Chapter 5. The reader is assumed to be familiar with the basics of Branch and Bound and cutting plane algorithms; an introduction to these concepts can be found in many books on Combinatorial Optimization.

4.1 Overview on the algorithm

In our approach to solve the given large scale MIP, a Branch and Cut framework serves as the base algorithm, i.e., a Branch and Bound algorithm, in which globally valid cutting planes are generated during the whole computation. In addition, at each node of the Branch and Bound tree, several sub-algorithms can be used independently of each other: separators, a capacity feasibility test and heuristics for finding feasible solutions. It is the feasibility test in which all routing and restoration information is coded, as will be described in detail in the following sections.

Our initial LP relaxation on which the Branch and Cut procedure is based consists only of the LP relaxation of (HLP), i.e., it contains only hardware requirements without integrality constraints. No conditions on the routing are included (in particular, no restoration requirements).

The optimal solution value of this LP relaxation defines a lower bound on the overall optimal solution value by definition of a relaxation.[1] Based on the LP relaxation of (HLP), a Branch and Bound tree is built, with a branching rule based on the available edge designs. Figure 4.1 illustrates the process at one node of the Branch and Bound tree.

At each node of the Branch and Bound tree, globally valid cutting planes are generated and added to the LP relaxation. When no more violated inequalities are found, the current (probably fractional) capacity vector $\bar{y}$ is tested for validity with respect to the missing restoration constraints. This is the point where the restoration LP (RLP) is actually considered.

If the capacities do not allow a routing satisfying the restoration constraints, a violated metric inequality (see Section 4.2.2) is separated in order to cut off the current infeasible

[1] In particular, if the LP relaxation is not solvable, the whole problem has no solution as well.

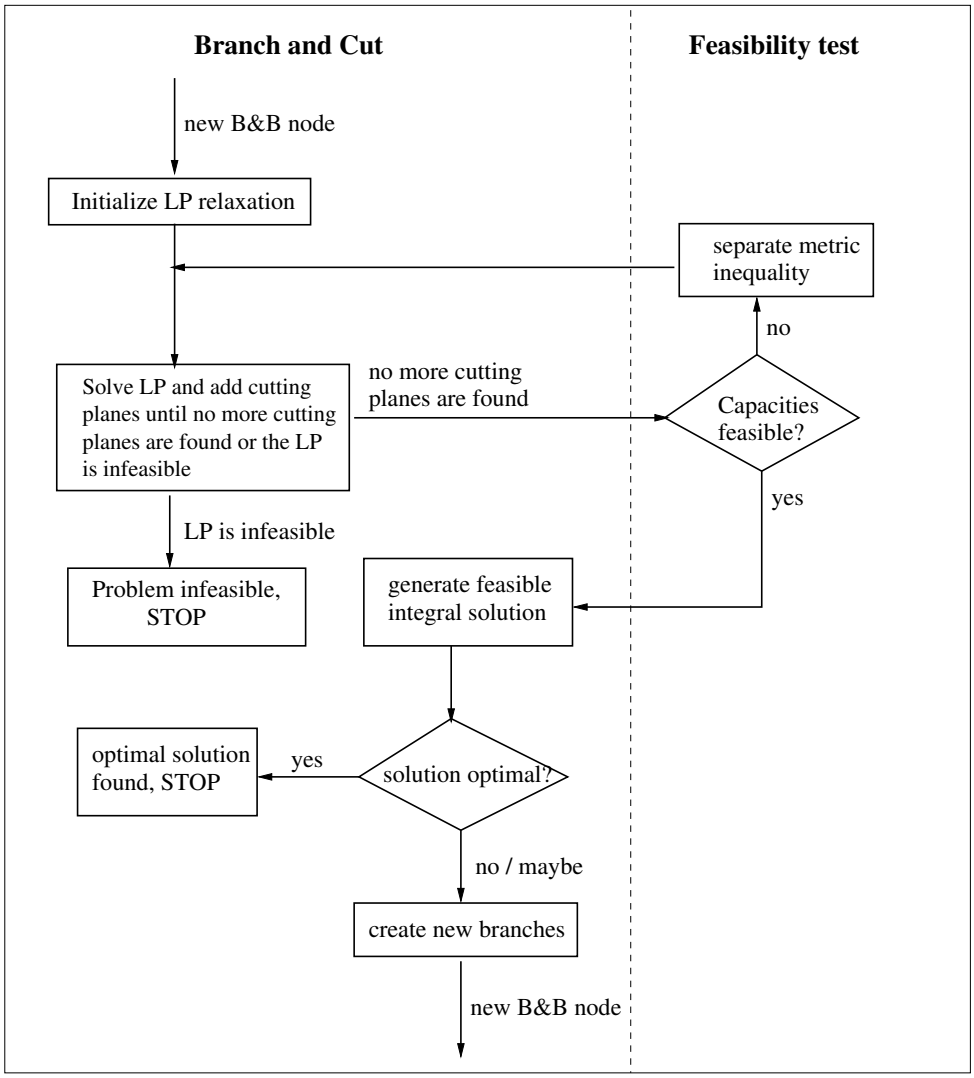

Figure 4.1: Overview on the process in one Branch and Bound node

capacity vector (thus raising capacity on some edges), and the process is iterated.

Otherwise, the capacities allow a feasible routing, and heuristics may be applied in order to obtain a feasible solution with integer capacities. Two cases have to be distinguished:

1. If the feasible capacity vector $\bar{y}$ is integral, it is feasible for the whole MIP and thus defines an upper bound on the overall optimal solution value. In addition, if the current node is the root node of the Branch and Bound tree, no variables are fixed yet. Since $\bar{y}$ is an optimal solution of a relaxation of the original MIP while being a feasible solution of the MIP itself, $\bar{y}$ is globally optimal in this case, and the calculation can be stopped. Otherwise, $\bar{y}$ just defines a feasible solution.

2. If the capacity vector $\bar{y}$ is feasible but not integral, a MIP heuristic can be employed to find a feasible integral (but usually not optimal) capacity vector, possibly yielding a better upper bound than previously known. The heuristic calculates the cheapest

hardware configuration over the current fractional capacity vector by solving a relatively small MIP (using CPLEX).

In any case, if the given capacity vector is feasible and not globally optimal, two child nodes are created for the current Branch and Bound node, each one corresponding to some range of allowed edge designs on a given edge.

The remaining of this chapter deals with the feasibility test, metric inequalities and pricing, as this is the part containing all restoration and routing information.

4.2 The capacity feasibility problem

The *capacity feasibility problem* consists of deciding whether a given (maybe fractional) capacity vector allows a routing which meets the restoration requirements, and if not, of finding a cutting plane that cuts off the infeasible point from the polyhedron induced by the feasible capacity vectors.

4.2.1 Testing the feasibility of a given capacity vector

Let $\bar{y} \in \mathbb{R}^E$ be a (probably fractional) capacity vector. The feasibility problem is to decide whether $\bar{y}$ allows routing variables $f_{uv}(P)$, $f_c^s(P) \geq 0$ such that

$$\begin{aligned}
\sum_{P \in \mathcal{P}_{uv}^0} f_{uv}(P) &= d_{uv} && \forall uv \in D \\
-\sum_{uv \in D} \sum_{P \in \mathcal{P}_{uv}^0 \cap \mathcal{IP}_c^s} \rho_{uv}^s f_{uv}(P) + \sum_{P \in \mathcal{P}_c^s} f_c^s(P) &\geq 0 && \forall s \in S \backslash \{0\},\ \forall c \in C^s \\
\sum_{uv \in D} \sum_{\substack{P \in \mathcal{P}_{uv}^0 \\ e \in P}} \gamma^s(P) f_{uv}(P) + \sum_{c \in C^s} \sum_{\substack{P \in \mathcal{P}_c^s \\ e \in P}} f_c^s(P) &\leq \bar{y}_e && \forall s \in S,\ \forall e \in E^s
\end{aligned}$$

is satisfied. In other words, we are interested in knowing whether $\bar{y}$ is contained in the polyhedron

$$Y := \{y \in \mathbb{R}_+^E \mid y \text{ satisfies (3.8), (3.9) and (3.10)}\}.$$

In order to arrive at this decision, the feasibility problem is transformed into an optimization problem by introducing a new variable $\alpha \in \mathbb{R}$ whose value is the maximum missing capacity on any edge for the preceding problem to be solvable (see also [Min81] for this approach). This leads to the linear program (P):

$$\begin{aligned}
&\min\ \alpha \\
\sum_{P \in \mathcal{P}_{uv}^0} f_{uv}(P) &= d_{uv} && \forall uv \in D \\
-\sum_{uv \in D} \sum_{P \in \mathcal{P}_{uv}^0 \cap \mathcal{IP}_c^s} \rho_{uv}^s f_{uv}(P) + \sum_{P \in \mathcal{P}_c^s} f_c^s(P) &\geq 0 && \forall s \in S \backslash \{0\},\ \forall c \in C^s \\
-\sum_{uv \in D} \sum_{\substack{P \in \mathcal{P}_{uv}^0 \\ e \in P}} \gamma^s(P) f_{uv}(P) - \sum_{c \in C^s} \sum_{\substack{P \in \mathcal{P}_c^s \\ e \in P}} f_c^s(P) + \alpha &\geq -\bar{y}_e && \forall s \in S,\ \forall e \in E^s \\
f_{uv}(P),\ f_c^s(P) \geq 0, &\quad \alpha \in \mathbb{R}
\end{aligned}$$

The capacity vector $\bar{y}$ allows a feasible routing if and only if the optimal solution value α^* of (P) satisfies $\alpha^* \leq 0$ (meaning that there is enough capacity on every edge to accommodate all flow on that edge).

As the number of paths may be exponential in the number of edges, this problem cannot be solved directly with a commercial LP solver like CPLEX. Instead, we have to generate a small initial set of paths that allows a (probably non-optimal) solution of (P) and apply a column generation procedure to solve (P) to optimality. This will be described in detail in Section 4.3. First, let π_{uv}, β_c^s and μ_e^s be the dual variables to the above constraints and consider the dual (D) of the above LP:

$$\begin{aligned}
\max \; & -\sum_{s\in S}\sum_{e\in E^s} \bar{y}_e \mu_e^s + \sum_{uv\in D} d_{uv}\pi_{uv} & & \\
& \sum_{s\in S}\sum_{e\in E^s} \mu_e^s & = \; 1 & \\
& -\sum_{e\in P} \mu_e^s + \beta_c^s & \leq \; 0 & \quad \forall s \in S\backslash\{0\},\ \forall c \in C^s,\ \forall P \in \mathcal{P}_c^s \\
& -\sum_{s\in S}\sum_{e\in E^s\cap P} \gamma^s(P)\mu_e^s - \sum_{s\in S\backslash\{0\}}\sum_{\substack{c\in C^s \\ P\in\mathcal{IP}_c^s}} \rho_{uv}^s \beta_c^s + \pi_{uv} & \leq \; 0 & \quad \forall uv \in D,\ \forall P \in \mathcal{P}_{uv}^0 \\
& \beta_c^s, \mu_e^s \geq 0, \quad \pi_{uv} \in \mathbb{R} & &
\end{aligned}$$

These constraints are the dual constraints to α, f_c^s and $f_{uv}(P)$, in this order. Note that although π_{uv} is unrestricted, it will always be nonnegative in an optimal solution. This is due to the fact that the demand constraints in (P) can as well be stated as

$$\sum_{P\in\mathcal{P}_{uv}^0} f_{uv}(P) \geq d_{uv} \qquad \forall\, uv \in D$$

without changing the solution space of the capacity variables.

From linear programming duality theory we know that (P) has an optimal solution if an only if (D) has one, and that in this case their optimal solution values are equal. In particular, $\bar{y}$ allows a feasible routing if and only if the common optimal solution value α^* of (P) and (D) is nonpositive, as explained above. Since the dual LP is a maximization problem, this is the case if and only if all feasible solutions for (D) have a nonpositive objective value. This immediately proves

Proposition 4.1 *A capacity vector $\bar{y}$ allows a feasible routing for the restoration problem defined by* (3.8), (3.9) *and* (3.10) *if and only if*

$$\sum_{s\in S}\sum_{e\in E^s} \bar{y}_e \mu_e^s \geq \sum_{uv\in D} d_{uv}\pi_{uv}$$

for all $\mu_e^s \geq 0$, $\beta_c^s \geq 0$ and $\pi_{uv} \geq 0$ satisfying

$$\begin{aligned}
\sum_{s\in S}\sum_{e\in E^s\cap P} \gamma^s(P)\mu_e^s + \sum_{s\in S\backslash\{0\}}\sum_{\substack{c\in C^s \\ P\in\mathcal{IP}_c^s}} \rho_{uv}^s\beta_c^s & \geq \pi_{uv} \quad \forall uv \in D,\ \forall P \in \mathcal{P}_{uv}^0 \\
\sum_{e\in P} \mu_e^s & \geq \beta_c^s \quad \forall s \in S\backslash\{0\},\ \forall c \in C^s,\ \forall P \in \mathcal{P}_c^s.
\end{aligned}$$

Note: The constraint $\sum_{s\in S}\sum_{e\in E^s}\mu_e^s = 1$ can be omitted without loss of generality: instead of minimizing α, we could as well minimize $C\alpha$ for any $C > 0$ since we are only interested in the sign of α. This means that the above constraint may be replaced by $\sum_{s\in S}\sum_{e\in E^s}\mu_e^s = C$ for some arbitrary $C > 0$. In other words, the argumentation above holds independently of the value of the sum of the μ_e^s, and the constraint may simply be omitted.

4.2.2 Metric inequalities

Proposition 4.1 is a generalization of the so-called *Japanese theorem* which has been found independently by Iri [Iri71] and by Onaga and Kakusho [OK71]:

Theorem 4.2 ***(Japanese theorem)****: Let π_{uv}^{μ} denote the shortest path distance between u and v with respect to the edge weights $\mu \in \mathbb{R}_+^E$. Then a given capacity vector $y \in \mathbb{R}_+^E$ allows a feasible NOS routing for demand values d_{uv} if and only if*

$$\sum_{e\in E} y_e\mu_e \geq \sum_{uv\in D} \pi_{uv}^{\mu} d_{uv} \tag{4.1}$$

for all edge length vectors $\mu \in \mathbb{R}_+^E$.

This theorem can be obtained from proposition 4.1 by considering only the NOS. Inequality (4.1) is called a *metric inequality*. The name reflects the fact that for given $\mu \in \mathbb{R}_+^E$, the shortest path lengths π_{uv}^{μ} define a pseudo-metric on the nodes of G, i.e., the distance defined by $d(u, v) := \pi_{uv}^{\mu}$ is nonnegative, symmetric (provided that the graph is undirected) and satisfies the triangle inequality. However, as this distance function does not satisfy $d(u, v) > 0$ for all $u \neq v$, it does not define a metric but only a pseudo-metric.

Observe that

$$\sum_{s\in S}\sum_{e\in E^s} y_e\mu_e^s = \sum_{e\in E}\Big(\sum_{\substack{s\in S:\\ e\in E^s}} \mu_e^s\Big) y_e$$

since the term $y_e\mu_e^s$ appears in the sum if and only if $e \in E^s$. Thus, Proposition 4.1 states that the inequalities of the form

$$\sum_{e\in E}\Big(\sum_{\substack{s\in S:\\ e\in E^s}} \mu_e^s\Big) y_e \geq \sum_{uv\in D} d_{uv}\pi_{uv} \tag{4.2}$$

for all μ, β and π satisfying the side constraints stated in the proposition, completely describe the capacity polyhedron Y. In particular, these inequalities are valid for Y. Observe that there are infinitely many of these inequalities, so this description of Y is far from being minimal. Fortunately, we know from polyhedra theory that any finite dimensional polyhedron can be described by finitely many inequalities, so we actually do not need all metric inequalities.

Due to the obvious analogy to (4.1), we will refer to the inequalities (4.2) as *metric inequalities* as well.

4.2.3 Separating metric inequalities

Recall from Section 4.2.1 that the given capacity vector $\bar{y}$ is contained in the capacity polyhedron Y if and only if the common optimal solution value α^* of (P) and (D) is nonpositive.

Otherwise stated, $\bar{y}$ is *infeasible* if and only if the objective function of (D) satisfies

$$\sum_{s \in S} \sum_{e \in E^s} \bar{y}_e \mu_e^{s*} < \sum_{uv \in D} d_{uv} \pi_{uv}^*.$$

This defines a violated metric inequality which can be added to the LP Relaxation to cut off the infeasible capacity vector. Thus, the Path LP acts not only as a feasibility tester for capacities but also as a separator for metric inequalities.

4.3 Solving the Path LP with column generation

As mentioned above, we have to start solving (P) with the variable α and a restricted number of path variables that allow a solution to (P), generating further variables on the fly as needed. As the choice of the initial path variables is an interesting implementation issue but not so much of theoretical interest, it will be covered in detail in Section 5.1.1.

In this section, the theoretical aspects of the *pricing problem* will be discussed. In addition to investigating the actual representation of the pricing problem for the different restoration techniques, its computational complexity will be discussed. It will be shown that the pricing problem is $\mathcal{NP}$-hard for Path Restoration with stub release, but polynomially solvable for Path Restoration without stub release or Link Restoration.

4.3.1 The pricing problem

Given an optimal solution of a restricted version of (P), the *pricing problem* consists of deciding whether the solution is optimal for the whole LP with all path variables, and, if not, which path variables to add to the linear program before resolving it.

