NTNU
Det skapende universitet

MASTERKONTRAKT
- uttak av masteroppgave

1. Studentens personalia

Etternavn, fornavn **Braaten, Sindre Møgster**	Fødselsdato **29. jan 1990**
E-post **sindre_m_b@hotmail.com**	Telefon **90180876**

2. Studieopplysninger

Fakultet **Fakultet for samfunnsvitenskap og teknologiledelse**	
Institutt **Institutt for industriell økonomi og teknologiledelse**	
Studieprogram **Industriell økonomi og teknologiledelse**	Hovedprofil **Anvendt økonomi og optimering**

3. Masteroppgave

Oppstartsdato **15. jan 2014**	Innleveringsfrist **11. jun 2014**
Oppgavens (foreløpige) tittel **Service Brokering in Cloud Computing**	
Oppgavetekst/Problembeskrivelse The purpose of this work is to study and analyse efficient use of several aspects of cloud computing in the provisioning of software services to clients. You should consider both costs and quality of service metrics in your study. The following main points must be considered: 1. Describe this planning problem. 2. Formulate one or more models for the problem. 3. Implement some of the formulated model(s) using appropriate software. 4. Test the model(s) 5. Discuss the applicability of the model(s) in practical planning.	
Hovedveileder ved institutt **Professor Bjørn Nygreen**	Medveileder(e) ved institutt **PhD Candidate Anders Gullhav**
Ekstern bedrift/institusjon **Telenor Research, Trondheim**	Ekstern veileder ved bedrift/instutisjon **Research Scientist Andres Gonzalez**
Merknader **1 uke ekstra p.g.a påske.**	

SMB BN

4. Underskrift

Student: Jeg erklærer herved at jeg har satt meg inn i gjeldende bestemmelser for mastergradsstudiet og at jeg oppfyller kravene for adgang til å påbegynne oppgaven, herunder eventuelle praksiskrav.

Partene er gjort kjent med avtalens vilkår, samt kapitlene i studiehåndboken om generelle regler og aktuell studieplan for masterstudiet.

NTNU 15/1-2014

.....
Sted og dato

.....
Student

.....
Hovedveileder

Originalen lagres i NTNUs elektroniske arkiv. Kopi av avtalen sendes til instituttet og studenten.

NTNU
Det skapende universitet

MASTERKONTRAKT
- uttak av masteroppgave

1. Studentens personalia

Etternavn, fornavn **Holmen, Mari**	Fødselsdato **12. jul 1989**
E-post **mahol@stud.ntnu.no**	Telefon **98663524**

2. Studieopplysninger

Fakultet **Fakultet for samfunnsvitenskap og teknologiledelse**	
Institutt **Institutt for industriell økonomi og teknologiledelse**	
Studieprogram **Industriell økonomi og teknologiledelse**	Hovedprofil **Anvendt økonomi og optimering**

3. Masteroppgave

Oppstartsdato **15. jan 2014**	Innleveringsfrist **11. jun 2014**
Oppgavens (foreløpige) tittel **Service Brokering in Cloud Computing**	
Oppgavetekst/Problembeskrivelse The purpose of this work is to study and analyse efficient use of several aspects of cloud computing in the provisioning of software services to clients. You should consider both costs and quality of service metrics in your study. The following main points must be considered: 1. Describe this planning problem. 2. Formulate one or more models for the problem. 3. Implement some of the formulated model(s) using appropriate software. 4. Test the model(s) 5. Discuss the applicability of the model(s) in practical planning.	
Hovedveileder ved institutt **Professor Bjørn Nygreen**	Medveileder(e) ved institutt **PhD Candidate Anders Gullhav**
Ekstern bedrift/institusjon **Telenor Research, Trondheim**	Ekstern veileder ved bedrift/instutisjon **Research Scientist Andres Gonzalez**
Merknader **1 uke ekstra p.g.a påske.**	

4. Underskrift

Student: Jeg erklærer herved at jeg har satt meg inn i gjeldende bestemmelser for mastergradsstudiet og at jeg oppfyller kravene for adgang til å påbegynne oppgaven, herunder eventuelle praksiskrav.

Partene er gjort kjent med avtalens vilkår, samt kapitlene i studiehåndboken om generelle regler og aktuell studieplan for masterstudiet.

NTNU 15/1-2014

Sted og dato

Student

Hovedveileder

Originalen lagres i NTNUs elektroniske arkiv. Kopi av avtalen sendes til instituttet og studenten.

NTNU
Det skapende universitet

SAMARBEIDSKONTRAKT

1. Studenter i samarbeidsgruppen

Etternavn, fornavn **Braaten, Sindre Møgster**	Fødselsdato **29. jan 1990**
Etternavn, fornavn **Holmen, Mari**	Fødselsdato **12. jul 1989**

2. Hovedveileder

Etternavn, fornavn **Nygreen, Bjørn**	Institutt **Institutt for industriell økonomi og teknologiledelse**

3. Masteroppgave

Oppgavens (foreløpige) tittel **Service Brokering in Cloud Computing**

4. Bedømmelse

Kandidatene skal ha *individuell* bedømmelse ☐
Kandidatene skal ha *felles* bedømmelse ☒

NTNU 15/1-2014
Sted og dato

Sindre Møgster Braaten

Hovedveileder

Mari Holmen

Originalen oppbevares på instituttet.

Preface

This thesis is the result of our work on the master in the subject TIØ4905 Managerial Economics and Operations Research at the Department of Industrial Economics and Technology Management at NTNU.

In this work we have combined concepts from both the fields of computer science and operations research; the technological and economic specialisations of our Master of Science degrees.

We would like to thank our supervisor, Professor Bjørn Nygreen for helpful discussions, advice and support leading up to and throughout the work of this thesis, and PhD candidate Anders Nordby Gullhav for his insight in model implementations and guidance in the writing of this thesis. In addition we give thanks to Research Scientist Andres Gonzales for good support and insight into the real world setting of cloud computing, and to Astrid Undheim for inspiration and good help in the work leading up to this thesis.

Note to reader: Some of the contents and text presented in this thesis is either based on or derived directly from the specialisation project Braaten and Holmen (2013) written by the authors autumn/winter 2013. The sections that are the derived from the project are indicated in each chapter's introduction.

Abstract

With the expanding cloud computing market new business models take form, and the focus in the market is on differentiation and adapting to the escalating demands of the customers. To satisfy the business segment of the market the offering of some quality of service (QoS) measures and guarantees is paramount. The next generation of brokers will be the QoS-aware brokers with cloud connectivity.

This thesis regards the decisions of a combined broker and carrier in a cloud computing ecosystem. Such a broker has to make decisions on the composition of his customer portfolio, where to deploy services, and how to provide the network connectivity to these services, while ensuring compliance with the customers' requested QoS.

In this thesis, three mixed integer programming (MIP) models, a link-flow model (LFM), path-flow model (PFM) and a mapping model, are presented; considering latency, availability, routing of the traffic, provisioning of backup paths, and the provider placement of each service; in order to select a customer portfolio and maximise profits. The PFM and the mapping model use pre-generation of paths and mappings, respectively. Three heuristic column generation methods for the mapping model are presented, in an attempt to provide fewer but potentially equally good columns as the pre-generation. Both C++ and the Mosel programming language have been used for implementing the MIP models, pre-generation methods, and column generation methods.

The six resulting solution models are tested using five constructed test instances with one varying scaling parameter. Some test cases have proven to be too hard to solve, requiring restrictions to the number of columns pre-generated to obtain comparable results. The solutions obtained from the tests show that the different solution methods have different applicabilities, the column generation methods or an implementation of the mapping model with restricted pre-generating for large problem instances seem to be the best suited for solving the presented problem. The scaling parameter used has its expected effect. Some of the solution methods presented provide optimal solutions and some provide good heuristic solutions to the test instances used.

Sammendrag

Med økende bruk av nettskyen tar stadig nye forretningsmodeller form, og fokuset i markedet er differensiering og tilpasning til de stadig økende krav som stilles av kundene. For å tilfredsstille næringslivets krav til nettskyen er det å kunne tilby garantier på tjenestekvalitet avgjørende. Den kommende generasjonen av *cloud brokers*, et bindeledd mellom kunder og tilbydere, kommer til å være de som har fokus på å tilby sine kunder garantier på tjenestekvalitet.

Denne masteroppgaven ser på nødvendig planlegging en kombinert nettverksoperatør og cloud broker må gjennom for å overleve i dagens og fremtidens skyverden. En slik broker må bestemme sammensetningen av sin kundeportefølje, hvilke tilbydere tjenester skal anskaffes fra, samt hvordan oppnå riktig tjenestekvalitet i henhold til kundenes krav.

I denne masteroppgaven presenteres tre heltallsmodeller, en linkbasert modell, en stibasert modell og en stikombinasjonsmodell, for å velge kundeportefølje med maksimal profitt, med hensyn til forsinkelse, tilgjengelighet, nettverksruting, tilordning av reservekapasitet i nettverket og valget av tilbyder for hver tjeneste. Den stibaserte modellen og stikombinasjonsmodellen benytter seg begge av pregenereringsmetoder. Tre heuristiske kolonnegenereringmetoder er utviklet i et forsøk på å tilby raskere problemløsning, disse produserer begrenset, men potensielt like god input til stikombinasjonsmodellen som pregenereringen. Modellene, pregenereringsmetodene og kolonnegenereringsmetodene er implementert ved hjelp av C++ og programmeringsspråket Mosel.

De seks presenterte løsningsmetodene er testet ved bruk av fem konstruerte testinstanser og en varierende skaleringsparameter. Noen kombinasjoner av testinstanser og løsningsmetoder har vist seg vanskelig å teste, og størrelsen på pregenerert input ble begrenset for å kunne få sammenlignbare resultater for disse kombinasjonene. Løsningene fra testene viser at løsningsmetodene passer til forskjellig bruk, en av kolonnegenereringsmetodene eller implementasjonen av stikombinasjonsmodellen med begrenset input for store probleminstanser viser seg å passe best å løse det presenterte problemet. Bruk av skaleringsparameteren har den forventede effekten og gir løsninger med passende egenskaper. Noen av løsningsmetodene presentert gir optimale løsninger, mens andre løsningsmetoder gir gode heuristiske løsninger på det presenterte problemet.

Contents

List of Figures

List of Tables

Abbreviations

BC	The combined role of a **B**roker and a **C**arrier
BCL	Xpress-BCL **B**uilder **C**omponent **L**ibrary
CRM	**C**ustomer **R**elationships **M**anagement
CSBQANR	**C**loud **S**ervice **B**rokering with **Q**uality **A**ware **N**etwork **R**outing
IaaS	**I**nfrastructure-**a**s-**a**-**S**ervice
ITU	**I**nternational **T**elecommunication **U**nion
LFM	**L**ink-**F**low **M**odel
LP	**L**inear **P**rogramming
MIP	**M**ixed **I**nteger **P**rogramming
NaaS	**N**etwork-**a**s-**a**-**S**ervice
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
PaaS	**P**latform-**a**s-**a**-**S**ervice
PFM	**P**ath-**F**low **M**odel
QoS	**Q**uality **o**f **S**ervice
SaaS	**S**oftware-**a**s-**a**-**S**ervice
SLA	**S**ervice **L**evel **A**greement
SMEs	**S**mall and **M**edium **E**nterprise**s**
SPPRC	**S**hortest **P**ath **P**roblem with **R**esource **C**onstraints
VDC	**V**irtual **D**ata **C**entre
VM	**V**irtual **M**achine
VN	**V**irtual **N**etwork

Chapter 1

Introduction

With the expanding cloud computing market new business models emerge. Traditionally there are the roles of providers, consumers, carriers, and in some cases brokers, that mediate between providers and customer. Increasingly more extensive demands from cloud customers give rise to a new level of requirements for the cloud market. This in turn increases the need for differentiation and adaption to the demands of the customers. In the role as a cloud broker it will be beneficial to adapt and enhance provided services to the customers and to provide a single connection point to a customised service delivery. Revenue potential for cloud services in the business segment increases dramatically when quality of service (QoS) attributes involving high performance, such as low latency and high availability, are offered. These are aspects that decrease the obstacles to extensive use of cloud services, from a potential cloud customer's perspective. (Undheim et al., 2012)

According to Undheim et al. (2012) the next generation of brokers will consist of the QoS-aware brokers with cloud connectivity who take requirements from the customers, find the providers that can meet them and deliver the service through their own network connections. Popular QoS attributes, both amongst the cloud customers and in relevant literature, are bandwidth, latency and availability. Especially availability has been extensively reviewed in literature, as it poses an extra challenge in linear modelling by having a non-linear nature when computed for paths through a network. To increase the availability provided to the customers, provisioning of backup paths, along with the primary paths, are used for availability sensitive connections.

A network QoS aware cloud broker has to decide what customers to select and which providers that can provide the most suitable service provisioning for those customers' service demands. In addition, such a broker has to make decisions on how to route the traffic to and from these providers' data centres in compliance with the requested QoS levels for each customer. The resulting problem from this combination of decisions is throughout this thesis referred to as a Cloud Brokering with Quality Aware Network Routing (CSBQANR) problem. This problem includes availability consideration and provisioning of backup to services requiring a high availability connection.

Three mixed integer programming (MIP) models are presented to address this problem, a link-flow model (LFM), a path-flow model (PFM) and a mapping model, where mappings consists of combinations of primary and backup paths. These models handle the non-linearity nature of the availability requirements by using different approximations. To solve the presented models, the LFM and PFM are implemented in the Mosel programming language, the mapping model is implemented in C++ with the *Xpress-BCL Builder Component Library* (BCL). The pre-generation methods for the PFM and the mapping model implementations are also implemented in C++. In the attempt to provide faster solution methods to the problem, three heuristic column generation methods are developed to provide the mapping model implementation with fewer, but potentially equally good columns; these are implemented in C++.