Suppose that the primal LP currently consists of α and some path variables from a restricted set $\mathcal{P}_r$ of paths, with corresponding flow variable vector $f = (f_P)_{P \in \mathcal{P}_r}$. Furthermore, suppose that $\bar{x} := (\bar{f}, \bar{\alpha})$ is an optimal solution of (P) with respect to $\mathcal{P}_r$. Let $\bar{z} := (\bar{\mu}, \bar{\beta}, \bar{\pi})$ be the corresponding dual solution that is feasible and optimal w.r.t. the restricted set of dual constraints corresponding to the paths in $\mathcal{P}_r$. In particular, $\bar{x}$ and $\bar{z}$ have the same objective value by construction.

Observe that the restricted primal LP already contains all necessary constraints. Since we may always add all missing path flow variables with flow value 0 without violating any constraint, $\bar{x}$ is also feasible for the whole primal LP.

Similarly, the restricted LP already contains all variables but not all constraints. This implies that $\bar{z}$, being optimal for the restricted dual LP, is optimal for the whole dual LP if and only if it satisfies all missing constraints.

All dual constraints corresponding to variables in the current primal LP (i.e., α and all paths in $\mathcal{P}_r$) are certainly satisfied by $\bar{z}$ since this point was defined to be a feasible (and optimal) dual solution with respect to the restricted number of constraints. But does $\bar{z}$ also satisfy the missing dual constraints?

Now, optimality of $\bar{x}$ can be tested by looking whether $\bar{z}$ violates any dual constraint. If not, this means that $\bar{z}$ is feasible for (D). This in turn implies optimality of $\bar{x}$ for (P) since $\bar{x}$ is feasible for (P) and $\bar{x}$ and $\bar{z}$ share the same objective value. If, on the other hand, a violated dual constraint is found, the corresponding path variable is added to the primal LP which is resolved afterwards.

Note that $\bar{z}$ may be infeasible for (D) although $\bar{x}$ is optimal for (P). Hence, in order to be sure that $\bar{x}$ is optimal, we have to continue adding path variables to (P) until we know that the current dual vector is feasible for (D).

How to test whether $\bar{z}$ violates any dual constraint will be subject of the following sections.

4.3.2 Pricing parts common to all restoration models

1. First of all, the equality $\sum_{s\in S}\sum_{e\in E^s}\mu_e^s = 1$ will always be satisfied for an optimal solution of (D) since the corresponding primal variable α is contained in the LP from the beginning.

2. The inequality

$$\sum_{e\in P}\mu_e^s \geq \beta_c^s \qquad \forall s \in S\backslash\{0\},\ \forall c \in C^s,\ \forall P \in \mathcal{P}_c^s \tag{4.3}$$

   nicely decomposes into one shortest path problem for every $s \in S\backslash\{0\}$ and every $c \in C^s$. These commodities are point-to-point demands, so if $st = c \in C^s$, we have to find a shortest st-path P with respect to edge weights μ_e^s; if its length is less than β_c^s, the path P violates a dual constraint and the variable $f_c^s(P)$ can be added to the primal LP. As the μ_e^s are nonnegative and as failure paths have no length restrictions, the Dijkstra shortest path algorithm can be used.

3. To test whether the inequality

$$\sum_{s\in S}\sum_{e\in E^s\cap P}\gamma^s(P)\mu_e^s + \sum_{s\in S\backslash\{0\}}\sum_{\substack{c\in C^s\\ P\in\mathcal{IP}_c^s}}\rho_{uv}^s\beta_c^s \geq \pi_{uv} \tag{4.4}$$

   is violated by some NOS path $P \in \mathcal{P}_{uv}^0$, we have to find a path that minimizes the left-hand side and look whether its evaluation w.r.t. the left-hand side exceeds π_{uv} or not.

   The term $\gamma^s(P)\mu_e^s$ appears in the first sum if and only if $e \in E^s \cap P$, i.e., the first sum can be written as

$$\sum_{s\in S}\sum_{e\in E^s\cap P}\gamma^s(P)\mu_e^s = \sum_{e\in P}\sum_{\substack{s\in S:\\ e\in E^s}}\gamma^s(P)\mu_e^s. \tag{4.5}$$

   When using Path Restoration without stub release or Link Restoration, the value $\gamma^s(P)$ equals 1 for every path P and every operating state s. In this case, the sum does not depend on the path P and integrates well into a shortest path problem. On the contrary, this sum poses problems with stub release and with Link Restoration, as will be explained later.

4.3.3 Pricing parts changing according to the restoration model

Up to this point, most in this chapter was independent of the restoration model and of the considered set of operating states. Now we are going to explore the differences in pricing between the various restoration techniques. Instead of studying every possible variant here (which probably would not yield great additional insight into the structure of the problem), we will focus on single component failures in the remaining of this section. These are the

most widely used restoration settings and those that we actually implemented. For ease of notation, we will sometimes simply denote by g the failure state in which only the given edge $g \in E$ fails; the same applies to single node failures.

Global Restoration

Path Restoration: With Path Restoration, a failing NOS path P causes an edge failure commodity c to exist if and only if P contains the failing edge. Similarly, P causes a node failure commodity to exist if and only if P contains the failing node as an inner node. By definition of $\gamma^s(P)$, for a given operating state s, we have $\gamma^s(P) = 1$ if and only if $P \in \mathcal{P}^0_{uv}$ (without stub release) or $P \in \mathcal{P}^s_{uv}$ (with stub release).

Under the assumption of loopless paths, a path P contains an inner node w if and only if P contains exactly two of the incident edges of w. In other words, we may assign half the weight of a node to each of its incident edges for solving a shortest path problem.

This observation yields that without stub release, Inequality (4.4) can be written as

$$\sum_{e \in P} \Big(\sum_{\substack{s \in S: \\ e \in E^s}} \mu^s_e + \delta(e,S)\rho^e_{uv}\beta^e_{uv} + \frac{1}{2} \sum_{\substack{w \in S \cap (V \setminus \{u,v\}): \\ e \in \delta(w)}} \rho^w_{uv}\beta^w_{uv} \Big) \geq \pi_{uv} \tag{4.6}$$

for all $uv \in D$ and for all $P \in \mathcal{P}^0_{uv}$, where

$$\delta(e,S) = \begin{cases} 1 & \text{if } e \in S \ \ (e \text{ is allowed to fail}), \\ 0 & \text{if } e \notin S \ \ (e \text{ is not allowed to fail}). \end{cases}$$

Observe that the last sum contains only two terms, one for each end node of e. The violation of the above inequality can be tested by calculating a length restricted shortest path for every NOS demand.

Note that for arbitrary edge costs $c_e \geq 0$, a cost bound $C \geq 0$, edge weights μ_e, a weight bound M and source and target nodes u and v, the problem of deciding whether a uv-path with weight at most M and cost at most C exists is $\mathcal{NP}$-complete. This can easily be seen by reduction to the 0-1-knapsack problem (or from [GJ79]).

On the contrary, if all costs are equal (as in our case, where $c_e = 1$ for all $e \in E$), the problem of finding a shortest path w.r.t. to μ among all uv-paths with cost at most C is polynomially solvable with a modified Dijkstra algorithm as described in [Wess00] (recall the nonnegativity of μ^s_e and β^s_c).

The number of shortest path problems per pricing step is bounded by the number of NOS demands, which in turn is bounded by the number of pairs of nodes in the network since we excluded parallel edges in the demand graph. Thus, the number of shortest path problems per pricing step cannot exceed $|V|^2/2$.

With stub release, the inequality reads

$$\sum_{e \in P} \sum_{\substack{s \in S \\ P \in \mathcal{P}^s_{uv}}} \mu^s_e + \sum_{s \in S \setminus \{0\}} \sum_{\substack{c \in C^s \\ P \in \mathcal{IP}^s_c}} \rho^s_{uv}\beta^s_c \ \geq \ \pi_{uv}. \tag{4.7}$$

Note that only the first sum has changed. Unfortunately, the first part of the edge weights now depends on the path which we want to find, so we have to resort to heuristics in this case. This applies to node failures as well as to edge failures. The heuristics which we used in our implementation will be discussed in Section 5.2.

Reservation: Recall from Section 3.3.4 that with Reservation, every NOS path P that can fail causes every failure commodity to exist, since upon failure of P, everything is rerouted. The failure commodities correspond to the original NOS demands, except for demands starting or ending in failing nodes. Thus, Inequality (4.4) can be written as

$$\sum_{e\in E}\sum_{\substack{s\in S:\\ e\in E^s}} \gamma^s(P)\mu_e^s + \sum_{\substack{s\in S\setminus\{0\}:\\ P\in\mathcal{P}_{uv}^s}} \sum_{\substack{xy\in D\\ x,y\in V^s}} \rho_{uv}^s\beta_{xy}^s \geq \pi_{uv}$$

for every uv-path P. To simplify notation, let from now on

$$\omega_{uv}^s := \sum_{\substack{xy\in D\\ x,y\in V^s}} \rho_{uv}^s\beta_{xy}^s.$$

As for Path Restoration, we may assign half the weight of a node to each of its incident edges. Without stub release, this leads to

$$\sum_{e\in P}\Big(\sum_{\substack{s\in S:\\ e\in E^s}} \mu_e^s + \delta(e,S)\omega_{uv}^e + \frac{1}{2}\sum_{\substack{w\in S\cap(V\setminus\{u,v\}):\\ e\in\delta(w)}} \omega_{uv}^e\Big) \geq \pi_{uv}. \tag{4.8}$$

Again, the violation of such an inequality can be decided by at most $|D| < |V|^2/2$ shortest path problems.

Note, however, that Reservation without stub release does not make much sense and is usually not an option in practice. On the contrary, Path Restoration without stub release is often applied to real world networks.

With stub release, again only the first part of the constraint changes, and Inequality (4.4) reads

$$\sum_{e\in P}\Big(\sum_{\substack{s\in S:\\ P\in\mathcal{P}_{uv}^s}} \mu_e^s + \delta(e,S)\omega_{uv}^e + \frac{1}{2}\sum_{\substack{w\in S\cap(V\setminus\{u,v\}):\\ e\in\delta(w)}} \omega_{uv}^e\Big) \geq \pi_{uv} \tag{4.9}$$

for all $uv \in D$ and for all $P \in \mathcal{P}_{uv}^0$.

But with Reservation, we have $\mathcal{P}_{uv}^s = \emptyset$ in any failure state as no NOS path is directly maintained (if it is, it is considered to be a failure path and not a NOS path). In the NOS, $\mathcal{P}_{uv}^s = \mathcal{P}_{uv}$ since no paths fails. Hence, the above sum changes to

$$\sum_{e\in P}\Big(\mu_e^0 + \delta(e,S)\omega_{uv}^e + \frac{1}{2}\sum_{\substack{w\in S\cap(V\setminus\{u,v\}):\\ e\in\delta(w)}} \omega_{uv}^e\Big) \geq \pi_{uv}. \tag{4.10}$$

The left-hand side of this inequality can directly be minimized using the Dijkstra shortest path algorithm.

Local Restoration

Let $\Gamma(w)$ be the set of adjacent nodes to $w \in V$. By using (4.5) and splitting the second sum into single node and single edge failures, Inequality (4.4) reads

$$\sum_{e\in P}\sum_{\substack{s\in S:\\ e\in E^s}} \mu_e^s + \sum_{\substack{g\in E\cap S:\\ g\in P}} \rho_{uv}^g\beta^g + \sum_{\substack{w\in V\cap S\\ w\neq u,v}} \sum_{\substack{s,t\in\Gamma(w),s\neq t\\ s\to w\to t\subseteq P}} \rho_{uv}^w\beta_{st}^w \geq \pi_{uv}$$

for all $uv \in D$ and for all $P \in \mathcal{P}^0_{uv}$.

To see why this holds, recall that for single link failures $g \in E \cap S$, only one commodity is created (thus only indexed by the operating state); a path P causes this commodity to exist if and only if $g \in P$. Similarly, a path P causes a failure commodity st in failure state $w \in V \cap S$ to exist if and only if $s \neq t$ and $s \to w \to t \subseteq P$.

By defining $\delta(e, S)$ as above, this inequality can be rewritten as

$$\sum_{e \in P} \Big(\sum_{\substack{s \in S: \\ e \in E^s}} \mu^s_e + \delta(e,S)\rho^e_{uv}\beta^e \Big) + \sum_{\substack{w \in V \cap S \\ w \neq u,v}} \sum_{\substack{s,t \in \Gamma(w), s \neq t \\ s \to w \to t \subseteq P}} \rho^w_{uv}\beta^w_{st} \geq \pi_{uv} \tag{4.11}$$

for all $uv \in D$ and for all $P \in \mathcal{P}^0_{uv}$.

Minimizing the first sum is simply a shortest path problem over all paths $P \in \mathcal{P}^0_{uv}$, for every demand $uv \in D$. Unfortunately, the last sum does not seem to integrate well into the shortest path problems at first glance. This problem is investigated in further detail in the following section.

4.3.4 Complexity results on the pricing problem

Having discussed the structure of the pricing problem for the different restoration mechanisms, we are now going to state some results on its computational complexity. It will be shown that the pricing problem is $\mathcal{NP}$-hard for Path Restoration with stub release but polynomially solvable without stub release or when using Link Restoration or Reservation.

Link Restoration

With Link Restoration, the pricing problem is polynomially solvable by transforming the underlying graph and solving a shortest path problem with the Dijkstra algorithm, as will be shown in this section.

The pricing problem for Link Restoration can be stated in a reduced form as follows:

> **Given** an graph $G = (V, E)$, $u, v \in V$, edge weights $c_e \geq 0$ for all $e \in E$, node weights $c^w_{st} \geq 0$ for all $w \in V$ and $s, t \in \Gamma(w)$ $(s \neq t)$,
> **find** a uv-path P minimizing
>
> $$c(P) := \sum_{e \in P} c_e + \sum_{\substack{w \in V \cap S \\ w \neq u,v}} \sum_{\substack{s,t \in \Gamma(w), s \neq t \\ s \to w \to t \subseteq P}} c^w_{st}.$$

In this context, $c^w_{st} = \rho^w_{uv}\beta^w_{uv}$, and c_e represents the weight that can directly be assigned to edge e in (4.11). The problem is that the weight of a node w depends on the path, or, more specifically, on the pair of adjacent nodes of w by which the path passes by w (if it does at all).

We will discuss the necessary transformation for undirected graphs only here; the adaption to directed graphs is straightforward.

First, note that if a node w has only two adjacent nodes s and t, a path passing by w has only one choice, namely passing by both s and t. In this case, simply add $c^w_{st}/2$ to each of the edges sw and wt. Nodes which are connected with the rest of the network by only one edge are not relevant at this point since they cannot be an inner node of a path without loops.

For each node $w \in V \cap S \backslash \{u, v\}$ with at least 3 adjacent nodes and each pair of adjacent nodes s, t of w, we do the following transformation (illustrated by Figure 4.2):

1. Create artificial nodes v_e on the edge $e = sw$ and v_f on the edge $f = wt$.
2. Replace the edges e and f by edges sv_e, v_ev_f and v_ft and delete node w.
3. Assign the weight $c_e/2$ to edge sv_e, the weight $c_f/2$ to edge v_ft and weight the $c_e/2 + c_f/2 + c^w_{st}$ to edge to edge v_ev_f.

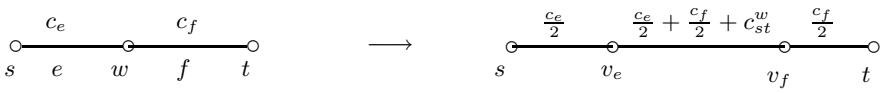

Figure 4.2: Transformation for one pair of adjacent nodes

A path passing by $s \to w \to t$ now gets the correct weight $c_e + c_f + c^w_{st}$.

Figure 4.3 illustrates the resulting graph in the neighbourhood of w after applying this procedure to all pairs of adjacent edges of w.

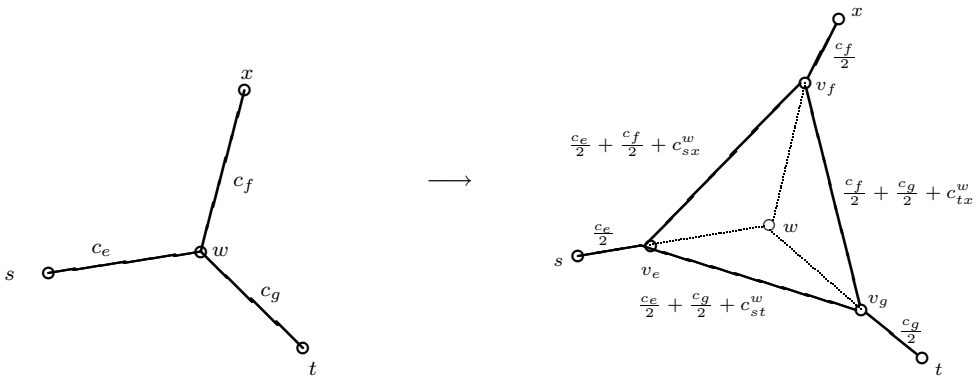

Figure 4.3: Transformation for all pairs of adjacent nodes

There is an immediate one-to-one correspondence between paths in the original graph and paths in the modified graph, and the weight is correct. For instance, suppose a path enters w by s in Figure 4.3. The value $c_e/2$ accounts for the fact that the path enters w by s. The other half of this weight is added on the edge from v_e to v_f or v_g, whichever of these edges is employed by the path; this edge also adds the correct weight c^w_{st} or c^w_{sx}, respectively. At the other end node (t or x), the principle is the same.

This leads us to the following proposition:

Proposition 4.3 *Using Link Restoration for the restoration of single component failures, it is possible in polynomial time in the number of nodes and edges to find a path violating a dual constraint or to determine that there is no such path.*

Proof. The graph transformation described above is polynomial in the number of nodes and edges since at most $|E|$ new nodes and at most $\sum_{w \in V} \deg(w)^2 \leq |V| \cdot |E|^2$ new edges are created and all occurring weights are bounded by three times the maximum of the original weights. After the transformation, the Dijkstra shortest path algorithm can be applied since all weights are nonnegative.

□

However, we did not employ this method in our implementation since it requires heavy graph transformation in each step, as seen above. As these would have taken too much time, we applied heuristics instead in order to handle node failures with Link Restoration (edge failures pose no problem anyway).

Path Restoration without stub release or Reservation

Summarizing the previous investigations, we obtain the following result:

Proposition 4.4 *Using Path Restoration without stub release or Reservation for the restoration of single component failures, it is possible in polynomial time in the number of nodes and edges to find a path violating a dual constraint or to determine that there is no such path.*

Proof. Recall that with Global Restoration, the failure commodities correspond to the NOS demands, so the number of failure commodities per operating state is bounded by $|D|$. When considering single component failures only, the number of failure states is $|S| - 1$. Thus, the violation of Inequality (4.3) can be tested by solving at most

$$|\bigcup_{s \in S \setminus \{0\}} C^s| \leq |D|(|S| - 1) < \frac{1}{2}(|V| + |E|)$$

shortest path problems, as seen in Section 4.3.2.

From the above considerations for Path Restoration without stub release and for Reservation, it follows that also the number of shortest path problems needed to test the violation of Inequality (4.4) is bounded by $|D| < |V|^2/2$ as at most one shortest path problem per NOS demand has to be solved. Thus, the violation of all dual constraints can be tested in polynomial time, which was to be shown.

□

Path Restoration with stub release

In contrast to this, the pricing problem is $\mathcal{NP}$-hard for Path Restoration with stub release, as will be shown now. Recall that in this case, we have to find a path that minimizes

$$\sum_{e \in P} \Big(\sum_{\substack{s \in S: \\ P \in \mathcal{P}^s_{uv}}} \mu^s_e + d_e \Big)$$

among all uv-paths, with suitable weights d_e representing the remaining parts of the edge weights in (4.7) and (4.9).

For ease of notation, we will write $v \in P$ for nodes in much the same way as we write $e \in P$ for edges, with the obvious meaning. Thus, when considering only single component failures (where $s = g \in E$ or $s = w \in V$), the above sum can be written as

$$\sum_{e \in P} \left(\sum_{s \notin P} \mu^s_e + d_e \right).$$

We will show the $\mathcal{NP}$-completeness of the corresponding decision problems for single link and single node failures separately; together, this implies the $\mathcal{NP}$-completeness for all single component failures.

For link failures, define the decision problem as follows:

PRICE_LINKFAILURES:
Given: a graph $G = (V, E)$, weights $\mu_e^g \geq 0$ for all $e, g \in E$, two nodes $u, v \in V$, a bound $K \geq 0$
Question: Is there a uv-path P with length

$$c(P) := \sum_{e \in P} \sum_{g \in E \setminus P} \mu_e^g \leq K \ ?$$

The corresponding decision problem for node failures is the following:

PRICE_NODEFAILURES:
Given: a graph $G = (V, E)$, weights $\mu_e^w \geq 0$ for all $e \in E$ and $w \in V$, two nodes $u, v \in V$, a bound $K \geq 0$
Question: Is there a uv-path P with length

$$c(P) := \sum_{e \in P} \sum_{w \in V \setminus P} \mu_e^w \leq K \ ?$$

We will reduce these problems to the problem of finding a maximal cut in a network, which is known to be $\mathcal{NP}$-complete ([GJ79]):

MAXCUT:
Given: the directed complete graph $K_n = (V_n, E_n)$,
symmetric weights $w_{ij} = w_{ji} \geq 0$ for all $i, j \in V$, a bound $K \geq 0$
Question: Is there a subset $A \subseteq V$ such that the weight of the cut $\delta(A)$ satisfies

$$c(A) := 2 \sum_{i \in A} \sum_{j \in V \setminus A} w_{ij} \geq K \ ?$$

The following proof for link failures is based on [Bley03].

Proposition 4.5 *The problem PRICE_LINKFAILURES is $\mathcal{NP}$-complete.*

Proof. First, observe that any uv-path with length $c(P) \leq K$ serves as a certificate to verify the "yes"-answer of PRICE_LINKFAILURES in $O(|E|^2)$ time, so PRICE_LINKFAILURES $\in \mathcal{NP}$.

In order to prove that it is $\mathcal{NP}$-hard, we construct an instance $(G, u, v, \mu, \tilde{K})$ of PRICE_-LINKFAILURES from an instance (K_n, w_{ij}, K) of MAXCUT as follows: let G have $n + 1$ nodes. For every node $i \in V_n$, introduce edges $v_i, \bar{v}_i$ in G as in Figure 4.4 and let u and v be the first and last node in this chain, respectively.
For each pair of nodes $i, j \in V_n$ let

$$\mu_{v_i}^{\bar{v}_j} := \mu_{\bar{v}_i}^{v_j} := w_{ij}, \qquad \mu_{v_i}^{v_j} := \mu_{\bar{v}_i}^{\bar{v}_j} := 0,$$

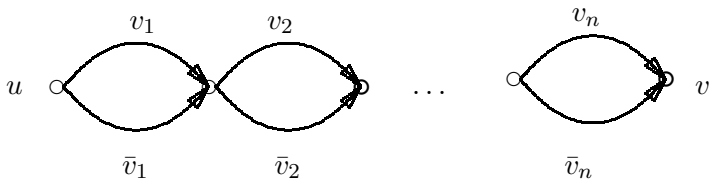

Figure 4.4: Graph for link failures

and let $\tilde{K} := \sum_{i,j \in V_n} w_{ij} - 2K$. There is a one-to-one correspondence between uv-paths P in G and cuts $\delta(A)$ in K_n by defining

$$i \in A \quad :\Leftrightarrow \quad v_i \in P \quad (\Leftrightarrow \bar{v}_i \notin P),$$

i.e., given a path P, the set A is defined to consist of those indices where P uses the upper of the two possible edges. Vice versa, every cut $\delta(A)$ defines a unique path by the above definition. By construction, the weight of a uv-path P_A as defined in PRICE_LINKFAILURES is

$$\begin{aligned} c(P_A) &= \underbrace{\sum_{v_i \in P_A} \sum_{v_j \notin P_A} \mu_{v_i}^{v_j}}_{=0} + \sum_{v_i \in P_A} \sum_{\bar{v}_j \notin P_A} \mu_{v_i}^{\bar{v}_j} + \sum_{\bar{v}_i \in P_A} \sum_{v_j \notin P_A} \mu_{\bar{v}_i}^{v_j} + \underbrace{\sum_{\bar{v}_i \in P_A} \sum_{\bar{v}_j \notin P_A} \mu_{\bar{v}_i}^{\bar{v}_j}}_{=0} \\ &= \sum_{i \in A} \sum_{j \in A} w_{ij} + \sum_{i \notin A} \sum_{j \notin A} w_{ij} \\ &= \sum_{i,j \in V_n} w_{ij} - 2 \sum_{i \in A} \sum_{j \notin A} w_{ij}. \end{aligned}$$

By definition of $\tilde{K}$, the value of this sum is at most $\tilde{K}$ if and only if the corresponding cut $\delta(A)$ satisfies $2 \sum_{i \in A} \sum_{j \notin A} w_{ij} \geq K$.

Hence, we have constructed an instance of PRICE_LINKFAILURES which has the answer "yes" if and only if the corresponding instance of MAXCUT has the answer "yes", which completes the proof.

□

Proposition 4.6 *The problem PRICE_NODEFAILURES is $\mathcal{NP}$-complete.*

Proof. As above, PRICE_NODEFAILURES is in $\mathcal{NP}$ since any path with length at most K may serve as a certificate for the "yes"-answer.

Given an instance (K_n, w_{ij}, K) of MAXCUT, we construct a corresponding instance $(G, u, v, \mu_e^w, \tilde{K})$ of PRICE_NODEFAILURES similar as above. Consider the graph shown in Figure 4.5. The upper nodes are $w_1, \dots, w_n$, the lower nodes $\bar{w}_1, \dots, \bar{w}_n$. The lower index of the edge names $v_i^k, \bar{v}_i^k$ indicates the number of its tail node ($1 \leq i \leq n$), the upper index is 1 if the edge leads to an upper node and 2 otherwise. The edges *originating* from an upper node are called v_i^k, the others $\bar{v}_i^k$. The edges $h_1, \dots, h_4$ are just supporting edges; the purpose of this construction will become clear soon.
Again, let $\tilde{K} := \sum_{i,j \in V_n} w_{ij} - 2K$. For $i, j \in V_n$ and $k = 1, 2$, let

$$\mu_{v_i^k}^{w_j} = \mu_{\bar{v}_i^k}^{\bar{w}_j} := w_{ij}, \qquad \mu_{v_i^k}^{\bar{w}_j} = \mu_{\bar{v}_i^k}^{w_j} := 0.$$

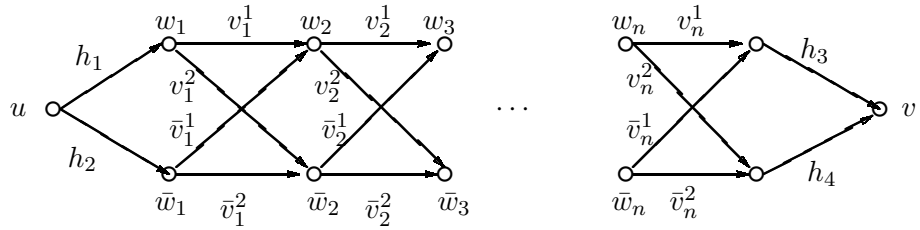

Figure 4.5: Graph for node failures

For the additional helper edges h_i at the beginning and end of the chain, let $\mu_{h_i}^{w_j} := 0$ for all $j \in V_n$. Again, there is a one-to-one correspondence between uv-paths P and cuts $\delta(A)$ by defining

$$i \in A \quad :\Leftrightarrow \quad w_i \in P \quad (\Leftrightarrow v_i^1 \in P \text{ or } v_i^2 \in P).$$

In other words, the set A consists of the indices where P employs the upper of the two *nodes* this time; the edges v_i^k are just a helping construction to make sure that a path can "choose" between the upper node w_i and the lower node $\bar{w}_i$ at each step.

By construction, the cost of a uv-path P_A is

$$c(P_A) = \sum_{k=1,2} \Bigg(\underbrace{\sum_{v_i^k \in P_A} \sum_{w_j \notin P_A} \mu_{v_i^k}^{w_j}}_{0} + \underbrace{\sum_{\bar{v}_i^k \in P_A} \sum_{\bar{w}_j \notin P_A} \mu_{\bar{v}_i^k}^{\bar{w}_j}}_{0} + \sum_{v_i^k \in P_A} \sum_{\bar{w}_j \notin P_A} \mu_{v_i^k}^{\bar{w}_j} + \sum_{\bar{v}_i^k \in P_A} \sum_{w_j \notin P_A} \mu_{\bar{v}_i^k}^{w_j} \Bigg)$$

Now for each i, the path P_A passes either by one of the edges v_i^k $(k = 1, 2)$ and thus by node w_i, or by or one of the edges $\bar{v}_i^k$ and by node $\bar{w}_i$. Recall that $i \in A$ in the first case and $i \notin A$ in the latter. Thus, the above sum can be written as

$$\begin{aligned} c(P_A) &= \sum_{i \in A} \sum_{j \in A} w_{ij} + \sum_{i \notin A} \sum_{j \notin A} w_{ij} \\ &= \sum_{i,j \in V_n} w_{ij} - 2 \sum_{i \in A} \sum_{j \notin A} w_{ij}, \end{aligned}$$

which is at most $\tilde{K}$ if and only if the weight of the cut $2\delta(A)$ is $\sum_{i \in A} \sum_{j \notin A} w_{ij} \geq K$. This completes the proof.

□

Now we are in a position to formulate the following result:

Corollary 4.7 *Using Path Restoration with stub release, the pricing problem is $\mathcal{NP}$-hard even when considering single node failures only or single edge failures only.*

Proof. Observe that the constructions above can be done in much the same way when the additional edge weights d_e are considered: we may simply set them to zero when constructing the instances of PRICE_LINKFAILURES and PRICE_NODEFAILURES.

□

4.4 Used cutting planes

As mentioned above, we separated cutting planes during the whole Branch and Bound procedure. This section shortly describes the inequalities which we used as cutting planes; for the separation algorithms, see Section 5.3 (at the exception of the metric inequalities, which will be fully discussed here).

For this section, consider the capacity polyhedron

$$Y = \text{conv}\{y \in \mathbb{R}^E_+ \mid y \text{ satisfies } (3.8), (3.9) \text{ and } (3.10)\}$$

defined at the beginning of this chapter and the MIP polyhedron

$$P_{MIP} := \text{conv}\{(x, y, f) \mid (x, y, f) \text{ satisfies } (3.1) - (3.6), (3.8), (3.9), (3.10)\}$$

of all feasible solutions to the network restoration problem.

Notice that since Y is a projection of P_{MIP} to a subset of variables, the following holds:

Observation 4.8 *Any inequality that is valid for Y is also valid for P_{MIP}.*

Suppose for instance $a^T y \geq b$ to be valid for Y and let $(\bar{x}, \bar{y}, \bar{f}) \in P_{MIP}$. This implies $\bar{y} \in Y$ by definition of P_{MIP}, which in turn leads to $a^T \bar{y} \geq b$ due to the validity of this inequality for Y, independently of $\bar{x}$ and $\bar{f}$. Thus, the inequality is also valid for P_{MIP}.

4.4.1 Metric inequalities

The separation of metric inequalities by means of the restoration LP has already been described in the previous sections, so we just address a related problem here:

Recall that a given capacity vector $\bar{y}$ allows a feasible routing if and only if it satisfies all metric inequalities whose coefficients fulfill certain restrictions (see Proposition 4.1). In particular, these metric inequalities are valid for Y.

A separator for a certain kind of inequalities is called *exact* if it reliably finds violated inequalities as long as there are some. In our case, where the separator for metric inequalities is the capacity feasibility test, this raises the question if the metric separation by means of the capacity feasibility test is exact or not. In fact, two such questions have to be answered:

1. Does the capacity feasibility test reliably detect infeasibility of $\bar{y}$?
2. Does it reliably detect feasibility of $\bar{y}$? In other words, is this metric separator exact?

The answer to the first question is *yes*. If the capacity vector does not allow a feasible routing, we will not be able to find a solution to the restoration LP, decide that the vector is infeasible and cut it off by a metric inequality, which is correct.

The answer to the second question depends on the restoration model. Applied to Path Restoration without stub release or to Reservation, the pricing problem can be solved exactly; as long as there is a violated dual inequality, we will find it. In other words, the pricing by shortest paths is exact in this case. Consequently, if the restoration LP is solvable and not unbounded, the algorithm described above will find an optimal solution of (P) and correctly decide that the capacity vector is feasible.

Using Path Restoration with stub release or Local Restoration, the pricing is not exact (at least in our implementation; recall that with Local Restoration, it could be done exactly

but we resorted to heuristics for reasons of calculation time). It may occur that no path violating a dual constraint is found, although there is one. Note that this can only happen with NOS paths, as the pricing of failure paths is exact. Otherwise stated, the algorithm can decide that the restoration LP has no solution although the capacities may be feasible. In this case, the current capacity vector is cut off by an invalid metric inequality. In particular, an optimal feasible solution can be rejected at this point.

Note, however, that by using the implementation techniques described in Section 5.2, the rejection of an optimal solution is possible but not very likely. In addition, our computational tests for small instances suggest that quite often, we actually calculated an optimal solution even with heuristic pricing (see Chapter 6 for details).

4.4.2 Strengthened metric inequalities

Consider a given metric inequality $\sum_{e\in E} \mu_e y_e \geq b$ which is valid for Y. For instance, this inequality could have been obtained by the feasibility test described above, or it could be a cut inequality, stating that the capacity on a cut must be sufficient to accommodate the demand that must be routed over the cut.

By a simple divide-and-round procedure that makes use of the integrality of the capacities, stronger inequalities can be derived. They are called *strengthened metric inequalities* and were introduced in [AGW96].

Recall that

$$y_e = \sum_{d\in\mathcal{D}(e)} C^d x_e^d.$$

Let $F := \{e \in E \mid \mu_e > 0\}$ be the support of the given metric inequality and let

$$g := \gcd\{\mu_e C^d \mid e \in F,\ d \in \mathcal{D}(e),\ \mu_e C^d < b\}$$

be the greatest common divisor of some suitable edge design capacities for the edges in F. Then the following proposition holds (see also [AGW96], [Wess00]):

Proposition 4.9 *Given a metric inequality $\sum_{e\in E} \mu_e y_e \geq b$ that is valid for Y, the inequality*

$$\sum_{e\in E} \min\left\{\frac{\mu_e C^d}{g}, \left\lceil\frac{b}{g}\right\rceil\right\} x_e^d \geq \left\lceil\frac{b}{g}\right\rceil$$

is valid for P_{MIP}.