The solution methods for solving the CSBQANR problem is tested on five constructed test instances, the different methods are compared and their applicability in practical planning is assessed. Given input data reflecting a real world problem, the solution methods could provide answers to tactical decision problems. With the use of different input networks provided from the user one can also see what effect changes in the infrastructure will lead to, and give indications of good solutions on strategic issues.

The rest of this thesis is organised as follows. Chapter 2 gives a brief introduction to important aspects of virtualisation, cloud computing and QoS, especially focusing on cloud computing roles and availability. This is followed by a description in Chapter 3 of the background for the CSBQANR problem, detailing assumptions made and how different aspects of the CSBQANR problem are modelled. Chapter 4 gives an overview of

relevant work related to this problem with a main focus on modelling of availability and backup path provisioning. The CSBQANR problem itself is described and detailed in Chapter 5. The CSBQANR MIP models, the LFM, the PFM and the mapping model are presented in Chapter 6, as well as the path and mapping pre-generating algorithms and three heuristic column generation methods for a more effective pre-generating. This is followed by the presentation of test instances, test setup and test cases, as well as a study of the computational results obtained from running the tests in Chapter 7. Chapter 8 gives the concluding remarks and proposes future direction for the work presented in this thesis.

Chapter 2

Theory

In this chapter important theoretical concepts for this thesis will be presented and described. First virtualisation will be defined, then the focus will be on cloud computing and the chapter ends with an introduction to QoS and availability. This chapter is based on the theoretic study done in the specialisation project Braaten and Holmen (2013) and so sections 2.1 and 2.2 are slightly altered sections from the specialisation project, while Section 2.3 consists in the most part of new material. Section numbers in this thesis does not necessarily coincide with section numbers in Braaten and Holmen (2013) and the section numbers specified in this thesis will point to sections in this thesis only.

2.1 Virtualisation

The term *virtualisation* broadly describes the separation of a resource or a request for a resource from the underlying physical delivery of that service (VMWare Staff, 2012, Accessed: 2014-02-03). An important benefit of virtualisation is better and more flexible utilisation of resources. The underlying physical infrastructure responsible for delivering any virtual resource is commonly referred to as the *substrate*. Virtualisation is one of the key enabling technologies of cloud computing. What makes virtualisation so important for the cloud is that it decouples the software from the hardware, it generalises the physical infrastructure into a virtualised pool of resources which could be shaped to fit the demands of users at any time.

A virtual machine (VM) is a software instance imitating a computer environment which can be utilised as a real physical computer. VMs allow for multiple operating systems to be run on a single physical system and share the underlying resources, this is known as *partitioning*.

Network virtualisation decouples the logical structure of a computer network, from the physical structure. This is done by virtualising network nodes and links, enabling the creation of new virtual networks (VNs) on top of the physical structure. Multiple virtual networks can coexist on the same substrate network. This technique can be used for abstraction of network hardware, resource sharing and isolation, giving the users exclusive access to their own networks and providing simpler interfaces.

2.2 Cloud Computing

Cloud computing is the new direction in which the computing world is evolving. The term cloud computing is the basis of a wide range of understandings, but the National Institute of Standards and Technology (NIST) defines it as: *a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction* (Mell and Grance, 2011). The most obvious benefit of cloud computing is the possibility of utilising computing resources in a more efficient manner. The idea is to dynamically distribute and share a large pool of resources between users using virtualisation, aggregating their variable demand, thus enabling a more evenly distributed load and utilisation of a higher percentage of the pooled resources. At the same time the management and underlying structure of the cloud is hidden from the user, who can use the amount of storage, applications and processing power he needs without considering the underlying infrastructure.

2.2.1 Cloud Computing Roles

In a cloud market there are several different actors that interact in different ways. A *cloud consumer* is a user of the cloud services provided by a cloud provider.A *cloud provider* is

the organisation responsible for making a service available to interested parties. A cloud provider acquires and manages the computing infrastructure required for providing the services, runs the cloud software that provides the services, and makes arrangements to deliver the cloud services to the cloud consumers through a network access.

A *cloud broker* is an entity between the consumer and provider that manages the relationship between the two, either by conveying the services from the provider directly to the consumer or providing the service in an extended or otherwise improved form. The cloud broker also manages the use and performance of the service. Liu et al. (2011) define three types of brokers; the *intermediator* who adds value to an existing service by enhancing some of its capabilities, the *aggregator* who combines and integrates fixed services from different cloud providers into a new value-added service, and the *arbitrator* who combines and integrates services from different cloud providers, which can be dynamically selected based on customer requirements and negotiated relationships.

A *cloud carrier* is the entity that transports the service from provider, either via a broker or directly to the consumer through a network. A *cloud auditor* is an actor that often also is defined, but it is not relevant to this thesis. Figure 2.1 illustrates how the different cloud computing roles interact with each other.

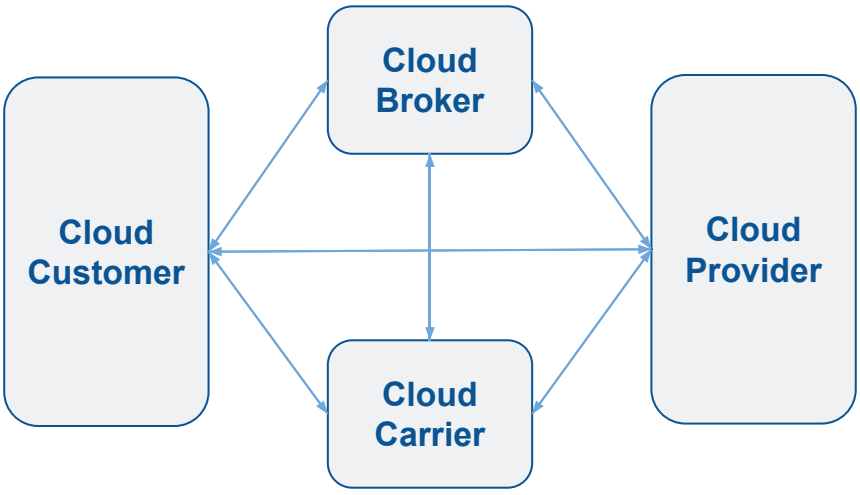


FIGURE 2.1: Interaction between cloud computing roles, modified figure from Svaet et al. (2013)

2.2.2 Cloud Computing Service Models

NIST defines three service models in the cloud ecosystem; *Software-as-a-Service*(SaaS), *Platform-as-a-Service* (PaaS) and *Infrastructure-as-a-Service* (IaaS) (Mell and Grance, 2011). In the SaaS model, the cloud user is provided with software or applications deployed in one or several of a provider's data centres which the user can access through a thin client interface, for example a web browser or a program interface. The user has no control over the underlying cloud infrastructure, network, storage, or software configurations, with the possible exception of a few user specific settings.

With PaaS the user has a higher level of control over the applications and its environment. PaaS provides the user with the possibility to deploy consumer-created or acquired applications in addition to managing these applications in the cloud infrastructure. A user can by using tools and programming languages provided by the provider build its own applications and deploy these in the cloud. Access to the network, server or operating system is still not given to the user; the configurations available to the user are environmental or application specific.

In the IaaS model the user has access to processing, network and other computing resources to deploy and run arbitrary applications. The user can manage and control operating systems, storage and deployed applications, but still has no control over the underlying cloud infrastructure; however he can have limited control over selected network components.

In other literature, *Network-as-a-Service* (NaaS) is also defined as a service model. International Telecommunication Union (ITU) defines NaaS as *a category of cloud services where the capability provided to the cloud service user is to use network/transport connectivity services and/or inter-cloud network connectivity services* (ITU, 2012). Examples of such services are providing virtual private networks, bandwidth on demand and mobile network virtualisation where the network operator sells access to his network to third party mobile virtual network operators. Figure 2.2 illustrates the interaction between the different cloud computing service models, cloud users and virtualisation platforms.

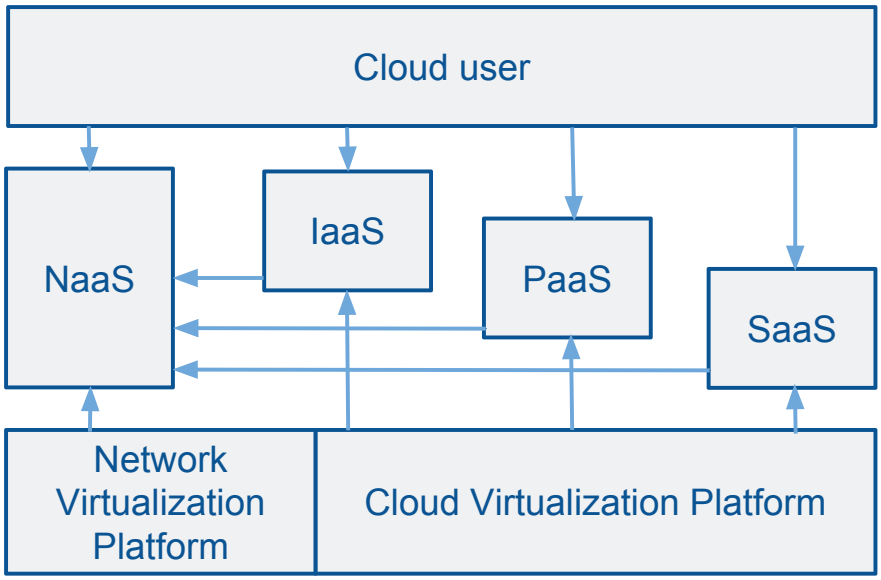


FIGURE 2.2: Interaction between cloud computing service models and network virtualisation (Baroncelli et al., 2010)

2.2.3 Cloud Computing Price Models

The cloud computing market prices its products mainly by the use of one of three price models; *pay-as-you-go*, *flat rate* or a *mixture model*. In the pay-as-you-go model the customers are charged for their actual usage. This is the analogue to most pricing found in markets of physical goods, where market forces drives the price down to the products' marginal cost. This price model has been problematic to introduce to the cloud market as cloud products often have a marginal cost equal to zero. In the flat rate model the customers are charged with a fixed amount per time unit (day, month) regardless of their usage. This model makes billing and management easier for both the provider and the consumer, but could seem inappropriate when resource usage is highly variable among users. The mixture model is a combination of the latter two, where the customer pays some fixed fee per time unit including a specified amount of usage, and is charged additionally for all usage exceeding this limit. (Han, 2009)

2.3 Quality of Service (QoS)

Quality of service (QoS) refers to the collection of networking techniques and technologies a network needs to provide a better service to selected traffic, using different

technologies to achieve the goal of providing guarantees on the quality of different network attributes. Common QoS elements of network performance include bandwidth (throughput), latency (delay), availability (uptime) and error rate.

Different kinds of services and customers have different needs in relation to QoS. Services with real time interaction, like online games and videoconferencing, benefit from low latency and high bandwidth compared to video streaming that can tolerate high latency and jitter, but still requires a high bandwidth connection. Safety-critical applications depend on little or no downtime while other attributes can remain less important.

QoS requirements can either be referred to as soft or hard, depending on the degree of strictness of guarantees that can be provided for QoS. Soft QoS may not include any guarantees to the actual experienced QoS, similar to the best effort characteristic of standard computer networks, while hard QoS requirements will include strict guarantees to the experienced QoS.

2.3.1 SLA - Service Level Agreement

QoS in a business relationship is often regulated through a service level agreement (SLA) where requirements for different QoS performance measures are agreed upon. An SLA is an agreement where the service and delivery of a service are formally defined. It often contains how different aspects of a service should be experienced from the customer's point of view. The SLA usually specifies limits and average levels of performance measures to satisfy the customers demand for the experienced quality. Average levels are often provided based on the network history and the limits as guarantees are based on the average level of performance where a safety margin is added to account for the variance in the performance. Several methods are developed to decide this safety margin, see Clemente et al. (2005) and Zhou and Grover (2005).

2.3.2 QoS with Network Virtualisation

There are many different approaches to the implementation of network virtualisation. One of the easily identifiable differences is their approach to QoS guarantees. Some of the known network virtualisation approaches, used in data centres network virtualisation, that do support QoS guarantees are Oktopus, SecondNet, Gatekeeper and CloudNaaS. (Bari et al., 2013)

The most common performance characteristics considered by different network virtualisation approaches in addition to bandwidth are latency and availability (Gonzalez, 2014). Bandwidth guarantees are implemented by reserving all or parts of the required bandwidth along assigned paths in the physical network. It is common to classify different services according to priority and only reserve large amount of bandwidth for the highest priority services. The proportion of required bandwidth actually reserved is determined by the degree of softness or hardness of the QoS guarantee.

As cloud computing services are delivered over a network their performance is highly dependent on the performance of the delivering network. Cloud computing services are today mainly provided over the Internet, leaving them with no better than best effort QoS. As shown in Svaet et al. (2013) the business segment of cloud customers are willing to pay extra for premium quality connectivity services, making a focus on QoS a valid business venture for a cloud broker.

2.3.3 Availability

Network availability is a key performance parameter for a network and it is a good indicator of the network resilience (Clemente et al., 2005). The availability of a system is defined as the fraction of time the system is available for use to the user during the entire service time and can also be defined as the probability that the system will perform as usual during a given time period (Zhang et al., 2003). Unavailability could be caused by failures in the network's components, data centre failure or other circumstances. Availability (D_i) of a network component i is defined by the use two terms; mean time

between failures (MTBF) and mean time to repair (MTTR) (Clemente et al., 2005):

$$D_i = \frac{MTBF}{MTBF + MTTR}$$

Let $\mathscr{N}_k$ be the network components along the connection path k. Assuming the failure probabilities of network components are independent, the availability (D_k^P) of a connection path k can be determined by the product of the availability of its network components, as shown in Equation 2.1.

$$D_k^P = \prod_{i \in \mathscr{N}_k} D_i \tag{2.1}$$

The service reliability of a connection could be improved by pre-reserving extra resources to be used in case of failure along the primary path. Several such protection schemes exist i.e., link-rerouted vs. path-rerouted, dedicated vs. shared, etc. For the link-rerouting scheme rerouting alternatives for each link in the path are provided and some amount of capacity is reserved on each of these link alternative sub routes. If a link is unavailable, its alternative sub route is used to route the traffic flow around the malfunctioning link. In the path-rerouting scheme resources are reserved along an alternative path (the backup path). If one or several parts of the primary path are malfunctioning the traffic is switched to the backup path. Dedicated protection is achieved by reserving the sufficient amount of capacity along the backup path or links to support all of the traffic along the primary path. In the shared protection scheme several services can share the capacity reserved for backup. This concept's validity is based on the low probability of several simultaneous failures in the network, and so the probability of several services needing the backup capacity at the same time is small. This approach requires a lower amount of capacity reserved for backup compared to the dedicated approach.