Proof. By dividing every coefficient and the right-hand side by g, the metric inequality can be written as

$$\sum_{e\in E} \frac{\mu_e C^d}{g} x_e^d \geq \frac{b}{g}.$$

As the design variables x_e^d are binary, the coefficients $\frac{\mu_e C^d}{g} \geq \frac{b}{g}$ may be reduced to the right-hand side. By rounding them up, thereby slightly weakening the inequality, we obtain

$$\sum_{e\in E} \min\left\{\frac{\mu_e C^d}{g}, \left\lceil\frac{b}{g}\right\rceil\right\} x_e^d \geq \frac{b}{g}$$

where all coefficients have integer values. Using again that the design variables are binary, we see that the left-hand side of the last inequality is integer for every feasible capacity assignment. Thus, the right-hand side may be rounded up, which completes the proof.

□

4.4.3 Band and GUB cover inequalities

Suppose an index set $I = I_P \cup I_N$ with $I_P \cap I_N = \emptyset$, where P stands for *positive* and N for *negative*. With each $i \in I$ we associate a set $S_i = \{1, \ldots, |S_i|\}$ (referred to as a *GUB*) which we assume to be pairwise disjoint with each other. For a subset $K \subseteq I$ let $S(K) := \{(i, s) \mid i \in K,\ s \in S_i\}$ be the set of items in the GUBs in K. Each item $(i, s) \in S(I)$ has a weight $a_i^s \in \mathbb{R}\backslash\{0\}$ (items with zero weights can be eliminated in advance), where $a_i^s > 0$ for all $i \in I_P$ and $a_i^s < 0$ for all $i \in I_N$. Furthermore, for simplification of notation, we assume the weights to be ascendingly ordered within each GUB, i.e.

$$a_i^1 < \ldots < a_i^{|S_i|} \quad \forall i \in I.$$

For $x_i^s \in \{0, 1\}$, consider the explicit GUB constraints $\sum_{s \in S_i} x_i^s = 1$. They state that we have to choose exactly one variable in each GUB. If for some GUB S_j we have the constraint $\sum_{s \in S_j} x_j^s \leq 1$ instead of equality, we introduce a slack variable with zero weight which we put at the beginning of positive GUBs and at the end of negative GUBs to conserve the ascending ordering; choosing this slack variable effectively means choosing no variable in this GUB. Hence, from now on, we assume the GUB constraints to be of the form $\sum_{s \in S_i} x_i^s = 1$ in each GUB i.

Let $b \in \mathbb{R}$ be the knapsack capacity. We are interested in the knapsack polytope with explicit GUB constraints

$$P_{GUB} := \left\{ x \in \{0,1\}^{S(I)} \;\middle|\; \sum_{i \in I} \sum_{s \in S_i} a_i^s x_i^s \geq b,\ \sum_{s \in S_i} x_i^s = 1\ \forall i \in I \right\}.$$

A *GUB cover* is a set $C \subseteq S(I)$ satisfying

$$|C \cap S_i| = 1 \quad \text{for all } i \in I.$$

A cover is called *valid* (w.r.t. the weights a_i^s and the knapsack capacity b) if $a(C) := \sum_{(i,s) \in C} a_i^s < b$. For a given cover C, denote by $s_i \in S_i$ the element hit by the cover, i.e.

$$\{s_i\} := B \cap S_i \ \ \forall i \in I.$$

The elements $\{s_i\}_{i \in I}$ are also called *breakpoints*. With these definitions, the validity condition of a cover can be rewritten as

$$\sum_{i \in I} a_i^{s_i} < b.$$

A GUB cover $C_a = \{t_i\}_{i \in I}$ is *above* C if $t_i \geq s_i$ for all $i \in I$ and $t_j > s_j$ for at least one $j \in I$. A valid GUB cover C is *maximal* if there is no valid cover above C, i.e., by raising any breakpoint, C gets invalid.

Figure 4.6 illustrates this concept for the special case where $I_N = \emptyset$, I_P corresponds to four edges $\{e_1, \ldots e_4\}$ and the a_i^s correspond to the capacities C_e^d of the edge designs installable

on an edge. In the figure, the same three capacities C^1, C^2 and C^3 are installable on each edge; the higher the breakpoint of an edge in the picture, the higher the capacity chosen by the corresponding cover.

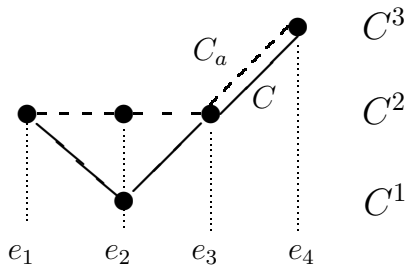

Figure 4.6: A cover C (solid) and another cover C_a (dashed) above C

Let C be a valid and maximal GUB cover w.r.t. b. Then the inequality

$$\sum_{i \in I_N} \sum_{s \leq s_i} x_i^s - \sum_{i \in I_P} \sum_{s > s_i} x_i^s \leq |I_N| - 1. \tag{4.12}$$

is valid for P_{GUB}. This class of inequalities was introduced by Wolsey [Wol90]. The above inequality states that when choosing an item with high negative weight in each negative GUB, we have to choose an item as least as heavy as the corresponding cover item in at least one positive GUB in order to exceed the knapsack capacity, which holds by definition of a valid cover. Observe that while the inequality remains valid if B is not maximal, it is stronger for a maximal cover.

We used this class of inequalities in two variants:

1. **Node capacity constraints:** In this case, the underlying knapsacks $a^T x \geq b$ are the Inequalities (3.3),

   $$\sum_{d \in \mathcal{D}(v)} C^d x_v^d - \sum_{e \in \delta(v)} \sum_{d \in \mathcal{D}(e)} C^d x_e^d \geq 0 \quad \forall\, v \in V$$

   from the hardware model. For these constraints, we have $|I_P| = 1$ and $|I_N| = |\delta(v)|$, and the a_i^s correspond to the installable node designs and edge designs.

   A GUB cover corresponds to the choice of a particular edge design $\bar{d}_e$ on each edge $e \in \delta(v)$ and of a node design $\bar{d}_v$ for node v.

   The resulting inequality,

   $$\sum_{e \in \delta(v)} \sum_{\substack{d \in \mathcal{D}(e): \\ C^d \geq C^{\bar{d}_e}}} x_e^d - \sum_{\substack{d \in \mathcal{D}(v): \\ C^d > C^{\bar{d}_v}}} x_v^d \leq |\delta(v)| - 1,$$

   states that choosing the cover capacity on every edge, a node design with higher capacity than the cover capacity has to be installed.

   The practical benefit of these inequalities seems to depend heavily on the test examples. For some of our examples, they helped to significantly speed up the calculation process, whereas for many examples, nearly no such inequalities were found, and if some were found then this usually happened very late in the calculation process.

2. **Band inequalities:**

   The other special case of GUB cover inequalities which we used are the so called *band inequalities* introduced by Dahl and Stoer [DS93]. In this case, $I_N = \emptyset$ and each positive GUB corresponds to an edge of some subset $F \subseteq E$.

   Again, a GUB cover (also called *band* in this context) is a particular choice of edge design $\bar{d}_e$ for each edge $e \in F$. The base knapsacks are metric inequalities of the form

   $$\sum_{e \in F} \mu_e \sum_{d \in \mathcal{D}(e)} C^d x_e^d \geq b,$$

   where

   $$\mu_e := \sum_{\substack{s \in S: \\ e \in E^s}} \mu_e^s \geq 0, \quad F := \{e \in E \mid \mu_e > 0\} \quad \text{and} \quad b := \sum_{uv \in D} d_{uv} \pi_{uv}.$$

   Hence, the a_i^s correspond to $\mu_e C^d$ for some edge design $d \in \mathcal{D}(e)$ (possibly a dummy design for the slack variable, since not every edge has to be equipped with a edge design).

   As a special case of a metric inequality, the base knapsacks could also be a cut inequality for some cut $\delta(W) \subseteq E$ where

   $$F = \delta(W), \quad \mu_e = \begin{cases} 1 & \text{if } e \in F, \\ 0 & \text{if } e \notin F, \end{cases}$$
   $$b = \text{ total demand value to be routed over } \delta(W),$$

   stating that the capacities on the cut must be sufficient to accommodate the demand to be routed over the cut.

   The definition of validity of a band can be rewritten as

   $$\sum_{e \in F} \mu_e C^{\bar{d}_e} < b$$

   in this context. For cut inequalities, the intuitive meaning of a valid band is that the capacities chosen by the band are not sufficient to accommodate the demand that has to be routed over the cut.

   In either case, the resulting inequality is a *band inequality* of the form

   $$\sum_{e \in F} \sum_{\substack{d \in \mathcal{D}(e): \\ C^d > C^{\bar{d}_e}}} x_e^d \geq 1,$$

   stating that for a particular choice of capacities on F, the capacity of at least one edge $e \in F$ must be augmented compared to the band in order for the base knapsack to be satisfied.

   Dahl and Stoer showed band inequalities to be facet defining for P_{GUB} if and only if $|F| \geq 2$ and the underlying band is maximal (see [DS93]).

Chapter 5

Implementational aspects

5.1 Initialization

As mentioned earlier, we first solved (P) with the α variable and a restricted number of path variables and generated further variables as needed. In this chapter, some heuristics for the computationally difficult parts of pricing are presented as well as details on the initial path calculation and separation algorithms for the used cutting planes.

5.1.1 Initial path calculation

The collection of path variables in the initial LP has to satisfy two aspects. On the one hand, it must allow a solution to the restricted LP; on the other hand, the initial path variables should have a high chance to be part of an optimal solution of (P) in order to avoid generating too many unnecessary path variables and to reduce the size of the linear program.

To allow a solution to the initial LP, we first determined one or more routing paths for every NOS demand to satisfy the demand constraints. Afterwards, we calculated one failure path for each failure commodity induced by these NOS paths in order to satisfy the flow saving constraints as well. The capacity constraints were always fulfilled for some suitable value of α.

As the goal was to minimize capacity cost, we tried to find short paths (for some suitable definition of *short*) because longer paths tend to raise capacity cost. Therefore, for every demand from u to v, we applied a k-shortest paths algorithm with some suitable weights c_e, yielding uv-paths $P_1, \ldots, P_k$ such that $c(P_1) \leq \ldots \leq c(P_k) \leq c(P)$ for all other paths $P \in \mathcal{P}^0_{uv}$. The k-shortest paths algorithm which we used is based on [AMMP90].

We tried different edge weights c_e:

1. One choice was $c_e := 1$ for all $e \in E$. With this definition, the length of a path is just the number of its edges.

2. In real world networks, every edge has some length. Thus, we also tried setting c_e to the length in km of edge $e \in E$. This definition seems to be appropriate when the length dependent part dominates the fix capacity cost of an edge.

Another interesting aspect is the decision how many paths should be generated initially and by pricing, respectively. On the one hand, generating more initial paths may reduce the

time spent on pricing; on the other hand, it enlarges the size of the LPs and thus the time to solve them.

Details on the performance of different intensities of initial path calculation and different edge weights can be found in Section 6.5.1.

5.1.2 Delayed constraint generation

Starting only with a relatively small set of path variables, writing all inequalities into the Path LP from the beginning would yield lots of redundant or useless constraints. For instance, a capacity constraint (3.10) for an edge e takes the form $-\alpha \leq \bar{y}_e$ as long as none of the paths currently in the LP contains e. This inequality is certainly correct but redundant. Similarly, a flow saving constraint (3.9) states the trivial inequality $0 \leq 0$ as long as no path variable in the current LP shares a nonzero coefficient with this constraint.

Thus, we added constraints to the LP only at the time when the first nonzero coefficient in the corresponding row was created by a newly added path variable.

It then turned out that still too many unnecessary constraints were contained in the LP:

1. Recall that the coefficient of the NOS path variables is 1 in the NOS capacity constraints and 1 or 0 in the failure state capacity constraints. Thus, as long as a capacity constraint for edge e in a failure state contains only NOS path flow variables, it is dominated by the NOS capacity constraint for this edge. Therefore, the failure capacity constraints need only be added upon arrival of the first failure flow variable containing the corresponding edge.

2. As long as a flow saving constraint contains only failure path variables, it states that the sum of these failure flow variables should be nonnegative. This is redundant since the variables have to be nonnegative anyway. Hence, we need to add these rows only when the first NOS path variable creates a nonzero coefficient.

The redundant flow saving inequalities could be avoided in advance by the fact that we computed the initial failure paths based on previously determined initial NOS paths.

On the contrary, between 50% and 90% of all constraints in the initial path LP were dominated failure capacity constraints. By creating these constraints only upon arrival of the first failure path variable, we could drastically reduce the size of the LP. With Path Restoration, this halved the overall computation time.

5.2 Pricing heuristics

We solved the pricing problem exactly only when applying Path Restoration without stub release. Using Path Restoration with stub release or Link Restoration, we resorted to heuristics; those used in our implementation will be presented now. Note that while there is no performance guarantee for any of these heuristics, they performed quite well in practice.

5.2.1 k-shortest paths

Recall that a part of the pricing problem consists of determining whether for some $uv \in D$, there is a uv-path P violating a dual constraint of the form

$$\sum_{e \in P} \sum_{\substack{s \in S: \\ e \in E^s}} \gamma^s(P) \mu_e^s + \sum_{s \in S \setminus \{0\}} \sum_{\substack{c \in C^s \\ P \in \mathcal{IP}_c^s}} \rho_{uv}^s \beta_c^s \geq \pi_{uv}.$$

Using Path Restoration with stub release, the edge weights

$$c_e := \sum_{\substack{s \in S: \\ e \in E^s}} \gamma^s(P) \mu_e^s$$

depend on the uv-path P which we want to find. Consequently, we cannot simply solve a shortest path problem with edge weights c_e.

We addressed this problem by ignoring the path dependent part of $\gamma^s(P)$ and approximating the weights c_e by path independent weights

$$d_e := \sum_{\substack{s \in S: \\ e \in E^s}} \mu_e^s$$

for all edges, as for Path Restoration without stub release.

We then applied a k-shortest paths algorithm to the edge weights d_e as described on Page 59.

Afterwards, the length of the found paths was evaluated with respect to the correct (path dependent) weights c_e. If the shortest of these lengths was smaller than the corresponding dual demand value π_{uv}, a violated path had been found. Note that this method does not guarantee that if there is a violated path, it will be found. However, hope is that for a well chosen value of k, a violated path can be found with high probability without affecting overall computation time too much.

We tried values of k between 1 and 30; the results are given in Section 6.5.1.

5.2.2 Node weights with Link Restoration

Recall that the network may contain parallel edges, so the notation sw refers to the whole set of (directed or undirected) edges from s to w in this section.

A problem specific to Link Restoration is finding a path P that minimizes the sum

$$\sum_{w \in V \cap S} \sum_{\substack{s,t \in \Gamma(w), s \neq t \\ s \to w \to t \subseteq P}} \rho_{uv}^w \beta_{st}^w$$

among all uv-paths. In this case we have path dependent node weights $c_{st}^w := \rho_{uv}^w \beta_{st}^w$, so we cannot simply apply a shortest path algorithm if node failures are considered. Note that with edge failures only, this sum vanishes entirely and poses no problem. On the contrary, with node failures, we have to find edge weights d_e^w for every edge $e \in \delta(w)$ such that the weight of a shortest path w.r.t. edge weights $d_e^s + d_e^w$ (where $e \in sw$) approximates the correct weight of the path as closely as possible.

Let $\Gamma(w)$ be the set of adjacent nodes of w and let $\Gamma^+(w) \subseteq \Gamma(w)$ be the set of nodes by which a path can leave w, i.e.,

$$\Gamma^+(w) := \begin{cases} \Gamma(w) & \text{if } G \text{ is undirected,} \\ \{u \in \Gamma(w) \mid (wu) \in E\} & \text{if } G \text{ is directed.} \end{cases}$$

Similarly, let $\Gamma^-(w)$ be the set of adjacent nodes of w by which a path can enter node w, i.e.,

$$\Gamma^-(w) := \begin{cases} \Gamma(w) & \text{if } G \text{ is undirected,} \\ \{u \in \Gamma(w) \mid (uw) \in E\} & \text{if } G \text{ is directed.} \end{cases}$$

Note that $\Gamma^-(w)$ and $\Gamma^+(w)$ need not be disjoint even if G is directed since it can contain both edges sw and ws.

If $|\Gamma(w)| = 2$, a path containing w as an inner node has only one possibility, namely passing by the two adjacent nodes of w. Thus, given that w has to be an inner node of the paths we want to find, we may safely assign half the node weight to each incident edge of w and obtain the correct weight for w for every path passing through it.

For the nodes of higher degree, let $s \in \Gamma^-(w)$ and $t \in \Gamma^+(w)$ and consider the average contribution of edge $e = sw \in P$ to the above sum.[1]

Given that path P enters node w by node s, let p^w_{st} be the probability that P leaves w by t (disregarding for the moment how to calculate this value); in particular, $\sum_{u \in \Gamma^+(w) \setminus \{s\}} p^w_{su} = 1$ since w was assumed to be an inner node of the path and we excluded node repetitions in the path.

With this definition, we used the value

$$\sum_{u \in \Gamma^+(w) \setminus \{s\}} p^w_{su} c^w_{su}.$$

as an approximation of the contribution of sw to the weight of a path passing by w.

If the graph G is directed, we may assign the weight

$$d^w_e := \sum_{u \in \Gamma^+(w) \setminus \{s\}} p^w_{su} c^w_{su}.$$

to each edge $e \in sw$. For undirected graphs, we don't know if a path passing by sw enters or leaves w by this edge. Thus, we model the node weight as the sum of two edge weights (sw and the edge on which the path leaves w), and assign to edge sw the weight

$$d^w_e := \frac{1}{2} \sum_{u \in \Gamma^+(w) \setminus \{s\}} p^w_{su} c^w_{su}.$$

The following observation supports the assumption that this is a good heuristic for our purposes, as at least the sparse ones among our telecommunication graphs had several nodes of degree 2.

Proposition 5.1 *If $|\Gamma(w)| = 2$, choosing edge weights as above yields the correct weight of w for every path passing through w.*

[1] Actually, $e = sw$ refers to the whole set of all parallel edges from s to w in this context.

Proof. Let s and t be the adjacent nodes of w. For undirected graphs, the set $\{t\} = \Gamma^+(w)\backslash\{s\}$ is a singleton, which yields $p^w_{st} = 1$ since a path entering w by s has only one choice: leaving it by node t. This leads to $d^w_{sw} = \frac{1}{2}c^w_{st}$. Similarly, the equation $d^w_{wt} = \frac{1}{2}c^w_{st}$ holds. If the edges sw and wt are directed, the same argumentation yields $d^w_{sw} = c^w_{st}$ and $d^w_{wt} = 0$.

Finally, every path passing by w has to pass by sw and wt, which completes the proof since the sum of these edge weights is $c^w_{st} = d^w_{sw} + d^w_{wt}$ in either case.

□

Note that if $\Gamma^+(w)\setminus\{s\}$ is empty, there can be no simple path passing through w without ending there. In this case, the contribution of w to the pricing sum is 0.

For the other nodes, we have not yet considered how to determine the probabilities p^w_{st} for $s \in \Gamma^-(w)$ and $t \in \Gamma^+(w)$. We tried three variants:

1. *Uniform distribution* on the adjacent nodes of w:

$$p^w_{st} := \frac{1}{|\Gamma^+(w)\setminus\{s\}| - 1}.$$

   Node s is excluded from $\Gamma^+(w)$ as we disallowed paths with loops.

2. *Degree weighted probabilities:* the idea is that the higher the degree of an adjacent node $t \in \Gamma^+(w)$, the higher the probability that a given path visits t. Thus, we set

$$p^w_{st} := \frac{\deg(t)}{\sum_{u\in\Gamma^+(w)\setminus\{s\}} \deg(u)},$$

   where $\deg(v) := |\delta(v)|$ is the degree of v.

3. *Edge length weighted probabilities:* another heuristic argument is that "good" paths in network restoration problems are often short with respect to their length in km since long paths lead to high edge capacity cost. Hence, we prefer short outgoing edges to long ones. For $t \in \Gamma^+(w)$, let l_{wt} be the length in km of the longest edge in wt. For $s \in \Gamma^-(w)$, let

$$M_s := \max\{l_{wt} \mid t \in \Gamma^+(w)\setminus\{s\}\}$$

   be the length in km of a longest outgoing edge from w and let

$$p^w_{st} := \frac{M_s - l_{wt}}{\sum_{u\in\Gamma^+(w)\setminus\{s\}}(M_s - l_{wu})}$$

   provided that $M_s > 0$. The case $M_s = 0$ means that all outgoing edges of w (except those leading to s) have length 0 (which is extremely unlikely in real world problems). In this case, resort to uniform distribution.

In our computational tests, we combined these approximated node weights with a k-shortest paths algorithm. It turned out that degree and length weighted probability calculation very slightly outperformed uniform probabilities in terms of quality of the obtained solutions while needing slightly more time. However, this was not always the case, and the differences were not significant. The value of k was had no influence at all on the obtained solution values and very little influence on total running time.

We also tried to apply all three probability variants one after another in each pricing step. This was done in the hope that if a violated path was not found by one probability variant, it would perhaps be found by another one. It turned out that applying all three variants had no effect on the quality of the found solutions but doubled or even tripled the overall solution time. Details can be found in Section 6.5.1.

5.3 Separation algorithms for cutting planes

5.3.1 (Strengthened) metric inequalities

The separation of metric and strengthened metric inequalities has already been discussed in detail in Chapter 4. As explained there, a violated metric inequality cutting off the current capacity vector was obtained from the objective function of the dual LP each time the capacity feasibility tester decided the capacity vector to be infeasible even with the whole set of paths.

For the separation of strengthened metric, band and GUB cover inequalities, we held a pool of valid inequalities (mostly metric and cut inequalities) which served as base knapsacks for the separation.

5.3.2 Band and GUB cover inequalities

As band inequalities are a special case of GUB cover inequalities, we used the same separation algorithm for both, as described in the following.

Suppose a knapsack $\sum_{i\in I}\sum_{s\in S_i} a_i^s x_i^s \geq b$ with GUB constraints $\sum_{s\in S_i} x_i^s = 1$ for all $i \in I$ (after introduction of a slack variable if necessary) that is valid for P_{GUB}. Given a fractional point $\bar{x}$, the goal is to separate a GUB cover inequality

$$\sum_{i\in I_N}\sum_{s\leq s_i} x_i^s - \sum_{i\in I_P}\sum_{s>s_i} x_i^s \leq |I_N| - 1$$

(as described in Section 4.4.3) separating $\bar{x}$ from P_{GUB}.

We want to find a valid cover $C = \{s_i\}_{i\in I}$ maximizing the violation of the above inequality w.r.t. $\bar{x}$, i.e., maximizing the value

$$\begin{aligned} 1 + \sum_{i\in I_N}\sum_{s\leq s_i} \bar{x}_i^s - \sum_{i\in I_P}\sum_{s>s_i} \bar{x}_i^s - |I_N| & \\ = 1 + \sum_{i\in I_N}(\sum_{s\leq s_i} \bar{x}_i^s - 1) - \sum_{i\in I_P}\sum_{s>s_i} \bar{x}_i^s & \\ = 1 - \sum_{i\in I}\sum_{s>s_i} \bar{x}_i^s & \end{aligned}$$

(recall that $\sum_{s\in S_i} \bar{x}_i^s = 1$ and thus $\sum_{s>s_i} \bar{x}_i^s = 1 - \sum_{s\leq s_i} \bar{x}_i^s$). This is equivalent to minimizing

$$\sum_{i\in I}\sum_{s>s_i} \bar{x}_i^s - 1 = \sum_{i\in I} c_i^{s_i} - 1 \quad \text{where} \quad c_i^t := \sum_{s>t} \bar{x}_i^s \quad \forall i \in I,\ t \in S_i.$$

The value $-c_i^t$ is the value by which item (i,t) contributes to the violation if breakpoint $s_i = t$ is chosen in GUB i. With this definition of c_i^t, we can formulate the Multiple Choice Knapsack

(MCK) problem

$$\begin{aligned} \min \sum_{i\in I}\sum_{t\in S_i}(\sum_{s>t}\bar{x}_i^s)y_i^t & \\ \sum_{i\in I}\sum_{s\in S_i} a_i^s y_i^s &< b \\ \sum_{s\in S_i} y_i^s &= 1 \quad \forall i \in I \\ y_i^s &\in \{0,1\} \qquad \forall i \in I,\ \forall s \in S_i \end{aligned}$$

The constraints of this MCK problem are just the definition of a valid GUB cover w.r.t. a_i^s and b; the objective function maximizes violation of the resulting band inequality w.r.t. $\bar{x}$ over all valid GUB covers.

There is a one-to-one correspondence between valid covers for the knapsack capacity b and the weights a_i^s and solutions of the above MCK problem: any solution y defines a valid cover by

$$C := \{(i,s) \mid i \in I,\ y_i^s = 1\},$$

and every valid cover defines a solution of the MCK problem by $y_i^s = 1$ if $(i,s) \in C$ and $y_i^s = 0$ else.

Note that a valid cover which is optimal w.r.t. violation is not necessarily a maximal cover since there may be a higher cover (i.e. a cover $\{t_i\}_{i\in I}$ with $t_i \geq s_i$ for all $i \in I$ and $t_j > s_j$ for at least one $j \in I$) with the same objective function value.

In order to achieve a maximal cover, we pass once or more over all GUBs, raising the breakpoints as long as possible, i.e., while the capacity d is not exceeded. Note that the objective value of the MCK (and thus the violation of the separated inequality) must remain the same during this procedure; the sole purpose of maximizing the cover is strengthening the separated inequality for the further course of the Branch and Bound algorithm.

In any case, given a cover C (maximal or not) defined by a solution of the MCK problem, the violation of (4.12) w.r.t. $\bar{x}$ can be obtained by calculating

$$1 - \sum_{i\in I} c_i^{s_i} = 1 - \sum_{i\in I}\sum_{s>s_i} \bar{x}_i^s$$

by the way the c_i^s were defined. If this value is positive, then a separated GUB cover inequality has been found and can be used to cut off $\bar{x}$ from P_{GUB}. On the contrary, if the separated inequality is not violated or if the MCK problem has no solution, we have to retry the procedure with another cover or another knapsack.

Observe that the order in which the GUBs are processed during maximization is arbitrary for getting a maximal cover. From a computational point of view, it is not clear in which cases it serves better to raise the breakpoints uniformly and in which cases one should better raise the breakpoints in one GUB while possible and then proceed to the next one. These two ways may lead to different separated inequalities and thus to a different continuation in the Branch and Bound process.

In our implementation, we tried both ways of augmenting the breakpoints; neither way clearly outperformed the other one in terms of total running time or total number of iterations. Since solving the multiple choice knapsack with dynamic programming was quite time consuming, we also tried to start with a zero cover (choosing the lowest breakpoint in

each GUB) and to raise the breakpoints greedily as long as possible with either of the two approaches to raise them. It turned out that in terms of overall solution time, it was worth spending the additional time for solving the MCK problem exactly as more violated cutting planes were found this way.

Chapter 6

Computational results

This chapter presents the computational tests done with our implementation of the model. After an overview on the tests and some preliminaries applying to all of them, our test instances together with their characteristics are presented. Afterwards, the details of each test together with the corresponding numerical results are given, accompanied by a discussion of these results. The chapter terminates with a summary and an outlook on possible further research.

6.1 Overview on computational tests

We tested our implementation of the mathematical model discussed in Chapter 3 against some real world telecommunication networks provided to us by several telecommunication providers.

Our computational tests consist of two main parts:

1. The first part is a comparison of optimal values of sparse test instances. This comparison includes the minimal network cost when the network is designed for NOS only, single node failures, single edge failures or all single failures. The applied restoration mechanism is either Link Restoration or Path Restoration with or without stub release. In connection with this comparison, the effects of the discrete cost structure are investigated by scaling the demand values on these sparse examples.

2. In the second part, the model is applied to networks of higher density. Since in these cases, we had difficulties to find good solutions at all in reasonable time, several heuristics are tested on these examples and compared with each other and with the best known exact values w.r.t. both solution quality and time.

All calculations were done on a computer with an 1667 MHz processor with 1 GB of RAM. For solving the intermediate LPs (in the Branch and Bound tree) and MIPs (for the MIP heuristic), CPLEX 8.0 was used. In order to limit the number of tests, we restricted ourselves to undirected graphs with full restoration, i.e., $\rho^s_{uv} = 1$ for all NOS demands $uv \in D$ and all operating states $s \in S$.

6.2 Test instances

6.2.1 Generalities

All test instances are based on real world examples provided to us by different providers. The structure of these examples varies widely:

- Some of these instances have node designs (and thus node restrictions and node costs), others have not.
- The number of installable edge designs varies between 2 and 9, and the cost structures are very different. In some instances, cost is linear or nearly linear with capacity, whereas most instances have cost structures which are far from being linear, as illustrated in Figure 6.1. In several cases, km-cost is not only degressive with capacity but also with the length of an edge.

  Note that cost structures as in the right part of 6.1, where a higher capacity leads to lower cost, cannot always be avoided by preprocessing, since edge capacity has influence on node cost as well and is restricted by the availability of interfaces, slots, modules, and node designs.
- The density of the networks varies between 2.3 and 9.
- The number of demands and the structure of the demand values (spectrum of demand values, proportion of demand values to installable capacities,...) are quite different among the instances.

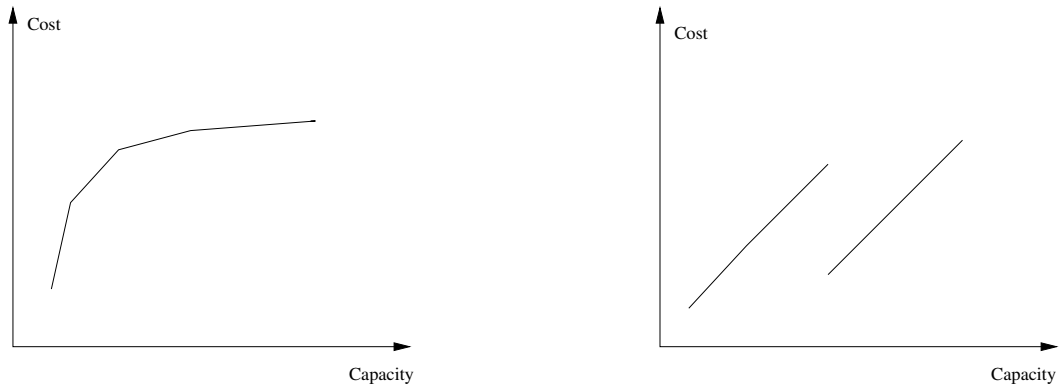

Figure 6.1: Common capacity cost structures

Table 6.1 shows the characteristics of our test instances: the number of nodes, edges and demands, the average nodal degree $\bar{d}_E = 2|E|/|V|$, the maximum number ld of edge designs installable on any edge and some remarks on the cost structure.

The first part of the table contains the instances that could be solved to optimality within an hour of CPU time, the second part shows the instances where we had difficulties to find good solutions at all. Unless otherwise stated, node designs pose no restrictions, and their cost is not taken into account. When a hop limit is given, it applies to all NOS demands; otherwise, there are no hop limits.

Sometimes, these examples will also be referred to by their number of nodes, edges and demands, like 10/25/45 for s1.

No.	$\|V\|$	$\|E\|$	$\|D\|$	$\bar{d}_E$	ld	cost structure
s1	10	25	29	5.0	2	Fix/km-cost nearly linear with capacity, hop limit 4.
s2	12	18	27	3.0	4	No fix cost, km-cost degressive with capacity and length.
s3	15	21	13	2.8	3	No fix cost, km-cost heavily degressive with capacity and length.
s4	15	22	105	2.9	7	No fix cost, km-cost nearly linear with capacity.
s5	18	21	62	2.3	9	Fix/km-cost piecewise linear with capacity. See Section 6.2.2 for details.
s6	18	27	62	3.0	9	Fix/km-cost piecewise linear with capacity. See Section 6.2.2 for details.
s7	20	28	119	2.8	6	Two node designs. See Section 6.2.2 for details.
d1	10	45	45	9.0	2	Fix/km-cost linear with capacity, hop limit 2. Complete graph.
d2	11	34	24	6.2	3	No fix cost, km-cost degressive with capacity and length.
d3	11	47	55	8.5	2	Progressive fix cost, km-cost linear with capacity. Half complete graph.
d4	13	28	45	4.3	3	No fix cost, km-cost degressive with capacity and length.
d5	14	32	43	4.6	8	No fix cost, km-cost degressive with capacity and length.
d6	15	55	36	7.3	2	No fix cost, km-cost degressive with capacity and length.
d7	19	34	33	3.6	8	No fix cost, km-cost degressive with capacity and length.

Table 6.1: Test instances

6.2.2 Special properties of some test instances

Some test instances are somewhat particular and deserve special attention in order to understand the numerical results:

- Instances s5 (18/21/62) and s6 (18/27/62) provide capacities 1, 2, 4, 8, 12, 16, 32 and 64 with a cost structure as shown in the second part of Figure 6.1. The fact that fix cost has a similar structure yields that for many edges, installing capacity 16 is cheaper than installing capacity 3.

- Instance s5 (18/21/62) is the same as s6 (18/27/62) with some edges removed.

- Instance s7 (20/28/119) has capacity/cost values of 1/1, 2/2, 3/3, 4/1, 8/2 and 12/3 (all values scaled appropriately) and no fix edge cost. This leads to a cost structure similar to the one shown in the second part of Figure 6.1. In particular, the only cost difference

between capacities 1 and 4 is caused by node hardware and thus independent of edge length, and installing capacity 4 is only slightly more expensive than installing capacity 1. The same holds for capacities 2 and 8 and for 4 and 12, respectively. The capacities of the installable node designs at each node are 4 and 8; node cost is degressive with node capacity.

- Instance s1 (10/25/45) is the same as d1 (10/45/45) with some edges and demands removed and the hop limits raised from 2 to 4.

- The capacity cost structure of instances s2, s3, s4, d2, d4, d5, d6 and d7 is similar to the one shown in the first part of Figure 6.1.