The probability of the case where at least one of the primary (A) or backup (B) path is available at a certain time is the sum of the availabilities of each of them minus the probability of them being available at the same time; the union probability equation.

$$P(A \cap B) = P(A) + P(B) - P(A)P(B|A)$$

Assuming independent failure probabilities of overlay network components, the availability of a network mapping involving a primary path k and backup path b, with the network components $\mathscr{N}_k$ and $\mathscr{N}_b$, respectively, can be calculated as follows.

$$D_{kb}^{M} = D_{k}^{P} + D_{b}^{P} - D_{k}^{P} \prod_{i \in (\mathscr{N}_b - \mathscr{N}_k)} D_i \tag{2.2}$$

Note that this equation does not require the primary path and backup path to be disjoint. This equation is correct for multiple simultaneous connection mappings when using dedicated path protection. When using shared backup path protection, the equation will only be an optimistic bound of the actual availability. Assuming the network components considered is the network links along the paths. For the equation to be correct for shared backup path protection, the D_i for links that are only used in the backup path b ($\mathscr{N}_b - \mathscr{N}_k$) must include that link's availability, as well as the probability that the shared backup capacity is available. That is, the probability that a link failure has not affected another service's primary path requiring the same backup capacity resource.

Chapter 3

Background

This chapter describes the background for the CSBQANR problem, which is introduced in Chapter 5, and the way different aspects relevant to the optimisation models presented later in this thesis are viewed. The basis of this problem is the same as in the specialisation project Braaten and Holmen (2013) so several of the topics mentioned here are the same, with the exception of the business motivation, the addition of availability and backup provision. Section 3.1 has been rewritten, but still covers many of the same concepts as earlier work, while sections 3.2 to 3.6 have undergone minor updates only. Sections 3.7 and 3.8 are completely new.

3.1 Business Motivation

The CSBQANR problem as well as the optimisation models modelling it, are in part based on the network infrastructure and business positions of some large telecommunication operators. Today, some telecommunication operators offer a small range of software services to its customers, bought from and provided by large software companies with cloud data centres. The services are hosted by external providers, while the telecommunications companies act as aggregators (see Section 2.2.2), selling these services to cloud consumers, without necessarily providing the network connectivity to the service.

Having a large customer base in both the private and business segment in addition to controlling a large network infrastructure, telecommunications companies could benefit

from an orientation towards the role as QoS aware cloud brokers, since existing network assets could be developed with comparatively low investments enabling this (Svaet et al., 2013). This could be achieved through cloud computing initiatives where the telecommunications companies add value to the existing cloud service offerings by acting as both brokers and carriers, supplying their customers with services in the cloud, provided by a third party, and offering certain connectivity guarantees through carrying the network connectivity through their own network infrastructure in combination with establishing SLAs with the providers.

In a long term perspective, the capabilities of traditional carriers could be exploited by bundling SaaS and cloud connectivity with assured quality. The targeted market segment of many telecommunications companies are small and medium enterprises (SMEs) who often have a need for customer relationship management systems (CRM), office productivity tools and telepresence services. These telecommunications companies have a business opportunity in the convenience the SMEs will experience by buying these services bundled with network connectivity from one broker instead of having to interact with many different providers with different SLAs. This would ensure cost-efficiency at the same time as relieving the customer of managing SLAs etc. The telecommunications companies could ensure end-to-end service delivery, which is seen as important for the SMEs. (Svaet et al., 2013)

3.2 The Role as both Broker and Carrier

According to Svaet et al. (2013), the next generation of brokers will be the QoS-aware broker with cloud connectivity that takes requirements from the customers. Such a joint role faces a two part problem. The assignment problem, where services are to be allocated to providers that can support the demand from customers, and the routing problem involving connecting the customers to the services through the network in a way that meets QoS demands. Both the providers and customers are external actors in this thesis. Issues like energy usage and how services are deployed within the data centres lie with the provider and are therefore outside the scope of this thesis. In addition the assumption that every provider has sufficient capacity to deploy all services demanded in this problem is made. The basis of this assumption is that the demand generated by

the customers in the CSBQANR problem, for a single Broker-Carrier (BC), is presumed to be small compared to the aggregated demand each of the providers experience. The combined role as broker and carrier is illustrated in Figure 3.1.

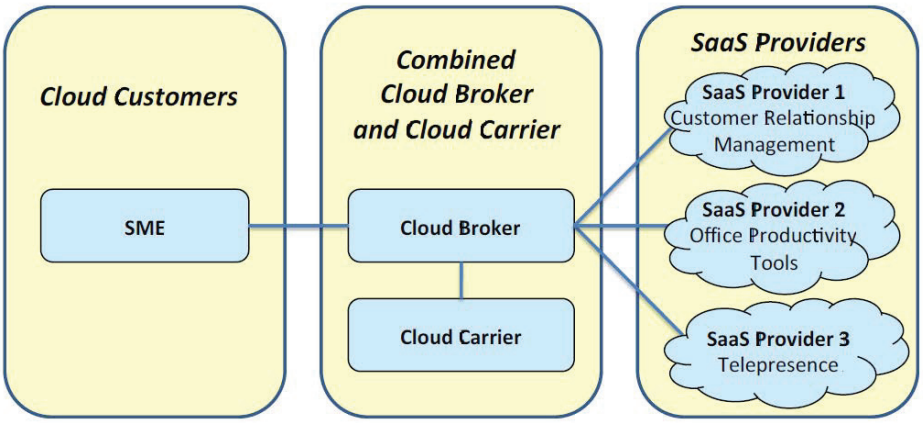


FIGURE 3.1: The combined role as broker and carrier, modified figure from Svaet et al. (2013)

The cloud BC, considered in this thesis, operates and owns its own network, spanning a large geographical region. The geographical region largely determines the customer base to be considered, as the BC wants to exploit its existing network infrastructure to provide better than best effort connection to cloud services. The BC owned network will often not include direct network connections to the provider's public clouds, thus the network considered will often be a combination of BC owned network and leased network links. Given SLAs for any leased links added to the network, these leased links can be treated as regular network links in the BC owned network.

3.3 QoS Requirements

In the selection of what QoS attributes to consider several aspects have been evaluated. The most frequently discussed QoS attributes in relevant literature are bandwidth, latency and availability. These are some of the most important QoS measurements as seen from the customers, especially those using SaaS. The services provided by telecommunication companies are mostly CRM and office services, services relying on a reliable bandwidth connection, and telepresence services relying on bandwidth, high availability and a low degree of latency in order to function satisfactory. The customers' understanding of the importance of the offered QoS attributes is paramount for these network products

to sell and to give the BC a differentiation factor. Bandwidth, latency and availability are parameters understood and appreciated by most SaaS customers in SMEs.

3.4 Customers and Demand

The BC's customers each require one or more services that must be obtained from a provider by the BC. Customers may demand bundles of services, requiring all of its services to be provided for the customer to choose the BC and generate revenue for the BC. The reason for including service bundles is to increase the cost-efficiency for the customers, and to provide a simplified managing situation for the customers by reducing the number and complexity of their business relationships, mentioned in Section 3.1. A customer may offer a demand for multiple different bundles of services, and can therefore be split into a logical customer for each service bundle, where the decision to serve each logical customer can be made independently. Single services, not bundled into service bundles, are for all intents and purposes considered as logical single service bundles in this problem.

Services provided in a typical cloud computing scenario can range from simple data backup services with traffic mostly in one direction (from the customer to the cloud) with low availability demands, to media streaming service with traffic in the opposite direction, and telepresence services with strict requirements on latency, availability and bandwidth in both directions. The services reviewed in the CSBQANR problem have requirements to the amount of bandwidth to be reserved in each direction. They also restrict the maximum amount of latency that can be tolerated to run the requested services effectively and the maximum allowed downtime or minimum availability.

3.5 Time Aspect

The assignment problem this thesis concerns is a one-time assignment problem that allocates services to providers to enable servicing a customer. This allocation is considered a constant or long time static allocation, changes in the demand or customer base will render it necessary to re-solve the CSBQANR problem. The services considered

imitate a subscription nature, where the customer service demand is relatively constant and does not change significantly during the period of time the problem depicts. The variation in demand of bandwidth and the need for dynamically changing the mapping of services to providers is assumed to be sufficiently small to be disregarded within the time period of the CSBQANR problem.

3.6 Prices

The most common pricing schemes in the cloud computing market are prices based either on amount of time used (in hours, days or months) or amount of storage or processing power used (see Section 2.2.3 for more on price models). The myriad of different price structures have made modelling with consideration to these structures a complicated matter which often has been circumvented in relevant litterature. In a long term perspective it is possible to approximate prices to reflect the general behaviour of the original price structure.

The time period the CSBQANR problem regards is sufficiently short to assume placement prices at each provider to be constant. Additionally, the effect different kinds of price models would have is sufficiently small in the time frame this problem regards, so the prices used are all translated to the same flat rate price model without considering the variations within the period. All prices used in the CSBQANR problem are therefore flat rate prices that do not vary through the period the problem depicts.

3.7 Availability

Network availability is defined as the fraction of the time a connection is available to the user (see Section 2.3.3). This availability is based on the combination of several network components which each have an expected availability. In this thesis non-failing nodes are assumed, as argued in Schupke and Rambach (2006). Different kinds of services have different needs in regards to availability. Safety critical services and emergency services are examples of services with high availability requirements, while services like word processing and customer relationship systems have less strict requirements in regard to

availability. If the availability along a single path does not fulfil a service's requirement, a backup path can be provided to be used for traffic routing in the cases when the primary path is unavailable.

A BC with an SLA with a customer is required to fulfil the availability requirement outlined in the SLA, but has no positive incentive to provide a higher availability to the customer. Providing the customer with a higher level of availability than needed could have an influence on the BCs profitability, as well as the ability to take on additional customers. Therefore it is beneficial to the BC not to provide a higher level of availability than the level of the customers' SLA availability requirement. Note that the term availability in this thesis refers to the theoretical expected availability level over a long period of time. In the short term, the experienced up-time for a majority of the users will be 100%, while a minority of the users will experience some downtime, in which case the measured availability will be significantly lower. Because of these large variations in availability in the short term, a strict SLA availability limit will be extremely costly to uphold unless one only considers the long term.

3.8 Backup Capacity

As described in Section 2.3.3 there are different ways of providing backup to a connection in a network. Dedicated protection allows no sharing of resources shared for backup, and will on average reserve a lot more capacity than needed. The shared protection scheme is a more economic scheme both in relation to capacity and demands regarding the network. In this scheme several services can use the same arcs as part of their backup path and the capacity reserved for backup on each arc does not have to cover the demand of all of the services at once. Shared protection schemes usually only allow overlapping backup paths for disjoint primary paths. Based on the assumption that the possibility of several services' disjoint primary paths going down at once is much lower than the probability of one link failure, the amount of reserved capacity on the shared backup arcs does not have to be as large as the sum of the demands of the services. The amount of backup capacity has to be sufficient to support the demand of each service alone, so a lower limit on the reserved capacity is the largest single backup demand amongst the services using the arc as backup, referred to as single backup path bandwidth allocation

(Cui et al., 2002). However allowing a large number of services to share the same arc for backup will increase the possibility of multiple failures leading to more than one service needing the backup arc at the same time (Zhang et al., 2003). In this case it might be beneficial to reserve more capacity than the maximum demand of the services. Several methods for deciding the amount of backup capacity to reserve has been presented in literature, see eg. Cui et al. (2002).

In most cases the use of backup path protection schemes is accompanied by the requirement of link-disjoint primary and backup paths. This requirement ensures that a single link failure will not lead to two services requiring the same backup resource. In a very dense network this is not such a severe restriction on the backup provision as there are several possible paths between customer and provider nodes. For less dense networks, this disjoint primary and backup paths requirement may be problematic. If there are few possible link-disjoint paths between customer and provider nodes, the disjoint requirement can restrict the BC to only serve very few or no customers at all unless most primary paths have sufficient availability levels on their own and backup is not needed. Since the problem described in this thesis has the objective to use the BC owned network as a main rule and BCs may not have a large, dense networks for cloud use, some alternatives to the disjoint requirement will be used in solving the CSBQANR problem. There will in every case be some restrictions in regards to overlap between primary and backup paths, but overlap will be allowed in some cases as long as it does not lead to too low availability levels.

Chapter 4

Related Work

This chapter will present some work in relation to operations research done on connection-provisioning, backup network provisioning, backup path allocation and backup bandwidth allocation, presenting both path-flow and link-flow models. Related work concerning network virtualisation, embedding problems, service deployment and resource optimising in a cloud computing setting is presented in Braaten and Holmen (2013). In this chapter the term article always refers to the work being reviewed, and not this thesis.