6.3 Cost comparison of restoration mechanisms

6.3.1 Results and observations

Among other criteria like maintenance and administration effort, the capacity cost necessary to implement a given restoration mechanism is of major interest to a network provider who is going to plan a failure resistant network.

Thus, we compared the optimal cost values of the small instances s1 to s7 with Link Restoration (LR), Path Restoration without stub release (PR-sr) and Path Restoration with stub release (PR+sr), each with single link failures only, single node failures only or both. In order to know which part of the costs is due to spare capacity for securing of the network against failures, we also compared these values with the costs of the same network designed for a cost minimal NOS capacity assignment.

The first part of Table 6.2 (the columns *Joint optimization*) shows the results. Due to a nondisclosure agreement and in order to better highlight the relations between the values, all cost values are scaled such that the NOS value of the corresponding instance is 100.

Strictly spoken, we do no know of all values whether they are really optimal since for Path Restoration with stub release and for Link Restoration, we solved the pricing problem by means of heuristics (for Path Restoration without stub release, the pricing problem could be solved exactly).

However, we have strong reason to believe that no optimal solution has been accidently cut off by a wrong metric inequality: recall that we had different heuristics for solving the pricing problem: a k-shortest paths algorithm (see Section 5.2.1) and different node probability heuristics for Link Restoration (see Section 5.2.2). For each scenario, we tried every value of $k \in \{1, 5, 10, 20, 30\}$, every node probability heuristic alone and all three of them together. All these combinations were in turn combined with different edge weights (uniform or edge length based) in the initial path calculation and with minimal, medium or heavy initial path calculation each (see also Section 6.5.1). The fact that we obtained the same solution values in all these tests strongly suggests that these values are indeed optimal.

The NOS value for s4 is not known to be optimal, but at least within 1.8% to the optimal value. As a result, the ratio of the network cost with restoration to the network cost without restoration may be slightly underestimated.

There is a number of interesting observations to make, some of which we did not expect at all:

No.	NOS	model	Joint optimization			shortest hop + SCA			shortest length + SCA		
			node	link	both	node	link	both	node	link	both
s1	100	LR	128	132	132	305	305	305	371	371	371
		PR-sr	128	132	132	305	305	305	371	371	371
		PR+sr	128	132	132	305	305	305	371	371	371
s2	100	LR	182	140	183	206	206	206	206	151	206
		PR-sr	182	140	183	195	161	206	195	151	206
		PR+sr	182	140	183	195	161	206	195	151	206
s3	100	LR	128	162	162	136	184	184	-	188	-
		PR-sr	128	177	177	133	188	188	142	180	180
		PR+sr	128	162	162	133	166	166	142	166	166
s4	100	LR	245	244	246	251	267	269	257	268	269
		PR-sr	245	244	246	251	250	252	257	250	258
		PR+sr	245	244	246	245	246	246	255	250	257
s5	100	LR	113	113	113	140	140	140	138	139	139
		PR-sr	113	113	113	140	140	140	138	138	138
		PR+sr	113	113	113	140	140	140	138	138	138
s6	100	LR	114	114	114	182	183	182	180	185	185
		PR-sr	114	114	114	180	183	182	179	179	179
		PR+sr	114	114	114	180	183	181	179	179	179
s7	100	LR	143	171	171	189	196	196	178	183	185
		PR-sr	143	171	171	189	196	196	178	183	185
		PR+sr	143	171	171	189	196	196	178	183	184

Table 6.2: Optimal values with joint and successive optimization

- In six out of seven instances (all but s3), we obtained the same solution value for all three restoration models. In all seven instances, the cost of implementing Link Restoration was the same as for Path Restoration with stub release. This was quite surprising to us, all the more as all investigated networks are sparse; several authors (e.g., [IMG98]) have found that especially for sparse networks, the cost savings of Path Restoration compared to Link Restoration can be significant. Possible reasons for this discrepancy are examined in Section 6.4.

- In only one instance (s3), the application of stub release caused a cost difference. In this instance, Path Restoration without stub release was more expensive than Link Restoration (recall that this is possible since with Link Restoration, failure capacity is only reserved for the local failure paths, not for the whole backup paths).

- In three instances (s1, s3 and s7), the cost of link failures was higher than the cost of node failures but as high as the cost of all single failures. In two instances (s5 and s6), all failure situations caused the same cost, and in the remaining two instances (s2 and s4), link failures were cheaper than node failures, which in turn were slightly cheaper than all single failures.

  A node failure usually affects more edges and paths than a link failure but often causes several demands to be lost. It is not immediately clear from the results which of these

effects has more influence on capacity cost. However, observe that in five out of seven cases, link failures cause the same cost as all single failures. For node failures, this holds only for two instances (s5 and s6). This observation indicates that the link failures might dominate cost and that it is usually more expensive to protect a network against link failures than against node failures. This would be in accordance with the results of [PD98] who observed a higher cost for link failures in five out of six test instances. We further investigate this issue in the following section.

- In the instance s4, the additional cost for restoration compared to a network designed only for NOS was extraordinarily high (245%) compared to the other instances. This can at least partially be explained by the cost structure: this instance provides the capacities $1, 2, 4, 8, 16, 32$ and 64 for each edge. With restoration, capacity 64 had to be installed on many edges although the maximal flow on this edge was only slightly above 32, such that very few edges were saturated in any failure state. On the contrary, with the network being designed for NOS only, about two out of three edges were saturated (with capacity 8 or 16). Thus, despite the quasi-linearity of capacity cost in this instance, the total network cost was very large with restoration due to the limited choice of edge designs.

- With restoration, all of these instances could be solved to optimality within at most an hour, sometimes only within a few minutes. With node failures only, the calculation times were shortest. The instances which needed the most time were s4 (15/22/105) and s7 (20/28/119), probably due to the relatively high number of demands.

  An interesting effect occurred upon calculation of the NOS values without restoration (in the first column of Table 6.2): while most instances needed significantly less solution time without restoration, this was not the case for the instances s4 and s6. Instance s6 required four days to be solved to optimality without restoration; for s4, after the best known solution had been found after half an hour of calculation time, the algorithm spent another 15 days trying to prove its optimality by raising the lower bound. This indicates that the use of further cutting planes should be tried in order to obtain faster improvements of the lower bound.

6.3.2 Which kind of failures dominates cost?

In order to see if capacity cost is dominated mainly by node failures or by link failures, we investigated the solutions obtained when securing a network against all single failures. More specifically, we looked in which failure states the maximal failure flow on an edge was attained and if these failure states were node or link failures.

It turned out that link failures were largely dominating the maximal failure flow. In many cases, the maximal failure flow was attained in several failure states, but often all these failure states were link failures. It was very rare that the maximal failure flow was attained in a node failure state only. Interesting enough, this result holds as well for instance s2 (12/18/27) where node failures are significantly more expensive than link failures.

An interesting result that can be seen from Table 6.2 was that whenever link failures were as least as expensive than node failures (s1, s3, s5, s6 and s7), securing the network against single link failures caused the same cost as securing it against *all* single failures. Thus, for the corresponding instances, we tested the solutions obtained for link failures for feasibility

with respect to node failures and with respect to all single failures. It turned out that the solutions were feasible in all cases.

This observation together with what we found out about the maximal failure flow allows us two conclude that node failures are often dominated by link failures (at least in our test instances). In other words, it should often be sufficient to calculate a solution for securing a network against all single link failures, leading to reduced calculation times (due to a smaller restoration LP). There is no guarantee but odds are that the obtained solution is also feasible for all single failure states.

6.4 Effects of joint optimization and cost structure

In Section 6.3, we found out that for many instances, Path Restoration (with or without stub release) was not cheaper than Link Restoration and that the application of stub release led to lower cost only in one case. In some cases, capacity cost was the same for all scenarios. As these results were quite surprising and in contrast to the results of other authors (e.g., [PD98]), we tried to find out the reasons for these discrepancies.

There were especially two candidates:

1. We optimized NOS and spare capacity together. Many authors (including [PD98]) optimized NOS capacity first and spare capacity afterwards. In many publications, a shortest hop or shortest length routing is used for the NOS, followed by spare capacity optimization.

   While this approach is of course of interest to a network provider who wants to secure an existing NOS network against failures, it drastically reduces the number of possible solutions (especially with Link Restoration) and can thus be expected to have a negative impact on the overall network cost compared to joint NOS and spare capacity optimization.

2. We took the discrete cost structure into account which highly non-linear in most cases. Most authors assume capacities than can take any continuous or any integer value. In some contexts, this assumption is valid, but in other contexts (including ours) it is not. The fact that in our test instances, there was only little choice in the installable edge designs (often only two or three edge designs per edge) should have a significant effect on the optimal solution value.

6.4.1 Effects of joint optimization

In order to see how results change and in order to test the quality of this heuristic for the more difficult instances, we determined a shortest path NOS routing and minimized spare capacity cost afterwards based on this NOS routing. Each NOS demand was routed over exactly one shortest path with respect to either uniform edge weights (shortest hop path) or the length of an edge as its weight (shortest length path).

Table 6.2 shows the results of these tests compared to joint optimization of NOS and failure states. The first group of columns shows the values for joint optimization, whereas the second and third group of columns show the values for spare capacity assignment (SCA) after determination of a shortest hop path or shortest length path routing, respectively.

Some interesting facts can be observed:

- In several cases, Link Restoration has been found to be actually more expensive than Path Restoration without stub release, and the application of stub release actually had an effect on total capacity cost. Both was not the case in our test instances with joint optimization of NOS and spare capacity.

  The fact that the obtained cost values for Link Restoration were often higher than those obtained for Path Restoration suggests that fixing a NOS routing has a higher impact on Link Restoration. This is understandable since with Link Restoration, fixing an NOS path drastically reduces the number of possible restoration paths, in contrast to Path Restoration. This is in accordance with the results of [IMG98] who found out that Link Restoration benefits more of joint NOS and spare capacity optimization compared to successive optimization than Path Restoration.

- For s4, the optimal solution has been found for one scenario (node and link failures, shortest hop NOS routing, Path Restoration with stub release). The worst optimal value ratio of successive to joint optimization is 2.89 (for s1 with node failures only, shortest length NOS routing and all restoration mechanisms). In other words, the value obtained by fixing an NOS routing was nearly three times the optimal value with joint optimization!

- The average optimal value ratio of successive to joint optimization is 1.30 which a shortest hop NOS routing (1.13 without instance s1) and 1.35 for a shortest length NOS routing (1.15 without s1). Given that calculation times were all within a few seconds, fixing an shortest path routing for the NOS seems to be not too bad as a heuristic for difficult instances. However, this approach may sometimes lead to very bad solutions (as in s1 or s6) or may even cause the problem to get infeasible, as can be seen at the example of s3 with Link Restoration.

- With the exception of s4, the cost relation between node failures, link failures and all single failures is at least approximately the same with joint and successive optimization.

- It is not clear whether a shortest hop routing or a shortest length routing performs better; this heavily depends on the test instance and the failure scenario. Even within a test instance, often none of the two heuristics is clearly better.

6.4.2 Effects of discrete cost structure

We observed that on many edges, the installed capacity was much higher than required by the flow through these edges. We supposed that this effect was due to the little choice of installable edge designs. To see if this holds and to investigate how resistant the obtained solution values and the observed results were w.r.t. changing demands, we performed a scaling test. For each of our small test instances, all demands values were scaled by the same factor $C \in \{0.1, 0.2, 0.5, 1, 2, 5, 10\}$. All other data was left unchanged.

Table 6.3 shows the cost of securing a network against all single component failures; the first line contains the scaling factor. As in Table 6.2, the optimal NOS value with $C = 1$ was scaled to 100. A hyphen (-) means that the corresponding problem had no solution since the installable capacities were insufficient to accommodate the increased demand values.
The results indicate that the discrete cost structure and the little choice of edge designs has in fact a major impact on the costs of the network:

No.	NOS	model	0.1	0.2	0.5	1	2	5	10
s1	100	LR	132	132	132	132	132	142	192
		PR-sr	132	132	132	132	132	140	186
		PR+sr	132	132	132	132	132	140	182
s2	100	LR	100	100	100	183	232	-	-
		PR-sr	100	100	100	183	232	-	-
		PR+sr	100	100	100	183	232	-	-
s3	100	LR	81	92	121	162	-	-	-
		PR-sr	81	92	114	177	-	-	-
		PR+sr	81	92	114	162	-	-	-
s4	100	LR	24	42	113	246	-	-	-
		PR-sr	24	41	112	246	-	-	-
		PR+sr	24	41	112	246	-	-	-
s5	100	LR	92	97	113	113	113	181	348
		PR-sr	92	97	113	113	113	181	348
		PR+sr	92	97	113	113	113	181	348
s6	100	LR	93	98	114	114	114	183	353
		PR-sr	93	98	114	114	114	183	353
		PR+sr	93	98	114	114	114	140	353
s7	100	LR	160	160	162	171	205	-	-
		PR-sr	160	160	162	171	205	-	-
		PR+sr	160	160	162	171	205	-	-

Table 6.3: Optimal values with scaled demands

- For the instance s1 (10/25/29), we obtained the same optimal value for scaling factors between 0.1 and 2. Otherwise stated, a multiplication of the demand values by 200 did not change total capacity cost! A multiplication of the demand values by 1000 causes cost to raise only by 37.8% (with Path Restoration). This is probably due to the cost structure: in this instance, the demand values (scaled with $C = 1$) are very small compared to the installable capacities (the smallest installable capacity is 50 times the maximal demand value). Only two edge designs are available for each edge. This makes that often much more capacity has to be installed than actually needed by the routed flow.

- In instance s5 (18/21/62), the solution value is the same from $C = 0.5$ to $C = 2$, multiplication of the demand values by 500 not even doubles cost (from 92 to 181), and scaling the demand values by 1000 increases total capacity cost by only a factor of 3.8.

  This can be explained by the cost structure as well: recall from Section 6.2.2 that although eight edge designs are available for each edge, it is often cheaper to install capacity 16 than to install capacity 3. In other words, as soon as the flow on such an edge exceeds 3, capacity 16 is installed. Apparently, the resulting capacity installation suffices to accommodate much more demand as well.

  The same holds for the instance s6 (18/27/62), which is identical with s5 except for six additional edges. It seems that the additional edges cannot compensate the effect of

the cost structure.

- Similar observations can be made for the instances s2, s3 and s7.

An exception is s4 (15/22/105) where cost raises nearly linearly with demand. In this instance, demands (in the unscaled version) are small compared to installable capacities (as in instance s1), but km-cost is nearly linear with capacity. Seven edge designs are available on each edge (which is quite a lot) and there is no node cost. Apparently, the mostly linear structure of capacity cost causes a behaviour that is more similar to continuous cost.

It may be interesting to remark that s4 is the instance where the best values were obtained with successive NOS and spare capacity optimization, and where even the optimal cost for implementing restoration was extraordinarily high compared to the cost without restoration (see Section 6.4.1).

Another interesting observation is that independently of the demand scaling factor, we often obtained the same result for every restoration mechanism. However, there are demand scaling factors such that Link Restoration is actually more expensive than Path Restoration without stub release (for instance, s3 and s4 with $C = 0.5$ and s1 with $C = 5$ or $C = 10$). In addition, there are two more instances where the application of stub release has an effect on capacity cost (s6 with $C = 5$ and s1 with $C = 10$). Observe however that these two scenarios and s3 with $C = 1$ are the only cases where stub release helped reducing capacity cost.

The observed resistance of the optimal solution values w.r.t. changing demand values could explain why we nearly always got the same cost with different settings for some test instances.

These results have two consequences which are of major practical interest:

1. As soon as there is only little choice in the installable capacities on an edge or as soon as the cost structure exhibits significant non-linear behaviour (as for s5, s6 and s7), assuming continuous capacities can be largely misleading. These conditions are often met in practice (recall that all our test instances are based on real world data).

2. When taking the discrete cost structure into account, changing demand values even by a large amount often does not affect optimality of the given solution. This is of practical interest since demand values can often only be roughly estimated.

6.5 Handling difficult instances

This section describes the computational results with those instances which for most scenarios could not be solved to optimality in reasonable time and where we had difficulties to find good solutions at all.

6.5.1 Obtaining a standard setting

First, in order to reduce the number of tests, we used the small test instances to determine a standard setting for some parameters:

1. the intensity of the initial path calculation (minimal, medium or heavy),

2. the edge weights in the initial path calculation (uniform or length based),

3. the node probability calculation with Link Restoration (uniform, length based, degree based or all three together),

4. the value of k in the k-shortest paths algorithm.

To this end, for each of the small test instances, each failure scenario (node failures, link failures or both) was tested with every restoration mechanism (Link Restoration and Path Restoration with or without stub release) and with different initial path calculation intensities and edge weights. With Link Restoration and node failures, every such test was combined with each of the four variants of node probability calculation. For these tests, we fixed the value of k in the k-shortest paths algorithm to 30.

With minimal initial path calculation (IPC for short), only one path was initially calculated for every demand. For medium and heavy IPC, we grouped the demands into three classes based on their value. Then, for some suitable values $l, m, h \in \mathbb{N}$, we applied the k-shortest paths algorithm to calculate l shortest paths for the demands with low value, m paths for the demands with medium value and h paths for the demands with high value. The values (l, m, h) were set to $(3, 5, 8)$ for medium IPC and to $(7, 10, 15)$ for heavy IPC.

It turned out that the choice of node probability calculation, edge weights or intensity of initial path calculation had no effect on the best obtained solution value, so we suppose that these values are indeed optimal even in the cases where pricing was not exact.

The calculation times reached from some seconds to one hour. The proportion of calculation times between single node failures, single edge failures and all single failures was often in the order of 1:5:10 but varied widely. The fact that consideration of node failures took the least time is probably due to the size of the restoration LPs: with node failures only, there is only a small number of operating states, which drastically reduces the number of inequalities.

Within one scenario, none of the settings was clearly superior to another one with respect to solution time. For nearly every setting, there were a test instance and a scenario where this setting performed worse than all others. The only clear result was that applying all three variants of node probability calculation with Link Restoration took about twice the time than applying only one of them without yielding better results. However, there were very slight advantages for the combination

minimal initial path calculation,
length based edge weights,
degree based node probability calculation,

so we used this as the standard setting in further tests.

For the value of k, we tested every value in $\{1, 5, 10, 20, 30\}$ with the standard setting for Link Restoration and for Path Restoration with stub release (these were the cases where the k-shortest paths algorithm was applied). Also the value of k had no effect on the best obtained solution value.

Usually, there was nearly no time difference between $k = 10$ and $k = 30$ (interesting enough, there were some rare cases where raising the value of k led to shorter calculation times). The time difference between $k = 1$ and $k = 10$ was in most cases negligible for Link Restoration; for Path Restoration with stub release, setting $k = 10$ took about 10-50% more time than setting $k = 1$. In fact, this time difference was not spent mainly in the k-shortest paths algorithm but in the postprocessing of the found paths: calculating the correct length of the paths and finding the shortest one. Finally, given that $k = 1$ took the least time in most

cases without degradation of the results, we used $k = 1$ as standard value in the following tests.

6.5.2 Trying to obtain optimal values

We tried to solve the instances d1 to d7 to optimality with each of our scenarios using the standard setting and imposing a time limit of four hours.

The first columns (*exact*) of Table 6.4 show the results obtained for all single failures; the best obtained lower bound (*lb*) was scaled to 100. The best obtained solution value for each scenario is listed in the column *ub*. The solution times are given in the format *hours:minutes*.

No.	model	lb	exact		minimal, 50/∞		heavy, 100/200	
			ub	time	ub	time	ub	time
d1	LR	100	**184**	4:00	196	0:37	219	0:49
	PR-sr	100	**207**	4:00	219	4:00	282	4:00
	PR+sr	100	218	4:00	**217**	4:00	266	4:00
d2	LR	100	**146**	4:00	151	4:00	161	4:00
	PR-sr	100	**147**	4:00	**147**	4:00	163	4:00
	PR+sr	100	**149**	4:00	152	4:00	169	4:00
d3	LR	100	**107**	4:00	114	4:00	125	4:00
	PR-sr	100	**115**	4:00	123	4:00	142	4:00
	PR+sr	100	**107**	4:00	121	1:03	140	4:00
d4	LR	100	**100**	1:42	104	1:07	105	0:31
	PR-sr	100	**107**	4:00	109	4:00	114	1:42
	PR+sr	100	**109**	4:00	110	1:43	113	1:57
d5	LR	100	**112**	4:00	113	4:00	121	4:00
	PR-sr	100	**126**	4:00	**126**	4:00	132	4:00
	PR+sr	100	**121**	4:00	128	4:00	132	4:00
d6	LR	100	**154**	4:00	155	4:00	170	4:00
	PR-sr	100	237	4:00	222	4:00	**188**	4:00
	PR+sr	100	252	4:00	**172**	4:00	190	4:00
d7	LR	100	**115**	4:00	118	4:00	120	4:00
	PR-sr	100	**128**	4:00	**128**	4:00	129	4:00
	PR+sr	100	126	4:00	128	4:00	**124**	4:00

Table 6.4: Values for dense instances with different pricing strategies

As can be seen from the table, only one instance could be solved to optimality within the given time limit: d4 (13/28/45) with Link Restoration. Apart from this instance, the obtained gaps (calculated as $(ub - lb)/lb$) reach from 7% (d4, PR-sr) to 152% (d6, PR+sr).

At this point, it is interesting to know that in the small instances where we know the optimal solution values, the lower bound obtained at the Branch and Bound root node was always at least half of the optimal value. Obviously, if this holds for the lower bound of the root node, it holds as well for the best obtained lower bound. We have thus reason to assume that the obtained solutions are not as bad as they sometimes seem to be (for instance, which d1 and d6). In fact, as will be described in Section 6.5.3, we sometimes obtained much better

solutions by means of heuristics.

In all test instances, the best lower bound was obtained with Link Restoration; in five out of seven instances (all except d2 and d3), also the best obtained solution value was lowest with Link Restoration. Together, this makes that the gap was lowest with Link Restoration in all cases.

For Path Restoration, the application of stub release or not caused only little difference w.r.t. the quality of the obtained bounds; in particular, none of these settings was clearly easier to handle than the other ones in all cases.

What cannot be seen from the table (since we rarely had an optimal NOS value as reference value for scaling) is the fact that the obtained lower bounds and solution values for different restoration models were usually very close to each other, within a difference of a few percent. Thus, this observation from the small networks seems to hold as well for more dense networks.

The only exception of this rule was instance d6, where the best obtained solution values for Path Restoration were 28% higher (without stub release) and 26% higher (with stub release) than those obtained with Link Restoration. Note that in this case (as in some others), the best known upper bound for Path Restoration with stub release can be improved to the value defined by the Link Restoration solution since every solution for Link Restoration also defines a solution for Path Restoration with stub release.

Our results for node failures only and link failures only are not given here in detail since they are very similar to the results for all single failures.

6.5.3 Comparison of heuristics

As the results with exact calculation left room for improvement, we tried several heuristics:

- PRICING_STOP: Since most of the calculation time was spent in the pricing process (especially in NOS path pricing), we simply stopped pricing after a certain number of iterations in the hope that at this point, enough suitable paths would have been priced as to allow good solutions. This was combined with different intensities of initial path calculation (IPC).

  For notational convenience, we denote by m/n the setting where NOS path pricing and failure path pricing were stopped after m and n pricing iterations, respectively. The tested settings were

  1. minimal IPC, ∞/∞ (no pricing stop; this corresponds to exact calculation),
  2. minimal IPC, $50/\infty$
  3. heavy IPC, 100/200.

- SUCCESSIVE_OPTIMIZATION: We routed each NOS demand over exactly one shortest hop path or shortest length path and minimized spare capacity based on this NOS routing.

- EDGE_DELETION: In order to decrease density of the networks, we removed some edges in a preprocessing step which we considered unlikely to be needed for good solutions. Note that every solution in the reduced network can easily be extended to a solution in the original network by setting the capacity of all removed edges to 0.

  The removed edges were especially long ones whose end nodes had a high degree or were far from the geographic center of the network. The latter criterion was based on

the heuristic argument that nodes which are "central" in some way need more incident edges to allow good solutions than nodes which are "far away". Note that this use of information about the geographic position of the nodes in addition to topological information only makes sense for real world based networks.

We implemented a two-step greedy heuristic with parameters m and n to remove edges as follows:

1. Consider the non-bridge edges in decreasing order by length. As long as there is an edge whose end nodes are both of degree greater than m, remove it unless the removal would create an articulation node. At the end of this step, the network should have an average degree of slightly above m.
2. Again, consider the remaining non-bridge edges decreasingly sorted by length. As long as there is an edge whose end nodes are both outer nodes of degree greater than n, remove the edge unless this would create an articulation node. A node was declared to be an *outer* node if its distance from the geographic center exceeds the average distance from the center taken over all nodes.

Our values for (m/n) were $(3/3), (4/2)$ and $(4/3)$. Observe that due to the greedy nature of the heuristic, an (m/n) network need not necessarily be a subgraph of the corresponding $(m + 1/n)$ network nor of the corresponding $(m/n + 1)$ network. In particular, this implies that the optimal solution w.r.t. a (4/3)-graph is not always better than the solution w.r.t. the corresponding (4/2)-graph.

In order to reduce the number of tests, we restricted ourselves to the case where all single failures are considered. This is the most difficult case, and every solution obtained for this setting is also feasible for node failures only or link failures only. For every heuristic, we imposed a time limit of four hours, as for exact calculation.

PRICING_STOP

Table 6.4 shows our results with a pricing stop after a certain number of iterations: minimal IPC without pricing limit, minimal IPC with NOS path pricing stopped after 50 pricing iterations and heavy IPC with NOS and failure path pricing stopped after 100 and 200 pricing iterations, respectively.

As can be seen from the table, we rarely obtained better solutions by means of these heuristics than with exact calculations within four hours. Only for test instance d6 with Path Restoration, the solution values could significantly be improved. In most scenarios, the time limit has been reached.

In some cases, solutions of similar quality as with exact calculation were found by the heuristics in significantly less time than four hours (for instance, d1 with Link Restoration). But in all these cases, the same value had been found in even less time without a pricing stop.

Summarizing these observations, it seems that stopping pricing is not a good heuristic for our needs, at least not with the tested parameters. Further tests would be necessary to see if the pricing stop should be done earlier (to avoid time consuming pricing) or later (to ensure that enough "good" paths are contained in the LP) and how to combine the parameters with different settings for initial path calculation.

SUCCESSIVE_OPTIMIZATION

Our results with successive NOS and spare capacity optimization are shown in Table 6.5. Again, the best known lower bound has been set to 100 and the best obtained upper bound with joint NOS and spare capacity minimization and without pricing stop (from Table 6.4) is given together with the lower bound as a reference value (column *exact*); again, the best obtained solution value for each scenario has been highlighted. The calculation times are shown in the format *hours:minutes:seconds*.

No.	model	exact		shortest hop		shortest length	
		lb	ub	ub	time	ub	time
d1	LR	100	**184**	590	0:00:03	590	0:00:03
	PR-sr	100	**207**	659	0:00:11	659	0:00:10
	PR+sr	100	**218**	657	0:00:10	657	0:00:10
d2	LR	100	**146**	180	0:00:02	180	0:00:02
	PR-sr	100	**147**	186	0:00:05	186	0:00:05
	PR+sr	100	**149**	188	0:00:05	188	0:00:05
d3	LR	100	**107**	285	0:00:05	-	-
	PR-sr	100	**115**	292	0:00:18	-	-
	PR+sr	100	**107**	291	0:00:18	-	-
d4	LR	100	**100**	117	0:00:04	112	0:00:03
	PR-sr	100	**107**	125	0:00:29	119	0:00:13
	PR+sr	100	**109**	126	0:00:25	121	0:00:12
d5	LR	100	**112**	148	0:00:05	148	0:00:07
	PR-sr	100	**126**	163	0:00:24	161	0:00:14
	PR+sr	100	**121**	165	0:00:15	159	0:00:12
d6	LR	100	**154**	186	0:00:06	181	0:00:06
	PR-sr	100	237	201	0:00:21	**197**	0:00:18
	PR+sr	100	252	200	0:00:21	**195**	0:00:18
d7	LR	100	**115**	132	0:00:08	132	0:01:17
	PR-sr	100	**128**	136	0:00:23	136	0:01:01
	PR+sr	100	**126**	135	0:00:13	136	0:00:50

Table 6.5: Joint and successive optimization of difficult instances

The first remarkable result than can be seen from Table 6.5 is that the calculation times are extremely short compared to exact calculation or the pricing stop heuristics. The longest needed time is slightly more than one minute (for d7), and most times are far below 30 seconds.

The second interesting result is that the quality of the solutions varies widely, with gaps ranging from 12% (d4 with Link Restoration) to 559% (d1 with Path Restoration). For one instance (d6 with Path Restoration), significantly better solutions than previously known have been found by fixing the NOS routing. However, this is the only instance where the upper bound could be improved, and the quality of these solutions is still poor (at least 95% gap).

Although on the average, the obtained results with a shortest length routing were slightly better than those obtained with a shortest hop routing, the shortest length routing version

did not even allow a solution for the instance d3. In other words, it depends on the test instance and the cost structure (more specifically, if capacity cost is dominated by length based or fixed edge cost) which version performs better, as has already been observed with the small instances in Section 6.4.1.

Summarizing this, we observed that this heuristic performs rather poor with respect to the quality of the solutions but very good with respect to solution times. In other words, it can well be applied as a starting heuristic to find some solutions in short time for instances where it is otherwise difficult to obtain solutions at all, but not for finding near-optimal solutions.

It is perhaps interesting to recall that the network of s1 is a subnetwork of d1 and that with successive NOS and spare capacity optimization, the quality of the obtained solution was very bad for both networks (221% gap for s1, 559% for d1), much worse than for the other networks. We do not know why successive optimization performs so poorly on these particular networks; however, we see that this effect can occur on sparse networks as well as on dense ones (the d1 network is complete).

The very short calculation times with successive optimization suggest that with joint optimization, the NOS routing is very often changed in the course of the algorithm in order to accommodate some additional failure flow without augmenting the currently installed capacities. This could explain to the long calculation times observed with joint optimization.

In order to improve the quality of the obtained solutions with successive optimizations, it could thus be interesting to calculate a routing for the NOS on more than one path per demand and to optimize spare capacity afterwards.

EDGE_DELETION

Table 6.6 shows the results obtained by the EDGE_DELETION heuristic. The best known lower bound was set to 100; the best obtained upper bound from exact calculation (taken from Table 6.4) is shown as a reference value. Both values were in most cases obtained after four hours of calculation time. As in the previous tables, the best obtained solution value in each row has been highlighted; a hyphen (-) indicates that there was no solution for the corresponding network and scenario.

A surprising result from these test is that in the sparse networks (3/3), often better results were obtained than in the denser networks (4/3). With the (4/2) networks, the results were rather poor. For two networks (d1 and d6), significantly better solutions could be found with the (3/3) heuristic than with exact calculation; for one network (d4), results of similar quality as with the exact approach could be obtained.

In many cases (including those where a better solution than previously known has been found), the calculation times were small; the solution process often took only some seconds or minutes. Often, the indicated solution has been found long before the given time; in these cases, a lot of time has been spent on raising the lower bound to prove that the obtained solution is indeed optimal w.r.t. the reduced network. In other cases, the time limit of four hours has been reached.

As can be seen from instance d7, the difference between a network which can be solved to optimality within some seconds and a network where four hours are not enough can be small: the (4/3) network, where the time limit has been reached, contains only eight edges more than the corresponding (4/2) network which could be solved to optimality in less than 20 seconds. Apparently, the removed edges in the (4/2) network dramatically reduced the number of possible solutions, as is also indicated by the solution values (260 vs. 138 with Link

No.	model	exact		3/3		4/2		4/3	
		lb	ub	ub	time	ub	time	ub	time
d1	LR	100	184	**128**	0:00:02	170	0:00:06	179	0:00:08
	PR-sr	100	207	**143**	0:00:12	190	0:00:11	200	0:00:16
	PR+sr	100	218	**142**	0:00:13	189	0:00:12	200	0:00:18
d2	LR	100	**146**	189	0:01:25	-	-	195	0:02:03
	PR-sr	100	**147**	194	0:16:02	-	-	201	0:19:08
	PR+sr	100	**149**	197	0:19:45	-	-	203	0:21:17
d3	LR	100	**107**	-	-	145	0:00:23	136	0:01:56
	PR-sr	100	**115**	-	-	144	0:01:02	136	0:15:46
	PR+sr	100	**107**	-	-	144	0:01:26	136	0:25:10
d4	LR	100	**100**	**100**	0:00:28	-	-	-	-
	PR-sr	100	**107**	**107**	0:04:06	-	-	-	-
	PR+sr	100	109	**108**	0:02:52	-	-	-	-
d5	LR	100	**112**	130	0:00:36	177	0:02:41	116	0:10:55
	PR-sr	100	126	132	0:08:08	128	0:16:34	**123**	1:15:39
	PR+sr	100	**121**	134	0:12:59	130	0:19:46	124	1:45:44
d6	LR	100	**154**	170	0:03:07	188	1:27:49	180	4:00:00
	PR-sr	100	237	**184**	0:54:28	208	4:00:00	205	4:00:00
	PR+sr	100	252	**183**	1:15:07	207	4:00:00	211	4:00:00
d7	LR	100	**115**	130	3:12:16	260	0:00:04	138	4:00:00
	PR-sr	100	**128**	134	4:00:00	278	0:00:18	149	4:00:00
	PR+sr	100	**126**	134	4:00:00	278	0:00:15	148	4:00:00

Table 6.6: Removing links from dense networks

Restoration).

Contrary to what we had expected, we could not find any obvious relation between the density of the original graph and the performance of this heuristic.

6.6 Summary and outlook

Summarizing the results of this chapter, we obtain the following observations and conclusions from the computational tests:

Capacity cost structure has a major impact on the stability of solutions w.r.t. changing parameters. In particular, this impact is often stronger than the impact of the choice of a restoration model or variations of the demand values. This holds for joint as well as for successive NOS and spare capacity optimization.

The cost increase caused by successive NOS and spare capacity optimization compared to joint optimization can be quite important (it can cause capacity cost to be more than tripled, as seen in Sections 6.4 and 6.5.3). As a heuristic for difficult instances, successive optimization performs very well w.r.t. solution time but rather poorly w.r.t. solution quality. As seen above, a good compromise could be to fix a NOS routing on more than one path per demand.