The CSBQANR problem and this thesis is based on the specialisation project report Braaten and Holmen (2013). The report introduces important concepts and ideas relevant to the cloud computing, virtualisation and QoS areas, and presents a problem for cloud service allocation and connectivity challenges met by a joined broker and carrier role, considering two intrinsically different QoS requirements (bandwidth and latency). The problem presented in the report is similar to the CSBQANR problem presented in Chapter 5, apart from some noticeable differences. The project's problem includes the ability to lease capacity on specified links owned by other operators than the BC owning the main network. The resulting model is a link-flow model (LFM) in which multi-pathing and reserving different up- and down- paths is allowed. Excluding these aspects, this model is the starting point of the LFM presented in this thesis (Section 6.2). The solving of the problem in Braaten and Holmen (2013) is done by a Mosel implementation, which is tested with different problem instances. The project report finds that multi-pathing, combined with a maximum allowed latency, complicates the solving

of real world sized problem instances when using a generic MIP solver. Some future extensions are presented, among them adding more relevant QoS attributes, relaxing parts of the problem, developing some pre-solving methods and providing more complex solution methods to be able to solve real world problem instances of the problem within a reasonable amount of time. These extensions have been attempted in this thesis.

Zhang et al. (2003) present a connection-provisioning framework to satisfy customers' availability requirements using appropriate protection schemes. This article considers wavelength-division multiplexing (WDM) mesh networks, but the methods proposed are still applicable for other network topologies as in this thesis. The network resource usage for services in the network is to be optimised for one-path-satisfiable connections, whose availability requirements can be satisfied without using backup paths, and protection-sensitive connections. For the one-path-satisfiable connections a linear program is presented where non-linear availability equations are linearised into a sum of costs for each link defined as the negative logarithm of the link's availability. This program will provide the most-reliable primary paths. For the protection-sensitive paths dedicated protection is used as an initial study. A mathematical formulation is presented, but the availability requirements are in this case not possible to linearise. To resolve this issue some approximation approaches are presented. The first is simply to assume satisfactory availability levels whenever a backup path is provided for a connection. The protection scheme may in most cases significantly improve a connection's availability and it is assumed that this approximation will hold for most of the connections used. Another approximation is to maximise the availabilities of the primary paths and at the same time minimise the total number of links used for backup. The testing of the different schemes gave good results where most of the connections' availability levels are fulfilled; the best results were given by a combination of the two approximations, where the unsatisfied connections from the linear program using the first approximation were satisfied using the latter approximation. However, an issue with these schemes is the over-provisioning of availability as none of the proposed solutions will consider the availability of the backup paths directly to avoid reserving paths with too good availability values. The proposed path and mapping models (sections 6.3 and 6.4) in this thesis will take this into account and provide a solution where excess availability is avoided. The LFM presented in Section 6.2 uses a

similar assumption as the first availability approximation presented in this article, and also performs linearisation of single path availability by using logarithms.

Guo et al. (2011) propose two shared backup network provisioning schemes for virtual network embedding; shared on-demand approach and shared pre-allocation approach. They argue that to enable a reliable infrastructure, effective backup mechanisms must be provided to protect against network failures and by using a shared backup bandwidth approach, the required backup bandwidth can be reduced to free capacity to enable the providing of more customers. Two linear program formulations are presented based on a link embedding procedure. A set of possible bypass paths is pre-computed for each link used in a primary path. The shared on-demand approach tries to minimise the total amount of resource reserved for primary and backup flows, while limiting the total bandwidth reservation on each link, and ensuring that the primary flow can support the demands and that restoration can be achieved for every primary flow. The shared pre-allocation approach allocate backup bandwidth in advance for each link so that the allocated bandwidth should be able to protect the maximum allowed primary flows on each link. The objective maximises the total protected bandwidth of the network and the program further ensures that all bandwidth allocated for primary flows should be fully protected by restoration flows over bypass paths and that every network failure can be supported by the overall allocation. The two approaches are tested in a discrete-event simulator against two corresponding approaches using dedicated backup instead of shared. The sharing approaches provide the best protection per revenue and have different advantages and disadvantages. The shared on-demand needs to be implemented during each VN embedding process, while this is done once with the shared pre-allocation. However, the pre-allocation approach has to always maintain the backup bandwidth regardless of VN requests and may therefore not be as effective at low VN request loads.

Cui et al. (2002) address the issue of backup path allocation in an overlay network when a failure in the physical substrate network could lead to multiple failures in the overlay network. In this case link-disjoint primary and backup paths in the overlay network is not sufficient to ensure that one link failure will not lead to failure in both the primary and backup paths. A correlated overlay link failure probability model is

presented and the joint failure probability of two links being affected by one failure is used further in a two part integer programming problem where the primary path is found by a shortest path algorithm and the backup path is found by minimising the joint failure probability given the primary path. Several approaches to determine the amount of bandwidth to reserve for backup is presented. Full backup path bandwidth allocation and single backup path bandwidth allocation are two of them where respectively the aggregated bandwidth demand of all connections is allocated and the maximum of the bandwidth demand is allocated on each link. Extensive simulations are conducted to test, and the results show that in terms of robustness the approach is near optimal and up to 30 % better than ignoring link failure probabilities. The models presented in this thesis regard a joint failure probability as well, though it is defined for paths and not for links as in this article. Instead of basing them on the probability of the two links failing as a result of a physical link failing as presented here, the models in this thesis base the failure values on the probability of the two paths failing simultaneously, regardless of the reason. It can however be made to include the meaning presented in this article without loss of generality. A variant of the single backup path bandwidth allocation is also used in all of the models presented in this thesis, however may be modified in some cases and extended to allocate more bandwidth with the increase of number of connections using the backup link.

Józsa and Orincsay (2001) regard the off-line global path optimisation in telecommunication networks where backup paths use shared bandwidth reservation and present three algorithms that can solve the problem efficiently. They argue that resource reservation can be reduced significantly without reliability degradation by using shared protection compared to the dedicated protection scheme. The problem is formally presented simply by reserving capacity along the primary paths and minimising the sum of required backup bandwidth reservation on each edge. The required backup reservation on an edge is equal to the maximum bandwidth required for backup on that link for any single link failure, referred to as single backup path bandwidth allocation by Cui et al. (2002). To solve the problem Dijkstra's shortest path algorithm is used to find the primary paths; the edges in the primary paths are then removed from the network to achieve disjoint backup paths when Dijkstra's is run again to find backup paths. First of the three algorithms a post-processing method is derived, which has a complete network

configuration as its starting-point, searches for the maximum required backup reservation of each column, and tries to re-route without using the corresponding failure edge in the primary path or backup reservation edge in the backup paths. The aim is to decrease the sum of reserved backup capacity. The next method is an adaptive method that uses an adaptive weight function to guide the routing of both primary and backup paths to achieve low reservations on both the primary and backup paths. The last method is an iterative method using the same weighted function used for backup paths as in the previous method to avoid the links with relatively high reservation values. The three algorithms are investigated through empirical testing and the results show that all algorithms can considerably extend the network throughput. The adaptive method is recommended as the best to use as it improves the network performance nearly to the highest degree while its running time is at a low level. As mentioned, a variant of the single backup bandwidth allocation used in this article is adopted in the models presented in sections 6.3 and 6.4. The algorithms proposed in this article are implemented using shared protection, but are not specialised to only function with this protection scheme; similarly, the work presented in this thesis can be used for both dedicated and shared backup reservation. The network routing in this article uses a procedure based on Dijkstra's Shortest Path Search algorithm, performing the routing as a shortest path problem, where the sole concern is to minimise bandwidth usage. In this thesis, two of the column generation heuristics proposed, finds paths using a shortest path problem with resource constraints, thus allowing the inclusion of additional connectivity requirements as limited path resources in addition to minimising costs from bandwidth usage. In addition, the column generation heuristics guide the paths generated by adjusting costs associated with using each link according to existing network flow, similarly to the adaptive weight method proposed by this article, although how the adaptive weights or costs are obtained differs. While this article attempts to find solutions guaranteeing continued network flow for any single link failure, this thesis attempts to offer solutions offering an expected availability of all network connections not limited to single link failures.

Schupke and Rambach (2006) present a basic link-flow model to address the generalised dedicated path protection problem with one or several disjoint paths. The model is only developed for one connection for simplicity; it is argued that extending it further

is trivial. In their availability consideration both node and link availability values are included though it is argued that one often can assume non-failing nodes. A subsequent consideration of availability is derived based on logarithms and a path-adding heuristic is presented to guarantee availability in design. However this approach can lead to sub-optimality or even fail, so an integrated consideration of availability is done. A non-linear availability equation is found based on an upper bound on the availability by summing up the unavailability values of the network components. As these values are sufficiently close to zero, the authors argue that this approximation holds. To linearise, new variables are introduced combining the edges in the network pairwise and assuming non-failing nodes. This increases the number of variables considerably and such the method is best suited for solving of small problem instances. Case studies show that the upper bound used in this method is a very accurate approximation to the actual availability. As in the latter approach of this article, the problem presented in this thesis assumes non-failing network nodes and focuses on the link failures. The use of logarithms on the availabilities combined with some pre-processing of the input data to the model presented in Section 6.3 provides a model with a different availability measure than the heuristic presented in this article. To provide an availability level on the resulting routing alternatives of the model in Section 6.3 close to the requirements the approximation of summing up unavailabilities is avoided as it may lead to over-provisioning in regards to availability.

Chapter 5

Problem Description

This chapter defines the Cloud Service Brokering with Quality Aware Network Routing optimisation problem derived from the scenario of a combined broker and carrier (BC) that is faced with a portfolio of customers (i.e. cloud consumers) who each demand a unique bundle of services. Most of its content is the same as in the specialisation project Braaten and Holmen (2013) with the exception of added availability requirement demands, the possibility of providing backup paths and the removal of leasing possibility. The first three paragraphs are approximately the same as in the project report, the rest are rewritten or new paragraphs.

Each unique service bundle consists of a set of unique services, each with their own requirements for connectivity performance and potential placements, in essence defining requirements for a virtual link connecting the service and the customer. The BC has to choose which customers to serve and to serve a customer its whole bundle must be provided. For a service bundle to be provided, each service must be placed at an eligible provider and at the same time network connectivity between the customer location and provider location, fulfilling all performance requirements, must be provided. The decision to serve a bundle is independent from any other bundles, thus a real world customer demanding a set different bundles can be split up into one logical customer for each bundle. Therefore, each bundle can be considered as unique customers.

Each customer that is served generates a revenue that is modelled as a one-time payment to the BC. This payment can be regarded as a price paid per time period the

CSBQANR problem depicts. In addition there is specified a unique cost for placing a service at each eligible provider covering all costs related to placing the service at that provider, which may include buying the service from the provider, leasing of any VMs necessary etc. These specified costs are also considered to be one-time costs for the time period the problem considers.

The network is a general collection of nodes connected by directed arcs that represents the underlying substrate network. Each node represents a network element that can route network traffic, and the arcs represent links between the network elements. In this thesis, when referring to the whole connection between two nodes, i.e. both of the opposite arcs connecting two nodes, the term link is used. Customers, providers and internal nodes are all represented by their own network element where network traffic can flow through. Customer and provider nodes double as traffic flow sources and sinks for network connections of services they consume or provide. Unless a node is acting as a source or a sink for a specific connection, it is regarded as an internal transit node for that connection and total traffic flow entering the node must be equal to the total flow exiting for that connection. There is no upper limit on the amount of traffic that can flow through a node, as the maximum flow generated in the CSBQANR problem due to bandwidth flow constraints on links is assumed to be lower than the nodes' bandwidth capacity. Since the arcs are directed, bandwidth capacity can be unique for each direction between two nodes and the flow in one direction is independent of the flow in the other direction, thus the network can be considered as a full-duplex network.

The network arcs, representing substrate links, are associated with three attributes; the amount of available bandwidth, their expected latency values and their expected availability. Depending on the specifications of the physical network the input data is based on, the bandwidth requirement can be regarded as having different degrees of hard or soft QoS guarantees. In the extreme hard QoS case the amount of bandwidth reserved for a service must be equal to the peak load bandwidth requirement of that service; this could lead to a lot of overhead and unused capacity. In a soft QoS case the reserved bandwidth represents the average utilisation of the link and the variations in used bandwidth in different services on the same link could be exploited. Substrate links can be either be owned by the BC itself or they can be leased from another network

operator. For modelling purposes the leasable link can be translated into network arcs, and dummy nodes as needed, using link performance attributes according to the SLA for the leasing agreement as basis for the arcs' performance attributes. This will not impact the modelling of the problem; it can be represented fully in the input data.

For each service a primary path through the network has to be chosen by reserving capacity on the selected arcs, in essence selecting the parts of the substrate network to be used for the routing. Each service can in addition to this primary path also require a backup path; this is determined by a cost-availability trade-off evaluation. The traffic routing to and from the provider is thought to use the same link in each direction, so if arc (i, j) is used to route traffic from a customer to a provider for a service, the arc (j, i) is used from the provider to the customer. This is the case for both the primary paths and the backup paths. Using the same path for routing in both directions through the network reflects the practice in most real life systems using centrally controlled network routing (Gonzalez, 2014). There are costs for reserving capacity on arcs used both for primary traffic and backup, in the form of a cost per unit of reserved capacity on each arc.

Each service has certain demands to the amount of bandwidth needed to support the traffic of the service, a maximum latency and minimum expected availability it can tolerate. The CSBQANR problem has to ensure that these limitations are fulfilled through the route chosen for each service. The bandwidth requirement for a service should be met by providing bandwidth flows between the end points of the connection between the service's customer and the selected provider, where the flows are equal to the service's required bandwidth in each direction. The latency requirement is met by ensuring that the chosen paths through the set of arcs chosen for a service has a total round trip latency lower than the maximum latency tolerance for that service. Only the latency of the network connection is considered, any latency in the delivery of the service internal to the provider is considered to either be subtracted from the service's latency requirement or included in the latency of the network link to the provider.

The availability demand of the service is upheld by choosing a primary path that has an equal to or higher expected availability than the availability requirement or

alternatively the combination of a primary and a backup path that together satisfy the availability demand of that service. Even though the network can be viewed as a full-duplex network, the availability of a link is the same regardless of the direction of the arc. A failure on a substrate link will for the majority of cases lead to a failure of both directed arcs between the two affected nodes in the modelled problem. As a result, the availability of a single network link between two nodes is only to be accounted for once, even though both directed arcs for that link are used in a connection path.

In cases where the availability demand of a service is not upheld by the primary path alone, a backup path is to be chosen for the service. This is done by either dedicated protection, some degree of shared protection or by single backup path bandwidth allocation (see Section 3.8). Accompanying these protection schemes are often some restrictions on the amount of overlapping between primary and backup paths of one service and also between different services' primary and backup paths. This is further discussed in Section 3.8 and will be handled in the different models and solution methods.

The objective of the CSBQANR problem is to maximise profits of the BC by maximising the revenue provided from serving the chosen customers, minimising the cost of routing the traffic through the network, reserving capacity for backup and placing the services at the providers.

Chapter 6

Optimisation Models

This chapter introduces three mixed integer programming (MIP) models for solving the CSBQANR problem defined in Chapter 5. In Section 6.1 some modelling regards are discussed, in Section 6.2 a link-flow model is introduced, a path-flow model follows in Section 6.3 and a mapping model is presented in Section 6.4. Section 6.5 finishes the chapter with a presentation of three column generation algorithms used to applicable to the MIP model from Section 6.4.

6.1 Modelling Regards

To design paths in network design and provisioning one can rely on different approaches. When modelled as a mathematical optimisation problem it can be based on link-flow models (LFMs) or path-flow models (PFMs) (Schupke, 2005). An advantage with the LFMs is their simplicity and their readability which makes them easy to understand to a novice reader. Consequently they are simpler to construct when the problem is not fully explored by the modeller. The PFMs require some pre-computation to provide the paths needed and such can be more complex to model. However, in many cases a PFM will be easier to solve than a LFM, and such is a natural extension to a LFM when dealing with large or complex problem instances. (Schupke and Rambach, 2006) In this chapter both a LFM and a PFM will be presented, where the PFM is an extended and more accurate formulation, compared to the LFM, in terms of the availability requirement.

Additionally, the PFM is developed further and a mapping model is presented. This formulation requires even more pre-computation, but could also prove to solve the problem even easier, enabling the solution of larger and more complex problem instances.

Every model description (sections 6.2, 6.3 and 6.4) is written so the reader could read each one individually, for this reason some of the descriptions for equivalent parameters, sets or constraints are reiterated in each model description.

6.2 The Link-Flow Model

This section introduces the first MIP model for the CSBQANR problem defined in Chapter 5, an LFM. The choice of modelling a link-flow model and doing so first is based on the fact that it is easily understood and easier to model when one is new to the problem, as discussed in Section 6.1. However this modelling style provides some challenges that restricts the possibilities of for example choice of approximation methods, this will be handled further in the following sections (Section 6.3 and Section 6.4).

First, the LFM's indices, sets, parameters and variables are introduced, and can be found in tables 6.1, 6.2, 6.3 and 6.4 respectively. Next, the objective function is presented, followed by the model constraints, finished by the logical constraints.

Table 6.1: Summary of the indices to the link-flow model

Index	Definition
c	Customer
i, j	Network node
p	Provider
s	Service

TABLE 6.2: Summary of the sets to the link-flow model

Set	Definition
$\mathscr{A}$	Set of all arcs (i, j)
$\mathscr{C}$	Set of all customers c
$\mathscr{I}_c$	Set of internal nodes from the perspective of customer c
$\mathscr{Q}_j$	Set of every node reachable from node j
$\mathscr{O}$	Set with every combination of different services (s, t) where $s < t$
$\mathscr{P}_s$	Set of all providers who can host service s
$\mathscr{S}_c$	Set of services demanded by customer c
$\mathscr{W}_j$	Set of every node that can reach node j

The nodes defining the arcs in $\mathscr{A}$ are numbered increasingly, so when a reference to links is needed, a link being the undirected link between two nodes or both arcs connecting the nodes, this is defined as $\forall(i, j) \in \mathscr{A}|i < j$. Similarly, the services are ordered, so when requiring all pairs of different services $\mathscr{O}$, the set of all combinations of services (s, t) where $s < t$ is used.

TABLE 6.3: Summary of the parameters to the link-flow model

Parameter	Definition
B_s^U, B_s^D	Bandwidth requirement for service s in each direction
	(U: up, customer to provider, D: down, provider to customer)
D_{ij}^L	(Expected) availability for link between i and j, where $D_{ij}^L = D_{ji}^L$
E_{ij}	Price per capacity unit used on arc between nodes i and j
G_s	Maximum round trip latency allowed by service s
H_{sp}	Cost of placing service s at provider p
I_{ij}	The amount of capacity available for reservation on arc (i,j)
J_p	Node for provider p
R_c	Revenue from serving customer c
T_{ij}	(Expected) latency on arc from node i to node j
V_c	Node for customer c
Y_s	Required minimum availability level for service s
β	Factor regulating the amount of extra capacity that must be reserved
	for an increased number of services sharing a backup arc

TABLE 6.4: Summary of the decision variables of the link-flow model

Variable	Definition
b_{ijs}	Binary variables that indicate if service s uses arc (i, j) as a part of the backup path from customer node to selected provider node (and (j, i) is used for the backup path in the opposite direction). The variables are not defined for arcs going into the customer node corresponding to service s
ℓ_{st}	Binary variable indicating if the chosen primary paths of services s and t overlap anywhere
r_{sp}	Binary variable indicating if a backup path is needed for service s to provider p
u_{ijs}	Binary variables that indicate if service s uses arc (i, j) as part of its primary path from customer node to selected provider node (and (j, i) is used for the primary path in the opposite direction). The variables are not defined for arcs going into the customer node corresponding to service s
x_{sp}	Binary variable that indicates if service s is placed at provider p
y_c	Binary variable that indicates if customer c is being served
λ_{ij}	Continuous variable indicating how much capacity to be reserved on arc (i, j) for backup use

The problem is defined as a maximisation problem, and the objective function therefore models the BCs profits consisting of the revenue from the customers being served and the costs associated with placement and connection routing for the customers' services. This objective function is defined as follows:

$$\begin{aligned} \max z \quad = \quad & \sum_{c \in \mathscr{C}} R_c y_c \\ & - \sum_{c \in \mathscr{C}} \sum_{s \in \mathscr{S}_c} \sum_{p \in \mathscr{P}_s} H_{sp} x_{sp} \\ & - \sum_{(i,j) \in \mathscr{A}} E_{ij} (\lambda_{ij} + \sum_{c \in \mathscr{C}} \sum_{s \in \mathscr{S}_c} (B_s^U u_{ijs} + B_s^D u_{jis})) \end{aligned} \tag{6.1}$$

The first term covers the revenue generated by serving the chosen customers; the second term covers the costs associated with the placement of the services of these customers

at the chosen providers. The remaining term covers costs associated with reservation of network capacity for primary routing as well as for backup use.

The requirement that the whole bundle of services of a customer must be served for a customer to generate revenue is assured by the following set of constraints:

$$y_c - \sum_{p \in \mathscr{P}_s} x_{sp} = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.2}$$

The network resource allocation to services must not exceed the capacity of the underlying network. That is, the sum of capacity reserved on an arc by all services, for their routing both from and to their provider placement, and any capacity reserved for backup must not exceed the available capacity on that arc.

$$\sum_{c \in \mathscr{C}} \sum_{s \in \mathscr{S}_c} (B_s^U u_{ijs} + B_s^D u_{jis}) + \lambda_{ij} \leq I_{ij}, \quad \forall (i,j) \in \mathscr{A} \tag{6.3}$$

Because a path uses the same links in each direction, the u_{ijs} variable is only equal to one for arcs used by the primary path in its up direction (routing from customer to provider). For this reason, u_{ijs} indicates if arc (i,j) is used the up direction, while u_{jis} indicates if arc (i,j) is used in the down direction (routing from provider to customer).

For all nodes not acting as a sink or source, we must enforce flow preservation for the bandwidth, meaning that if a path of a service enters such a node, it must also exit that node. For the internal nodes (see Chapter 5) without the possibility to act as a sink or a source, this is covered by the following constraints, for both primary and backup bandwidth reservation. The customer and provider nodes, with the possibility of acting as sources and sinks, are covered later by the constraints (6.6) to (6.9).

$$\sum_{i \in \mathscr{W}_j} u_{ijs} - \sum_{i \in \mathscr{Q}_j} u_{jis} = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall j \in \mathscr{I}_c \tag{6.4}$$

$$\sum_{i \in \mathscr{W}_j} b_{ijs} - \sum_{i \in \mathscr{Q}_j} b_{jis} = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall j \in \mathscr{I}_c \tag{6.5}$$

To ensure that the connectivity requirements are met for a service, the following constraints makes sure that a path, starting from the customer's node, is created for all services of customers that are being served. The reservation variables (u_{ijs}) are not

defined for the arcs into the customer node, thus returning the path directly into the customer node is avoided, and does therefore not have to be handled by additional constraints.

$$\sum_{j \in \mathscr{Q}_{V_c}} u_{V_c j s} - y_c = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.6}$$

An equivalent set of constraints makes sure that a backup path, starting from the customer's node, is created for any services requiring a backup path. Similarly, the backup path returning directly to the customer node is avoided by not defining the b_{ijs}-variables for the arcs going into the customer node.

$$\sum_{j \in \mathscr{Q}_{V_c}} b_{V_c j s} - \sum_{p \in \mathscr{P}_s} r_{sp} = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.7}$$

The following constraints make sure that the equivalent requirements are met for provider nodes providing the service; there has to be a chosen arc going into the chosen provider node. However, the reservation variables are defined both into and out from the provider node, as opposed to customer nodes. This allows the provider node to act as a sink for a service if selected as that service's provider, and to act as an internal transit node otherwise. This added possibility is accounted for by the second term in the constraint. This is done for both the primary and backup allocation.

$$\sum_{i \in \mathscr{W}_{J_p}} u_{i J_p s} - \sum_{j \in \mathscr{Q}_{J_p}} u_{J_p j s} - x_{sp} = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s \tag{6.8}$$

$$\sum_{i \in \mathscr{W}_{J_p}} b_{i J_p s} - \sum_{j \in \mathscr{Q}_{J_p}} b_{J_p j s} - r_{sp} = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s \tag{6.9}$$

The latency requirement for each service is ensured by the following constraints. The round trip latency, for either the primary path or the backup path, cannot be greater than the latency requirement from the customer for each service. To account for the latency in both the up and down direction, T_{ij} has to be added both if arc (i, j) is used in the up direction ($u_{ijs} = 1$) and if the same arc is used in the down direction ($u_{jis} = 1$). The same requirement must also be met for the backup path.

$$\sum_{(i,j) \in \mathscr{A}} T_{ij}(u_{ijs} + u_{jis}) \leq G_s, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s \tag{6.10}$$

$$\sum_{(i,j)\in\mathscr{A}} T_{ij}(b_{ijs} + b_{jis}) \leq G_s, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s \tag{6.11}$$

For a single path through a network, the associated availability is the product of all the links' availabilities along that path, which gives a nonlinear expression unfit for use in a linear optimisation model. There is therefore a need for linearisation of the availability expression. By using logarithms one can obtain a linear expression for one single path's availability. The larger issue arises when, as in this thesis, one can have both a primary and a backup path which together should provide the necessary availability level. The expression for the availability of such a combination will be the sum of the products of the availabilities of the links in each path minus their joint availability and therefore using logarithms will not solve the linearisation problem. To avoid this problem it is in this model assumed that if a link-disjoint primary and backup path pair is used, the availability requirement will be met, and an exact constraint for the availability requirement will not be needed, as seen in Zhang et al. (2003) and explained in Chapter 4. However, if a backup path is not allocated for a service, this constraint ensures that the primary path alone has an expected availability sufficient for the service. The first term makes the constraint redundant if a backup path is introduced for the service. Every link is only accounted for once in each constraint when using the u_{ijs} variables, as these variables only indicate the up direction of the path.

$$\sum_{p\in\mathscr{P}_s} M_s^A r_{sp} + \sum_{(i,j)\in\mathscr{A}} \ln(D_{ij}^L)\ u_{ijs} - \ln Y_s \geq 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.12}$$

The expected availability for every link as well as the availability requirement for each service will always be less than or equal to 1, thus the logarithms of these values will always be non-positive. M_s^A can then be defined as follows.

$$M_s^A = \ln Y_s - \sum_{(i,j)\in\mathscr{A}} \ln D_{ij}^L, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c$$

The following constraints require that the total bandwidth reserved for backup on an arc is at least as much as the largest of the individual bandwidth demands of the services that use the arc for backup in either the up or down direction, thus ensuring that the backup bandwidth capacity reserved will be able to support any of the backup requirements alone. This is referred to as the single backup bandwidth allocation in Chapter 4 and

seen in (Cui et al., 2002) and Józsa and Orincsay (2001).

$$B_s^U b_{ijs} + B_s^D b_{jis} - \lambda_{ij} \leq 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall (i,j) \in \mathscr{A} \tag{6.13}$$

To regulate the degree of sharing or dedication of backup capacity on the arcs in the network, the following constraints are used. A β value of zero will not restrict the number of services sharing a backup link at all, and a β equal to one is equivalent to using dedicated protection. If a backup link is highly shared it might be beneficial to force the reservation of more than the single backup path bandwidth allocation. In such cases β can be set to an appropriate value between zero and one, and this will force the reserved capacity to be at least a factor β of the sum of the bandwidth demands of the services using this backup arc. If there are many services using this arc, this will ensure that the arc can support more than one service in case of multiple failures. This method of using a factor regulating the amount of backup capacity to reserve for multiple overlapping backup paths has, to the authors knowledge, not been done previously, and the effect of it will be discussed in Chapter 7.

$$\sum_{c \in \mathscr{C}} \sum_{s \in \mathscr{S}_c} \beta (B_s^U b_{ijs} + B_s^D b_{jis}) - \lambda_{ij} \leq 0, \quad \forall (i,j) \in \mathscr{A} \tag{6.14}$$

In the LFM, it is assumed that having link-disjoint primary and backup paths for a service will fulfil any requirements this service has to availability, if a primary path alone is not sufficient. To ensure that the primary and backup paths allocated to a service are link-disjoint, the following constraints are used.