A pricing stop in the course of the algorithm did not yield good results, neither with

respect to solution time nor with respect to quality of the obtained solutions.

Edge deletion can be a quite good heuristic with respect to both time and solution quality if a suitable subset of edges is chosen. On the other hand, deleting the wrong edges can lead to both bad solutions and long calculation times or even to infeasible problems, as seen in the previous section. It is sometimes hard to determine a suitable subset of edges to be deleted.

There remains an interesting question: what makes an instance difficult to solve? Regarding the characteristics of the investigated networks in Table 6.1 together with the obtained results, we see that the most important factor is rather density than size or cost structure: the all of the instances d1 to d7 have a higher average nodal degree than instances s1 to s7 (hence the names: 's' for *sparse* and 'd' for *dense*), and the difficult instances which could be solved best are d3, d4, d5 and d7, three of which are rather sparse compared to the other networks.

The fact that dense networks are, in general, more difficult to solve is probably due to the high number of eligible routing paths in dense networks. This makes that the NOS routing can very often be changed in the course of the algorithm and that there are many possible path variables to be priced out.

Among networks of similar size and density, the number of demands has a major impact on the overall solution time. In fact, the small instances which took the longest time to be solved were s4 and s7, both with many demands. Especially with Path Restoration, the number of demands directly influences the number of failure commodities and thus the number of constraints.

We observed that overall solution time was often determined by a very slow improvement of the lower bound. Often, during a long time, metric inequalities were the only separated constraints, and much time was spent on proving the optimality of a given solution by raising the lower bound. Hence, it could be advantageous to apply further types of cutting planes in order to achieve a faster improvement of the lower bounds.

Furthermore, alternatives should be found to the time consuming feasibility test where most of the solution time has been spent. These alternatives could either be exact to preserve optimality, or they could be heuristic in order to speed up the solution process while giving up guaranteed optimality.

Chapter 7

Conclusion

In this thesis, a Mixed Integer Programming model for the network restoration problem has been proposed that applies to an arbitrary restoration mechanism in conjunction with an arbitrary set of operating states. In particular, this includes the commonly used restoration techniques Link Restoration and Path Restoration, but also Reservation, which is more of theoretical interest. The model integrates restoration and hardware requirements and takes into account that capacity cost structure is often highly non-linear in practice. It allows NOS and spare capacity to be optimized together as well as in separate steps.

After presentation of the model, the theoretical and implementational aspects of the employed algorithmic procedure have been presented. It turned out that the pricing problem with this model is polynomially solvable with Link Restoration or with Path Restoration without stub release (release of no longer used capacity in failure states), but $\mathcal{NP}$-hard when Path Restoration with stub release is applied. Several heuristics have been discussed how to solve the problem to optimality with high probability nevertheless.

The described algorithm has been implemented in a Branch and Cut framework and tested against 14 real world test instances. We compared the optimal values of small instances w.r.t. all single edge failures, all single node failures or all single failures, each of these settings combined with Link Restoration and Path Restoration with or without stub release.

We found that the particular choice of restoration mechanism had only little influence on capacity cost in our test instances. Since this result was in contrast to the results of other authors, we further investigated the effects of a discrete cost structure and of joint working and spare capacity optimization.

It turned out that the discrete, highly non-linear cost structure of some instances had a major effect on total network cost; such a cost structure often caused capacity cost to be very stable w.r.t. a change of restoration mechanism or changes of demand values even by a large amount. Optimizing spare capacity based on a given NOS shortest path routing can be substantially more expensive than jointly optimizing NOS and spare capacity.

On some instances which were difficult to solve, we compared several heuristics: deleting some edges from the network, optimizing spare capacity based on a fixed NOS shortest path routing and stopping pricing after a certain number of pricing iterations. It turned out that minimizing spare capacity cost based on a given NOS routing was excellent w.r.t. solution time but rather poor w.r.t. solution quality. A pricing stop in the course of the algorithm performed poorly with regard to both aspects, and the performance of the edge deletion heavily depends on the choice of deleted edges.

Further research should be done to reduce solution times for big or dense instances. This concerns the use of additional cutting planes to improve the lower bound as well as testing new heuristics to improve the upper bound; some possible heuristic improvements have been proposed in this work in context with our computational tests.

Appendix A

Appendix

A.1 Table of symbols

$\mathbb{N}$	the natural numbers $0, 1, 2, \ldots$
$\mathbb{Z}$	the integer numbers $\ldots, -2, -1, 0, 1, 2, \ldots$
$\mathbb{R}$	the real numbers
$\mathbb{R}_+$	the nonnegative real numbers
$A \subset B$	A is a proper subset of B (in particular, $A \neq B$)
V	the set of nodes of the network
E	the set of edges of the network
S	the set of operating state
$V^s \subseteq V$	the set of usable nodes in operating state $s \in S$
$E^s \subseteq E$	the set of usable edges in operating state $s \in S$
D	the set of NOS demands
C^s	the set of failure commodities in $s \in S$
ρ^s_{uv}	restoration level $\in [0, 1]$ for demand uv in operating state s
$\{u, v\}$	an unordered pair of nodes, an undirected edge between nodes u and v or the set of all undirected edges between u and v, according to the context
(u, v)	the same for directed edges or ordered pairs of nodes
$\delta(v)$	the set of incident edges of $v \in V$
$\delta^+(v)$	the set of outgoing edges of a node $v \in V$ in a digraph
$\delta^-(v)$	the set of incoming edges of a node $v \in V$ in a digraph
$\delta(W)$	the set of edges with one end node in $W \subset V$ and the other in $V \setminus W$.
$\Gamma(v)$	the set of adjacent nodes to v
$\Gamma^+(v)$	the set of target nodes of edges in $\delta^+(v)$
$\Gamma^-(v)$	the set of source nodes of edges in $\delta^-(v)$
$\mathcal{P}$	the set of all paths in the network
$\mathcal{P}_{uv}$	the set of all uv-paths
$\mathcal{P}^s$	the set of valid paths in operating state $s \in S$
$\mathcal{P}^s_{uv}$	$= \mathcal{P}^s \cap \mathcal{P}_{uv}$, the set of valid uv-paths in $s \in S$
$\mathcal{IP}^s_c$	the set of paths responsible for failure commodity $c \in C^s$
ℓ_{uv}	hop limit for all NOS paths $P \in \mathcal{P}^0_{uv}$
$f_{uv}(P)$	NOS flow on path $P \in \mathcal{P}^0_{uv}$
$f^s_c(P)$	failure flow on path $P \in \mathcal{P}^s_c$

List of Tables

List of Figures

Bibliography

[AGW96] D. Alevras, M. Grötschel and R. Wessäly: *A network dimensioning tool (1996).* ZIB Preprint SC-96-49, available at `www.zib.de`

[AGW97] D. Alevras, M. Grötschel and R. Wessäly: *Cost Efficient Network Synthesis from leased lines.* ZIB Preprint SC-97-22 (`www.zib.de`), Annals of OR, 76:1-20

[Ata00] A. Atamtürk: *On capacitated network design cut-set polyhedra (Dec 2000).* Mathematical Programming 92, 425-437, 2002. Available at `http://ieor.berkeley.edu/~atamturk`.

[AMMP90] J.A. de Azevedo, J. Madeira, E. Martins and F. Pires: *A shortest paths ranking algorithm.* Proceedings of the Annual Conference Associazione Italiana di Ricerca Operativa, Sorrento (Italy), October 90 (AIRO'90), 1001-1011.

[BMSW98] A. Balakrishnan, T.L. Magnanti, J. Sokol and Y. Wang: *Modeling and solving the Single Facility Line Restoration Problem.* Technical report (1998), MIT, Operations Research Center

[BMSW00] A. Balakrishnan, T.L. Magnanti, J. Sokol and Y. Wang: *Telecommunication Link Restoration Planning with Multiple Facility Types (December 2000).* Annals of Operations Research, 106:127-154

[BM97] D. Bienstock and G. Muratore: *Strong inequalities for capacitated SND problems.* Technical Report, Columbia University, 1997.

[Bley03] *Personal communication with Andreas Bley, January 2003.*

[CGS98] S. Chopra, I. Gilboa and S. Sastry *Source sink flows with capacity installation in batches.* DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science, 85

[CMS93] C.E. Chow, S. McCaughey and S. Syed: *Rapid Restoration of Active Communication Truncs.* UCCS EAS Technical Report 92-18. Proceedings of the 2nd IEEE Network Management and Control Workshop, Westchester, New York, Sep 21-23, 1993.

[CG01] M. Clouqueur and W.D. Grover: *Dual failure availability analysis of span-restorable mesh networks.* IEEE JSAC Special Issue on Recent Advances in Fundamentals of Network Management, vol. 20, no. 4, May 2002, pp. 810-822. Available at `http://www.ee.ualberta.ca/~grover/wdgjournals.html`.

[DS93] G. Dahl and M. Stoer: *A Polyhedral Approach to Multicommodity Survivable Network Design.* Numerische Mathematik 68(1): 149-167, ZIB-Report SC-93-09, available at `www.zib.de`

[DS98] G. Dahl and M. Stoer: *A Cutting Plane Algorithm for Multicommodity Survivable Network Design Problems.* INFORMS Journal on Computing, 10(1), p. 1-11, 1998

[DG01a] J. Doucette and W.D. Grover: *Comparison of Mesh Protection and Restoration Schemes and the Dependency on Graph Connectivity.* 3rd International Workshop on Design of Reliable Communication Networks (DRCN 2001), Budapest, Hungary, pp. 121-128, October 2001

[DG01b] J. Doucette and W.D. Grover: *Increasing the Efficiency of Span-Restorable Mesh Networks on Low-Connectivity Graphs.* 3rd International Workshop on Design of Reliable Communication Networks (DRCN 2001), Budapest, Hungary, pp. 99-106, October 2001.

[GJ79] M. Garey and D. Johnson: *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness* W.H. Freeman and Company, San Francisco, 1979

[GMS95] M. Grötschel, C.L.Monma, M.Stoer: *Design of Survivable Networks.* Handbook in Operations Research and Management Science, Volume on "Networks", Chapter on "Design of Survivable Networks", North Holland 1995

[Hei02] Heise Verlag: *Tausende Spanier nach Sabotage-Aktion ohne Telefon.* `http://www.heise.de/newsticker/data/gr-15.06.02-003/`

[IMG98] Iraschko, McGregor, Grover: *Optimal capacity placement for path restoration in STM or ATM mesh survivable networks.* IEEE/ACM Transactions on Networking 6, No 3, pp. 325-336, 1998.

[Iri71] M. Iri: *On an extension of the maximum-flow minimum-cut theorem to multicommodity flows.* Journal of the Operations Research Society of Japan, 13:129-135, 1971.

[Krö03] A. Kröller: *Network optimization: Integration of hardware configuration and capacity dimensioning.* Diploma thesis, ZIB and TU Berlin, to appear in spring 2003

[LOVG99] A. Lisser, A. Ouorou, J.P. Vial and J. Gondzio: *Capacity planning under uncertain demand in Telecommunication Networks.* Logilab technical report, Department of Management Studies, Univ. of Geneva, Switzerland (October 1999).

[LM01] S. Lumetta and M. Médard: *Towards a deeper understanding of Link Restoration Algorithms for Mesh Networks.* Proc. of the 20th Annual Joint Conf. of the IEEE Computer and Communication Societies (INFOCOM 01), April 22-26 2001, pp. 367-375.

[MW97] T.L. Magnanti and Y. Wang: *Polyhedral Properties of the Network Restoration Problem – with the Convex Hull of a Special Case.* Working Paper OR 323-97, Operations Research Center, MIT, 1997.

[Min81] M. Minoux: *Optimum Synthesis of a Network with non-simultaneous Multicommodity Flow Requirements.* Annals of Discrete Mathematics, 11:269-277

[OK71] K. Onaga and O. Kakusho: *On feasibility conditions of multicommodity flows in networks.* Transactions on Circuit theory, CT-18(4):425-429, 1971.

[PD98] F. Poppe and P. Demeester: *Economic Allocation of Spare Capacity in Mesh-Restorable Networks: Models and Algorithms.* Proc. of the 6th Int. Conf. on Telecommunication Systems, Modeling and Analysis, Nashville, March 5-8, 1998, pp. 77-86

[RA00] M. Riis and K.I. Andersen (Department of OR, University of Aarhus): *Capacitated network design with uncertain demand.* Working Paper 2000/5, ISSN 1398-8964

[SNH90] H. Sakauchi, Y.Nishimura and S. Hasegawa: *A self-healing network with an economical spare-channel assignment.* IEEE Global Telecommunication Conference (Globecom) 1990, p. 438-443

[SWSLGF98] P. Soriano, C. Wynants, R. Séguin, M. Labbé, M. Gendreau, B. Fortz: *Design and Dimensioning of Survivable SDH/SONET Networks.* Telecommunications Network Planning, B. Sans and P. Soriano (ed.), pp. 147-167, Kluwer, Norwell, MA, 1998.

[Wang01] P. Wang: *Integrierte Planung ausfallsicherer Telekommunikationsnetze mit Link Restoration.* Diploma thesis, 2001, ZIB and TU Berlin.

[Wess00] R. Wessäly: *DImensioning Survivable Capacitated NETworks.* PhD Thesis, 2000, ZIB and TU Berlin.

[Wol90] L.A. Wolsey: *Valid inequalities for 0/1 knapsacks and MIPs with Generalised Upper Bound constraints.* Discrete Applied Mathematics 29 (1990), pp. 251-261

[XM97] Y. Xiong, L. Mason: *Restoration Strategies and Spare Capacity Requirements in Self-Healing ATM Networks.* Proceedings IEEE Infocom '97 pp. 353-360

[Xiong98] Y. Xiong: *Optimal Design of restorable ATM Mesh networks.* 4th IEEE ATM Workshop, Fairfax, Virginia, 26-29 May 1998

[Wu92] T.W. Wu: *Fiber Network Service Survivability.* Telecommunications library, Artech house

Index

Zusammenfassung

In dieser Diplomarbeit geht es um die Dimensionierung ausfallsicherer Telekommunikationsnetze. Das Ziel ist die Bestimmung einer kostenminimalen Kapazitätsbelegung auf den Kanten und Knoten eines Netzwerks, so daß gegebene Bedarfe auch dann erfüllt werden können, wenn eine oder mehrere Netzwerkkomponenten ausfallen. Dabei müssen zusätzliche Bedingungen berücksichtigt werden, wie z.B. Längenbeschränkungen der Routingpfade oder Hardwarebeschränkungen.

Wir haben ein mathematisches Modell mit einem ganzzahlig linearen Programm entwickelt, das sowohl Restorationbedingungen als auch Hardwarebeschränkungen berücksichtigt und von dem tatsächlich verwendeten Restoration-Mechanismus und den betrachteten Ausfallsituationen abstrahiert. Zusätzlich erlaubt unser Modell sowohl die *gleichzeitige* als auch die *sukzessive* Optimierung der NOS-Kapazität, die für den ausfallfreien Zustand bestimmt ist, und der Zusatzkapazität, die für das Umrouten von Fluß in Ausfallsituationen bestimmt ist. Unsere flexible Formulierung der Kapazitätskosten ermöglicht es, die Effekte diskreter, nichtlinearer Kostenstrukturen zu berücksichtigen, wie sie in der Praxis oft auftreten.

Unseres Wissens gibt es noch keine Veröffentlichung, die alle diese Aspekte berücksichtigt, geschweige denn in einem einzigen Modell vereinigt, obwohl sie von hoher praktischer Relevanz sind.

Wir haben unser Modell auf der Basis eines Branch and Cut-Algorithmus mit externem Zulässigkeitstest implementiert; der theoretische Hintergrund unseres algorithmischen Ansatzes wird ausführlich beschrieben. Im Zusammenhang damit wird die Komplexität des Pricing-Problems untersucht.

Es zeigte sich, daß die o.g. Abstraktionen sowohl unter theoretischen als auch unter Implementationsaspekten sinnvoll waren. Tatsächlich legen unsere Untersuchungen eine Unterscheidung von Restoration-Techniken in *lokale* und *globale* Konzepte nahe statt der gängigen Unterscheidung in Link Restoration, Path Restoration, Reservation und Mischformen dieser Mechanismen. Zusätzlich zum theoretischen Hintergrund des algorithmischen Ansatzes werden einige Implementationsdetails erläutert.

Wir haben unsere Implementation an 14 realen Beispielinstanzen getestet. was ausführlich beschrieben wird. Ein Teil der numerischen Resultate besteht aus einem Vergleich der minimalen Kosten eines Netzwerks mit unterschiedlichen Restoration-Mechanismen, wenn alle einzelnen Kanten- und/oder Knotenausfälle betrachtet werden. Zusätzlich werden die Effekte diskreter Kostenstrukturen näher untersucht, die in der Literatur nur selten berücksichtigt werden. Außerdem wird betrachtet, welchen Effekt auf die Kosten eine gleichzeitige Optimierung von NOS und Ausfallsituationen im Vergleich zu sukzessiver Optimierung hat. Im zweiten Teil der numerischen Resultate werden Heuristiken für das Netzwerkdimensionierungsproblem beschrieben und auf ihre Anwendbarkeit bezüglich Lösungsqualität und -zeit getestet.

Hiermit versichere ich an Eides statt die selbständige und eigenhändige Anfertigung dieser Diplomarbeit.

(Sebastian Orlowski)