$$b_{ijs} + u_{ijs} \leq 1, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall (i,j) \in \mathscr{A} \tag{6.15}$$

In addition, the following set of constraints is needed to make sure that the backup paths created have the same destination as their service's primary path.

$$r_{sp} - x_{sp} \leq 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \forall p \in \mathscr{P}_s \tag{6.16}$$

The shared protection scheme requirement of only allowing backup paths of two services to overlap if their primary paths are disjoint, and vice versa, is incorporated in to this model. This will ensure that a link failure affecting two services' primary paths will not

lead to both their backup paths requiring the same backup resource. The following constraints regard overlap between primary and backup paths of services pairwise, ensuring that the primary and backup paths of two services cannot both be overlapping. To force the variable indicating overlapping primary paths for a combination of two services to 1 if the two services have primary paths with one or several overlapping links, the following constraints are used. As the constraints are defined for each link, the arc reservation variable, u_{ijs}, is included for both directions.

$$u_{ijs} + u_{jis} + u_{ijt} + u_{jit} - \ell_{st} \leq 1, \quad \forall (i,j) \in \mathscr{A} | i < j, \ \forall (s,t) \in \mathscr{O} \tag{6.17}$$

The following set of constraint ensures that backup paths of two services do not overlap if their primary paths are overlapping. As the above constraints, these constraints are only defined once for each link, and the b_{ijs} variables are therefore included for both directions.

$$b_{ijs} + b_{jis} + b_{ijt} + b_{jit} + \ell_{st} \leq 2, \quad \forall (i,j) \in \mathscr{A} | i < j, \ \forall (s,t) \in \mathscr{O} \tag{6.18}$$

The logical constraints in the link-flow model define y_c, x_{sp}, r_{sp}, u_{ijs}, b_{ijs} and ℓ_{st} as binary variables. The u_{ijs} and b_{ijs} variables are not defined for arcs going into the customer node corresponding to the customer demanding service s. The variables λ_{ij} are defined as non-negative continuous variable.

$$\begin{aligned}
y_c &\in \{0,1\}, \quad \forall c \in \mathscr{C} \\
x_{sp} &\in \{0,1\}, \quad \forall c \in \mathscr{C}, \ \forall s \in \mathscr{S}_c, \ \forall p \in \mathscr{P}_s \\
r_{sp} &\in \{0,1\}, \quad \forall c \in \mathscr{C}, \ \forall s \in \mathscr{S}_c, \ \forall p \in \mathscr{P}_s \\
u_{ijs} &\in \{0,1\}, \quad \forall c \in \mathscr{C}, \ \forall s \in \mathscr{S}_c, \ \forall (i,j) \in \mathscr{A} \mid j \neq V_c \\
b_{ijs} &\in \{0,1\}, \quad \forall c \in \mathscr{C}, \ \forall s \in \mathscr{S}_c, \ \forall (i,j) \in \mathscr{A} \mid j \neq V_c \\
\ell_{st} &\in \{0,1\}, \quad \forall (s,t) \in \mathscr{O} \\
\lambda_{ij} &\geq 0, \quad \forall (i,j) \in \mathscr{A}
\end{aligned} \tag{6.19}$$

6.3 The Path-Flow Model

The second MIP model for the CSBQANR problem presented is a PFM, where a primary path and possibly a backup path are allocated to each service being served. Where the LFM had no possibility of modelling the joint availability of primary and backup paths, the PFM provides the possibility to approximate these values. A PFM also has the potential to be able to solve larger instances within a reasonable amount of time compared to an equivalent LFM. However, the enhanced modelling accuracy incorporated in this model could leave the PFM slower than the LFM with similar input data. The downside of a potential increase in solution time must be considered against the upside of an increase in quality of the solutions derived from the PFM.

The links used in each path in the following model are the same from customer to provider and in the other direction, that is, a service is routed through the same but reversed path up and down through the network. The input to this model are sets of latency valid paths specific for each service provider pair with corresponding values for each path's total costs, availability, and bandwidth capacity usage on each arc. This input is provided by a pre-generation algorithm presented in Section 6.3.1.

This section opens with a presentation of the PFM's indices (Table 6.5), sets (Table 6.6), parameters (Table 6.7) and variables (Table 6.8). Following this, the the objective function and constraints are presented, finishing with the logical constraints. Lastly, a path pre-generation algorithm is presented in Section 6.3.1.

Table 6.5: Summary of the indices to the Path-Flow Model

Index	Definition
c	Customer
i, j	Network node
k, b	Path
p	Provider
s, t	Service

TABLE 6.6: Summary of the sets to the Path-Flow Model

Set	Definition
$\mathscr{A}$	Set of all arcs (i, j)
$\mathscr{C}$	Set of all customers c
$\mathscr{K}$	Set of all paths k
$\mathscr{K}_{sp}^{SP}$	Set of all paths k where service s is placed at provider p
$\mathscr{L}_{ij}$	Set of all paths k using the arc (i, j)
$\mathscr{O}$	Set with every combination of different services (s, t) where $s < t$
$\mathscr{P}_s$	Set of all providers p who can host service s
$\mathscr{S}_c$	Set of services s demanded by customer c

The nodes defining the arcs in $\mathscr{A}$ are numbered increasingly, so when a reference to links is needed, a link being the undirected link between two nodes or both arcs connecting the nodes, this is defined as $\forall(i, j) \in \mathscr{A}|i < j$. Because paths in this model use the same links in both directions, the set $\mathscr{L}_{ij}$ is symmetric, i.e. $\mathscr{L}_{ij} = \mathscr{L}_{ji}$. Additionally, the services are ordered, so when requiring all pairs of different services $\mathscr{O}$, the set of all combinations of services (s, t) where $s < t$ is used.

TABLE 6.7: Summary of the parameters to the Path-Flow Model

Parameter	Definition
D_k^P	Availability for path k
D_{kb}^C	Probability of both primary path k and backup path b being available simultaneously
E_{ij}	The cost of reserving a unit of capacity for backup on arc (i, j)
E_k^P	The cost of using a path k as primary path
I_{ij}	The amount of capacity available for reservation on arc (i,j)
R_c	Revenue from serving customer c
U_{ijk}^P	The amount of capacity used by path k on arc (i,j)
Y_s	Required minimum availability level for service s
β	Factor regulating the amount of extra capacity that must be reserved for an increased number of services sharing a backup arc

TABLE 6.8: Summary of the variables to the Path-Flow Model

Variable	Definition
f_{ijs}	Binary variable indicating if there is a need for backup capacity reservation on arc (i, j) for service s
ℓ_{st}	Binary variable indicating if the chosen primary paths of services s and t overlap anywhere
o_{kb}	Binary variable indicating if the primary and backup path combination (k, b) is chosen
q_{ijs}	Positive continuous variable displaying the amount of capacity needed to be reserved for backup on arc (i, j) for service s. Equals zero if f_{ijs} is zero
u_k	Binary variable indicating if path k is chosen as a primary path
v_k	Binary variable indicating if path k is chosen as a backup path
x_{sp}	Binary variable indicating if service s is provided by provider p
y_c	Binary variable indicating if customer c is being served
λ_{ij}	Positive continuous variable displaying the amount of capacity reserved on arc (i, j)

The problem is defined as a maximisation problem, and the objective function therefore models the BCs profits. The first term of the objective function sums up the revenue generated from the served customers, the second term subtracts the cost of using the selected primary paths and the third term subtracts the cost of reserving capacity for backup on each arc.

$$\max z \quad = \quad \sum_{c \in \mathscr{C}} R_c y_c - \sum_{k \in \mathscr{K}} E_k^P u_k - \sum_{(i,j) \in \mathscr{A}} E_{ij} \lambda_{ij} \tag{6.20}$$

The requirement that the whole bundle of services of a customer must be served for the customer to generate revenue is the same as in the LFM and is assured by the following set of constraints:

$$y_c - \sum_{p \in \mathscr{P}_s} x_{sp} = 0, \quad \forall c \in \mathscr{C}, \ \forall s \in \mathscr{S}_c \tag{6.21}$$

The following constraints ensure that a primary path, connecting a service's customer and provider, must be chosen for every served service.

$$x_{sp} - \sum_{k \in \mathscr{K}_{sp}^{SP}} u_k = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s \tag{6.22}$$

To ensure that the reserved capacity of an arc does not exceed the arc's available capacity, the following constraints are used. The sum of every chosen primary path's required bandwidth capacity on an arc plus the capacity reserved for backup paths must not exceed the arc's bandwidth capacity.

$$\sum_{k \in \mathscr{L}_{ij}} U_{ijk}^{P} u_k + \lambda_{ij} \leq I_{ij}, \quad \forall (i,j) \in \mathscr{A} \tag{6.23}$$

The issue with the non-linear availability equations is in this model solved by allowing the path pre-generation to calculate the expected availability for each path and the expected availability of every possible combination of primary and backup paths, i.e. the probability that both the primary and backup paths are fault free at the same time, and using these values in a linear expression. The linear expression is based on the probability expression presented in Section 2.3.3: $P(A \cap B) = P(A) + P(B) - P(A)P(B|A)$. The following constraints ensure that the primary or the combination of a primary and backup path chosen for a service provides an availability level that satisfies the customer's availability limit for each service. This set of constraints also force the reservation of backup for those primary paths that does not by themselves have a satisfactory availability level.

$$\sum_{p \in \mathscr{P}_s} \sum_{k \in \mathscr{K}_{sp}^{SP}} D_k^P (u_k + v_k) - \sum_{p \in \mathscr{P}_s} \sum_{k \in \mathscr{K}_{sp}^{SP}} \sum_{b \in \mathscr{K}_{sp}^{SP}} D_{kb}^C o_{kb} - Y_s y_c \geq 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.24}$$

To ensure that the variable indicating that there is a backup reservation need on an arc for a service equals 1 if there is reserved backup capacity for a service on that arc, the following constraints are used.

$$q_{ijs} - M_{ijs}^{B} f_{ijs} \leq 0, \quad \forall (i,j) \in \mathscr{A},\ \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.25}$$

The maximum value for the continuous backup reservation variable is the greatest of the

individual chosen path's bandwidth capacity requirement on that arc. M^B_{ijs} can then be defined as follows.

$$M^B_{ijs} \geq U^P_{ijk}, \quad \forall (i,j) \in \mathscr{A},\ \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s,\ \forall k \in \mathscr{K}^{SP}_{sp} \cap \mathscr{L}_{ij}$$

The amount of backup capacity needed to be reserved on an arc for a service is set by the following constraints. If a service has primary and backup paths both using the same arc, the maximum of zero and the difference between backup and primary capacity demand is used, since in the case where the backup path takes over the primary flow, any resources used by the primary path will be made available. This difference can only be different when the primary and backup paths overlap if they use the arc for opposite routing directions (customer to placement and placement to customer), as services can specify different bandwidth flow requirements in each direction. As an example take the case where a service has the demands 10 and 20 in its up and down directions. Assume its primary path uses an arc (i,j) in its up direction and the backup path uses the same arc in its down direction. The difference to be reserved for backup on (i,j) will then be 10, since the primary path will require 10 and the backup path will require 20. For the opposite arc (j,i) the routing directions will be reversed, and the difference to be reserved for backup will be -10, and there will therefore be no need for backup reservation. Without overlap between primary and backup paths, the amount of bandwidth capacity required for backup is simply the capacity demanded by the chosen backup path.

$$\sum_{p \in \mathscr{P}_s} \sum_{k \in \mathscr{K}^{SP}_{sp} \cap \mathscr{L}_{ij}} U^P_{ijs}(v_k - u_k) - q_{ijs} \leq 0, \quad \forall (i,j) \in \mathscr{A},\ \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.26}$$

The following constraints set the amount of backup capacity to be reserved on an arc to be at least as much as the maximum of the individual backup capacity requirements of the services using the arc for backup, referred to as the single backup bandwidth allocation in Chapter 4 and seen in (Cui et al., 2002) and Józsa and Orincsay (2001).

$$q_{ijs} - \lambda_{ij} \leq 0, \quad \forall (i,j) \in \mathscr{A},\ \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.27}$$

The effect of decreasing availability with the increase of the numbers of services sharing backup capacity on a link (see Section 3.8) is in some part captured by the following

constraints, which are equivalent to the Equation 6.14 in LFM. These are as in the LFM used to regulate the degree of sharing or dedication of backup on the links in the network. A β value of zero will not restrict the number of services sharing a backup link at all, and a β equal to one is equivalent to using dedicated protection. If a backup link is highly shared it might be beneficial to force the reservation of more than the single backup path bandwidth allocation (see Section 3.8). In such cases β can be set to an appropriate value between zero and one, and this will force the reserved capacity to be at least a factor β of the sum of the bandwidth demands of the services using this backup arc. If there are many services using this arc, this will ensure that the arc can support more than one service in case of multiple failures. This method of using a factor regulating the amount of backup capacity to reserve for multiple overlapping backup paths has, to the authors knowledge, not been done previously, and the effect of it will be discussed in Chapter 7.

$$\beta \sum_{c \in \mathscr{C}} \sum_{s \in \mathscr{S}_c} q_{ijs} - \lambda_{ij} \leq 0, \quad \forall (i,j) \in \mathscr{A} \tag{6.28}$$

To set the variable indicating whether or not a combination of paths are used as primary and backup paths the following constraints are used. These constraints ensure that the combination variable is set to 1 whenever two paths are chosen as backup and primary paths for a service and to 0 otherwise.

$$u_k + v_b - o_{kb} \leq 1, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s,\ \forall k \in \mathscr{K}_{sp}^{SP},\ \forall b \in \mathscr{K}_{sp}^{SP} \tag{6.29}$$

The constraint set of Equation 6.24 uses the combined availability of a primary and potential backup path to determine if the availability requirement is met, and handles link overlap between the two paths. The assumption that link-disjoint primary and backup paths will fulfil the availability requirement, used in the LFM, is not required in the PFM, and the link-disjoint requirement to a service's primary and backup path is therefore removed.

The shared path protection requirement of only allowing backup paths of two services to overlap if their primary paths are disjoint, and vice versa, is incorporated in to the PFM. This will ensure that a link failure affecting two services' primary paths will not lead to both their backup paths requiring the same backup resource. The following constraints

regard overlap between primary and backup paths of services pairwise, ensuring that the primary and backup paths of two services cannot both be overlapping. To force the variable indicating overlapping primary paths for a combination of two services to 1 if the two services have primary paths with one or several overlapping links, the following constraints are used. These constraints are link-based because of the fact that a link failure leads to both arcs representing the physical link to go down.

$$\sum_{p\in\mathscr{P}_s}\sum_{k\in\mathscr{K}_{sp}^{SP}\cap\mathscr{L}_{ij}} u_k + \sum_{p\in\mathscr{P}_s}\sum_{k\in\mathscr{K}_{tp}^{SP}\cap\mathscr{L}_{ij}} u_k - \ell_{st} \leq 1, \quad \forall(i,j)\in\mathscr{A}|i<j,\ \forall(s,t)\in\mathscr{O} \tag{6.30}$$

The overlap between primary paths is link-based, but since the backup bandwidth reservation is separate for the two arcs constituting a link, and a backup path may not need bandwidth capacity on both arcs when overlapping with its primary path, the backup overlap constraints are arc-based.

$$f_{ijs} + f_{ijt} + \ell_{st} \leq 2, \quad \forall(i,j)\in\mathscr{A},\ \forall(s,t)\in\mathscr{O} \tag{6.31}$$

The following constraints are logical constraints for the variables used in this model. The variables y_c, x_{sp}, u_k, v_b, o_{kb}, ℓ_{st} and f_{ijs} are binary, q_{ijs} and λ_{ij} are continuous variables greater than or equal to 0.

$$\begin{aligned}
y_c &\in \{0,1\}, \quad \forall c\in\mathscr{C}\\
x_{sp} &\in \{0,1\}, \quad \forall c\in\mathscr{C},\ \forall s\in\mathscr{S}_c,\ \forall p\in\mathscr{P}_s\\
u_k &\in \{0,1\}, \quad \forall k\in\mathscr{K}\\
v_k &\in \{0,1\}, \quad \forall k\in\mathscr{K}\\
o_{kb} &\in \{0,1\}, \quad \forall c\in\mathscr{C},\ \forall s\in\mathscr{S}_c,\ \forall p\in\mathscr{P}_s,\ \forall k\in\mathscr{K}_{sp}^{SP},\ \forall b\in\mathscr{K}_{sp}^{SP}\\
\ell_{st} &\in \{0,1\}, \quad \forall(s,t)\in\mathscr{O}\\
f_{ijs} &\in \{0,1\}, \quad \forall(i,j)\in\mathscr{A},\ \forall c\in\mathscr{C},\ \forall s\in\mathscr{S}_c\\
q_{ijs} &\geq 0, \quad \forall(i,j)\in\mathscr{A},\ \forall c\in\mathscr{C},\ \forall s\in\mathscr{S}_c\\
\lambda_{ij} &\geq 0, \quad \forall(i,j)\in\mathscr{A}
\end{aligned} \tag{6.32}$$

6.3.1 Path pre-generation

The PFM presented in this section assumes a set of paths with valid latency levels for each pair of service and eligible providers for that service. These paths include both the path from the customer node to the provider node, as well as the return path from the provider node to the customer node, as they are restricted to follow the same links in the network in both directions. Algorithm 1 is the pseudo code representation of the algorithm described in this section. Some of the parameters and sets used in the description of this pre-generation method are taken from the LFM (Section 6.2).

A complete set of these valid paths can be pre-generated by an algorithm performed for each possible service provider pair (s, p). This proposed algorithm starts at the customer node of service s and performs breadth first expansions along the links of the network. The total latency is accumulated for each path by adding up the latency of both arcs of the link used in the path expansion. The pre-generating algorithm will keep performing expansions as long as there are unexpanded paths not ending in the destination provider node J_p, having links (i, j) from its current end node i, with sufficient bandwidth capacity I_{ij}, where j is a not an already visited node, and the resulting latency from the expansion does not exceed the latency requirement of service s, G_s. The output of this algorithm is the set of all the created paths having the destination provider node J_p as it final node, and forms the basis for the paths to add to the PFM.

Equation 2.1 is used to calculate the availability level of each generated path, D_k^P. Only link failures are considered, meaning that the two arcs of a link have the same expected availability where both share the same availability state. As a result, the set of network components used in the calculation is the set of network links, each link represented by only one of its arcs. D_{kb}^C, the probability of two paths k and b both being available simultaneously, is pre-calculated as the third term of Equation 2.2, for every pair of two paths connecting a customer and a provider for a service.

A path k's cost E_k^P includes all costs of bandwidth usage by the path as well as the cost of placing a service s at the provider p at the path k's destination, H_{sp}. The cost from bandwidth usage is calculated by adding the arc bandwidth cost, E_{ij}, multiplied

by the bandwidth usage on that arc, U^P_{ijk}, for all arcs in both directions of path k. The bandwidth usage of a path k on an arc (i, j), U^P_{ijk}, is set to the service's bandwidth requirements B^U_s and B^D_s for any arc used in the from customer to provider and provider to customer, respectively. In addition, the model uses a set of paths for each arc (i, j) containing all paths using that arc, $\mathscr{L}_{ij} = \{k \in \mathscr{K} \mid U^P_{ijk} > 0\}$.

Algorithm 1 Path Generation

procedure PATHGENERATION(s, p) ▷ Service s and provider p
  setup:
  $a \leftarrow V_{customerFor(s)}$ ▷ Start node for up direction of path
  $t \leftarrow J_p$ ▷ Destination node for up direction of path
  $l \leftarrow$ *initial path label* ▷ Path labels: path objects with added information
  $l.latency \leftarrow 0$
  $l.visited \leftarrow \{a\}$
  $l.end \leftarrow a$
  $l.upArcs \leftarrow \{\}$
  $unfinished \leftarrow \{l\}$ ▷ Unexpanded path labels
  $finished \leftarrow \{\}$ ▷ Valid paths to destination node

  work:
  while $unfinished \neq \varnothing$ **do**
    $l \leftarrow$ *path label removed from unfinished* ▷ For breadth first, remove oldest
    if $l.end = t$ **then**
      $k \leftarrow new\ path(l.upArcs)$
      $finished \leftarrow finished + \{k\}$
    else
      for all $(i, j) \in \mathscr{A} \mid i = l.end$ **do**
        if $j \notin l.visited \wedge l.latency + T_{ij} + T_{ji} \leq G_s \wedge B^U_s \leq I_{ij} \wedge B^D_s \leq I_{ji}$ **then**
          $l' \leftarrow copy(l)$
          $l'.upArcs \leftarrow l'.upArcs + \{(i, j)\}$
          $l'.visited \leftarrow l'.visited + \{j\}$
          $l'.end \leftarrow j$
          $l'.latency \leftarrow l'.latency + T_{ij} + T_{ji}$
          $unfinished \leftarrow unfinished + \{l'\}$
  return *finished*

In some cases the number of possible paths to add to the model may be immensely large. For these cases it may be beneficial to limit the number of paths generated, simplifying the problem but only providing a heuristic solution. By only including a subset of all the possible paths, there is no guarantee of the optimal paths being included. The subset of paths can be selected by introducing an upper limit to the number of paths for each combination of service and provider, terminating the breadth

first expansion once this limit has been reached for a combination. This will result in a generated set of the shortest paths with respect to the number of links used.

A lower number of arcs in a path will, if costs and availability are in the same order of magnitude, in general have a lower accumulated cost and provide a better availability level than paths with many arcs. It is therefore reasonable to assume that these shorter paths will give an input set that is more likely to include the optimal paths than sets of the same size of paths generated at random or by other simple means of selection.

6.4 The Mapping Model

This section presents a MIP model to the CSBQANR problem providing another level of abstraction compared to the PFM, taking pre-generated sets of possible complete routings for each service as input, these routing as referred to as mappings. A mapping consists of the placement of a service and a network routing of the service that satisfies its requirements. The network routing consists of a primary path or the combination of a primary and backup path between the customer and provider node. The mapping model provides another level of abstraction above the PFM and could have the potential to solve even larger problem instances. The enhanced level of abstraction results in a model more suited to be used in combination with a column generation algorithm.

The input to this model is a set of mappings specific to each service, which all satisfies the availability limits given by the customers for each service. Mapping costs, revenues, routing information, backup path capacity needs and primary path capacity usage are also provided as input to this model. The input is created by a pre-generating algorithm presented in Section 6.4.1 or by column generating heuristics presented in Section 6.5.

First to be presented in this section are the indices (Table 6.9), sets (Table 6.10), parameters (Table 6.11) and variables (Table 6.12) used in this mapping model. Following this, the objective function and the constraints are presented, finishing with the logical constraints. Lastly, a mapping pre-generating method for providing the model with input is presented in Section 6.4.1.

Table 6.9: Summary of the indices to the mapping model

Index	Definition
c	Customer
i, j	Network node
m	Mapping
k, b	Path
s, t	Service

TABLE 6.10: Summary of the sets to the mapping model

Set	Definition
$\mathscr{A}$	Set of all arcs (i, j)
$\mathscr{C}$	Set of all customers c
$\mathscr{K}$	Set of all paths k
$\mathscr{L}_{ij}$	Set of all paths using the arc (i, j)
$\mathscr{M}$	Set of all mappings m
$\mathscr{M}_k^B$	Set of all mappings m using path k as backup path
$\mathscr{M}_k^P$	Set of all mappings m using path k as primary path
$\mathscr{M}_s^S$	Set of all mappings m that can be used by a service s
$\mathscr{O}$	Set with combination of all different services (s, t) where $s < t$
$\mathscr{S}_c$	Set of services demanded by customer c

The nodes defining the arcs in $\mathscr{A}$ are numbered increasingly, so when links are needed, a link being the undirected link between two nodes or both arcs connecting the nodes, this is defined as $\forall(i, j) \in \mathscr{A} | i < j$. Because paths in this model use the same links in both directions, the set $\mathscr{L}_{ij}$ is symmetric, i.e. $\mathscr{L}_{ij} = \mathscr{L}_{ji}$. Additionally, the services are ordered, so when requiring all pairs of different services $\mathscr{O}$, the set of all combinations of services (s, t) where $s < t$ is used.

TABLE 6.11: Summary of the parameters to the mapping model

Parameter	Definition
E_{ij}	The cost of reserving a unit of capacity for backup on arc (i, j)
E_k^P	The cost of using a path k
F_{ijm}	Indicates whether mapping m needs backup capacity on arc (i,j) or not. Equal to 1 if Q_{ijm} is larger than 0, and 0 otherwise
I_{ij}	The amount of capacity available for reservation on arc (i,j)
Q_{ijm}	The amount of capacity needed to be reserved for backup on arc (i,j) by mapping m. Equal to the maximum of zero and the difference between backup and primary capacity demand if backup and primary paths overlap on (i,j)
R_c	Revenue from serving customer c
U_{ijm}^M	The amount of capacity used by mapping m on arc (i,j)
β	Factor regulating the amount of extra capacity that must be reserved for an increased number of services sharing a backup arc

TABLE 6.12: Summary of the variables to the mapping model

Variable	Definition
ℓ_{st}	Binary variable indicating if the chosen primary paths of services s and t overlap anywhere
w_m	Binary variable indicating if mapping m is chosen
y_c	Binary variable indicating if customer c is being served
λ_{ij}	Positive continuous variable showing the amount of capacity reserved on arc (ij)

The problem is defined as a maximisation problem, and the objective function therefore models the BCs profits. The first term of the objective function sums up the revenue generated from the served customers, the second term subtracts the cost of using the primary paths of the selected mappings and the third term subtracts the cost of reserving capacity for backup on each arc.

$$\max z \quad = \quad \sum_{c \in \mathscr{C}} R_c y_c - \sum_{k \in \mathscr{K}} E_k^P \sum_{m \in \mathscr{M}_k^P} w_m - \sum_{(i,j) \in \mathscr{A}} E_{ij} \lambda_{ij} \tag{6.33}$$

The following constraints ensure that one of the available mappings for a service must be chosen for each service of a customer that is being served.

$$y_c - \sum_{m \in \mathscr{M}_s^S} w_m = 0, \quad \forall c \in \mathscr{C}, \ \forall s \in \mathscr{S}_c \tag{6.34}$$

To ensure that the sum of the capacity reserved for all primary paths of the chosen mappings and the amount of capacity reserved for backup use on an arc do not exceed the available capacity of that arc, the following constraints are used.

$$\sum_{m \in \mathscr{M}_s^S} U_{ijm}^M w_m + \lambda_{ij} \leq I_{ij}, \quad \forall (i,j) \in \mathscr{A} \tag{6.35}$$

The following constraints make sure that the amount of capacity reserved on an arc for backup is at least as big as every individual bandwidth requirement of the services using this arc for backup, or effectively at least as large as the largest bandwidth requirement among these. This is referred to as the single backup bandwidth allocation in Chapter 4 and seen in (Cui et al., 2002) and Józsa and Orincsay (2001).

$$\sum_{m \in \mathscr{M}_s^S} Q_{ijm} w_m - \lambda_{ij} \leq 0, \quad \forall (i,j) \in \mathscr{A}, \ \forall c \in \mathscr{C}, \ \forall s \in \mathscr{S}_c \tag{6.36}$$

The effect of decreasing expected availability with the increase of the numbers of services sharing backup capacity on an arc (see Section 3.8) is in some part captured by the following constraints, which are equivalent to the Equation 6.14 in the LFM and Equation 6.28 in the PFM. These are as in the LFM and PFM used to regulate the degree of sharing or dedication of backup on the arcs in the network. A β value of zero will not restrict the number of services sharing a backup arc at all, and a β equal to one is equivalent to using dedicated protection. If a backup arc is highly shared it might

be beneficial to force the reservation of more than the single backup path bandwidth allocation (see Section 3.8). In such cases β can be set to an appropriate value between zero and one, and this will force the reserved capacity to be at least a factor β of the sum of the bandwidth demands of the services using this backup arc. If there are many services using this arc, this will ensure that the arc can support more than one service in case of multiple failing primary paths. This method of using a factor regulating the amount of backup capacity to reserve has not yet been seen in literature, and the effect of it will be discussed in Chapter 7.

$$\beta \sum_{m \in \mathscr{M}} Q_{ijm} w_m - \lambda_{ij} \leq 0, \quad \forall (i,j) \in \mathscr{A} \tag{6.37}$$

All mappings taken as input to this model already includes a valid combination of primary and backup paths, and the link-disjoint requirement to the primary and backup paths used in the LFM is not included in the mapping model.

The shared path protection requirement of only allowing backup paths of two services to overlap if their primary paths are disjoint, and vice versa, is incorporated in to the mapping model. This will ensure that a link failure affecting two services' primary paths will not lead to both their backup paths requiring the same backup resource. The following constraints regard overlap between primary and backup paths of services pairwise, ensuring that the primary and backup paths of two services cannot both be overlapping. To force the variable indicating overlapping primary paths for a combination of two services to 1 if the two services have primary paths with one or several overlapping links, the following constraints are used. These constraints are link-based because of the fact that a link failure leads to both arcs representing the physical link to go down.

$$\sum_{k \in \mathscr{L}_{ij}} \sum_{m \in \mathscr{M}_k^P \cap \mathscr{M}_s^S} w_m + \sum_{k \in \mathscr{L}_{ij}} \sum_{m \in \mathscr{M}_k^P \cap \mathscr{M}_t^S} w_m - \ell_{st} \leq 1, \quad \forall (i,j) \in \mathscr{A} | i < j, \ \forall (s,t) \in \mathscr{O} \tag{6.38}$$

The overlap between primary paths is link-based, but since the backup bandwidth reservation is separate for the two arcs constituting a link, and a backup path of a mapping may not need bandwidth capacity on both arcs, the backup overlap constraints are

arc-based.

$$\sum_{k\in\mathscr{L}_{ij}}\sum_{m\in\mathscr{M}_k^B\cap\mathscr{M}_s^S} F_{ijm}w_m + \sum_{k\in\mathscr{L}_{ij}}\sum_{m\in\mathscr{M}_k^B\cap\mathscr{M}_t^S} F_{ijm}w_m + \ell_{st} \leq 2, \quad \forall(i,j)\in\mathscr{A},\ \forall(s,t)\in\mathscr{O} \tag{6.39}$$

The following constraints are logical constraints for the variables used in this model. The variables y_c, w_m and ℓ_{st} are binary, and λ_{ij} are continuous variables greater than or equal to 0.

$$\begin{aligned} y_c &\in \{0,1\}, \quad \forall c\in\mathscr{C} \\ w_m &\in \{0,1\}, \quad \forall m\in\mathscr{M} \\ \ell_{st} &\in \{0,1\}, \quad \forall(s,t)\in\mathscr{O} \\ \lambda_{ij} &\geq 0, \quad \forall(i,j)\in\mathscr{A} \end{aligned} \tag{6.40}$$

6.4.1 Mapping pre-generation

The MIP-model introduced in this section is dependent on a set of pre-generated mappings for every service; combinations of provider placement, primary network path as well as backup path if needed. Since a mapping also contains the provider placement decision, only one set of mappings is needed per service ($\mathscr{M}_s^S$), compared to one set of paths for every service provider pair, needed in the PFM from Section 6.3.

The pre-generation of mappings takes as input sets of pre-generated paths for each service provider pair, described in Section 6.3.1. The set of mappings for each service is then generated from these pre-generated paths.

Each set of mappings is defined for a service, but the generation of mappings is done separately for each service provider pair, adding the resulting mappings to that service's set of mappings. For each such service provider pair, every path for that pair is considered. If a path alone is sufficient to fulfil a service's requirements, a mapping is created using the provider from the current service provider pair, the path found as the primary path, and no backup path. If the path alone is not sufficient for the service's requirements, combinations with all the other paths of the same service provider pair are tested, using the first path as the primary path, and the other paths as backup paths. For every valid combination found, a mapping for the service is added using the current provider and the combination of primary and backup path. A pseudo code representation of the mapping pre-generation is shown in Algorithm 2.

The cost of a mapping is simply the cost of its primary path k; E_k^P. This cost includes both the cost from bandwidth usage as well as the provider placement cost of the mapping's service, as described in Section 6.3.1. Costs from backup paths are not included in the mappings' costs, as these are included in the global backup bandwidth reservation costs found in the mapping model of Section 6.4.

The amount of capacity used by a mapping m on every arc (i, j), U_{ijm}^M, is equal to its primary path k's bandwidth usage, U_{ijk}^P. The amount of capacity needed to be reserved

Algorithm 2 Mapping Generation

procedure MappingGeneration(s, p) ▷ Service s and provider p
  $\mathscr{M}_s^S \leftarrow$ *Set of mappings for service* s
  $\mathscr{K}_{sp}^{SP} \leftarrow$ *Set of all paths for service* s *to provider* p

  for all $k \in \mathscr{K}_{sp}^{SP}$ **do**
    if $D_k^P > Y_s$ **then** ▷ Path availability D_k^P from Equation 2.1
      $m \leftarrow$ *new mapping*(k) ▷ Mapping with primary only
      $\mathscr{M}_s^S \leftarrow \mathscr{M}_s^S + \{m\}$
    else
      for all $b \in \mathscr{K}_{sp}^{SP} | k \neq b$ **do**
        if $D_{kb}^M \geq Y_s$ **then** ▷ Mapping availability D_{kb}^M from Equation 2.2
          $m \leftarrow$ *new mapping*(k, b) ▷ Mapping with primary and backup
          $\mathscr{M}_s^S \leftarrow \mathscr{M}_s^S + \{m\}$
  return $\mathscr{M}_s^S$

for backup on arc (i, j) by mapping m, Q_{ijm}, is the maximum of 0 and the bandwidth demand for a mapping's backup path b, U_{ijb}^P, minus the bandwidth demand of the primary path k's demand, U_{ijk}^P. This is done to avoid redundant backup reservations where the primary and backup paths of a service overlap, as a service will never need its primary and backup paths simultaneously. F_{ijm} indicates whether mapping m needs backup capacity on (i, j) or not. It is set to be 1 if Q_{ijm} is larger than zero, and zero otherwise.

6.5 Column Generation

This section introduces three heuristic approaches to solving the CSBQANR problem, using column generation to add feasible mappings to the mapping model presented in Section 6.4, as opposed to including the complete set of valid pre-generated mappings. The column generation is performed for the LP-relaxation of the model only, followed by solving the MIP-version of the model with the generated mappings, thus only guaranteeing a heuristic solution to the problem. The approach described in this section can be extended to provide better or optimal solutions, depending on the column generation algorithm used, by incorporating the column generation into the branching tree of the MIP-solution process, this is left for further work (see Section 8.2) and is not pursued further in this thesis.

These column generation approaches are introduced both to reduce the dependency on pre-generation, as well as to provide good solutions in shorter time frames compared to the complete mapping model using pre-generation and enabling the solution of larger problem instances. Only a small subset of possible mappings will be included in the model by the column generation step, reducing the size of the problem when solving the MIP-version, and potentially making the solution process simpler and faster. The assumption that good columns in the LP-relaxation are likely to also be good in the MIP-version is made, thus good solutions is expected even though the solution space is greatly reduced.

The column generation methods use dual variables obtained from running the LP-relaxation of the model. A summary of the dual variables can be found in Table 6.13. These dual variables are used to guide the creation of, and evaluate, potential mappings. Three column generation approaches are suggested, the first based on partial pre-generation (paths for service-provider pairs), followed by two approaches using no pre-generation, generating the primary path and backup path for each new mapping as needed.

TABLE 6.13: Summary of the dual variables obtained from the model in Section 6.4

Dual Variables	Originating Constraint Set	Range	Domain
α_s	Equation 6.34	$free$	$\forall c \in \mathscr{C}, \forall s \in \mathscr{S}_c$
γ_{ij}	Equation 6.35	≥ 0	$\forall (i,j) \in \mathscr{A}$
ϕ_{ijs}	Equation 6.36	≥ 0	$\forall (i,j) \in \mathscr{A},\ \forall c \in \mathscr{C}, \forall s \in \mathscr{S}_c$
ω_{ij}	Equation 6.37	≥ 0	$\forall (i,j) \in \mathscr{A}$
θ_{ijst}	Equation 6.38	≥ 0	$\forall (i,j) \in \mathscr{A} \vert i < j,\ \forall (s,t) \in \mathscr{O}$
τ_{ijst}	Equation 6.39	≥ 0	$\forall (i,j) \in \mathscr{A},\ \forall (s,t) \in \mathscr{O}$

To evaluate a potential new mapping, the reduced cost of that mapping's respective w_m variable is calculated. Since the master problem is a maximisation problem, a new mapping with positive reduced cost is wanted. The generic formula for calculating the reduced cost vector, σ is $\sigma = \mathbf{c} - \mathbf{A}^T\mathbf{y}$, where $\mathbf{c}$ is the objective coefficient vector, $\mathbf{A}$ is the constraints coefficient matrix and $\mathbf{y}$ is the dual variable vector. In the context of generating columns, the reduced cost σ_m of a column representing a mapping m, belonging to service s and consisting of a primary path k and a potential backup path b, is calculated from the following equation.

$$\begin{aligned}
\sigma_m = & - E_k^P \\
& + \alpha_s \\
& - \sum_{(i,j)\in\mathscr{A}} U_{ijm}^M \gamma_{ij} \\
& - \sum_{(i,j)\in\mathscr{A}} Q_{ijm} \phi_{ijs} \\
& - \beta \sum_{(i,j)\in\mathscr{A}} Q_{ijm} \omega_{ij} \\
& - \sum_{(i,j)\in\mathscr{A}\vert i<j \wedge U_{ijm}^M > 0} \ \sum_{(s',t)\in\mathscr{O}\vert s'=s \vee t=s} \theta_{ijs't} \\
& - \sum_{(i,j)\in\mathscr{A}} \ \sum_{(s',t)\in\mathscr{O}\vert s'=s \vee t=s} F_{ijm} \tau_{ijs't}
\end{aligned} \tag{6.41}$$

Three different column generation approaches are proposed in this thesis. The first approach, presented in Section 6.5.1, adapts the mapping pre-generation algorithm from

Section 6.4.1 to a column generation setting, in essence using column generation to filter the large amount of mappings possible to pre-generate. The remaining two approaches both use a series of Shortest Path Problems with Resource Constraints (SPPRC) to find potential paths for new mappings, both presented in Section 6.5.2.

6.5.1 Column Generation by Brute Force Search

This column generation approach, referred to as the *Column Generation by Brute Force Search* method, is a simple adaptation of the mapping pre-generation approach from Section 6.4.1 to a column generation setting, and is as such dependent on pre-generated paths. The algorithm is executed for every service provider pair and consists of looping through all possible paths for that pair, trying to create feasible mappings, combining the current path with different backup paths as needed. The reduced cost σ_m of a feasible mapping m belonging to service s is then evaluated as in Equation 6.41. Any mapping m with a positive, non-zero reduced cost σ_m is considered beneficial to add to the model. The version of the algorithm returning the best mapping with a positive, non-zero reduced cost is described in Algorithm 3, but can easily be modified to add the first beneficial mapping or all of the beneficial mappings found.

Algorithm 3 Brute Force Column Generation

```
1:  procedure BruteForceColumnGeneration(s, p)   ▷ Service s and provider p
2:      m* ← none                                ▷ Best mapping found so far
3:      e* ← 0                                   ▷ Best evaluation found so far
4:      for all k ∈ K_sp^SP do
5:          if D_k^P ≥ Y_s then                  ▷ Path availability D_k^P from Equation 2.1
6:              m ← new mapping(k)               ▷ Mapping with primary only
7:              e ← σ_m                          ▷ Reduced cost from Equation 6.41
8:              if e > e* then
9:                  e* ← e
10:                 m* ← m
11:         else
12:             for all b ∈ K_sp^SP | k ≠ b do
13:                 if D_kb^M ≥ Y_s then         ▷ Mapping availability D_kb^M from Equation 2.2
14:                     m ← new mapping(k, b)    ▷ Mapping with primary and backup
15:                     e ← σ_m                  ▷ Reduced cost from Equation 6.41
16:                     if e > e* then
17:                         e* ← e
18:                         m* ← m
19:     return m*
```

6.5.2 SPPRC Column Generation

This section proposes two versions of a column generation heuristic that does not depend on pre-generation of paths, solving SPPRCs to find paths. These two methods are simply referred to as *SPPRC Column Generation Heuristic A* and *SPPRC Column Generation Heuristic B*. The SPPRC is an extension of the well-known shortest path problem, where the total usage of one or more resources along the path is constrained. SPPRCs can be solved by dynamic programming algorithms referred to as labelling algorithms. For a more extensive discussion of the SPPRC and labelling algorithms, see Irnich and Desaulniers (2005). The values of the dual variables are used to assign costs to the arcs in the network used by the SPPRCs. Note that this is a heuristic for generating new columns to the master problem, and thus does not guarantee an optimal solution even if incorporated in the MIP branching process.

From Equation 6.41, the equation for reduced cost, the only term that can have a positive contribution is the second term, α_s. Since the constraint in Equation 6.34 is an equality constraint, α_s has a free range, and can therefore potentially provide both a positive and negative contribution to the reduced cost. However, due to the structure of the problem, the $\leq$ part of the equality constraint is expected to be the binding one, opposed to the $\geq$ version. α_s will therefore have a positive value and thus also a positive contribution to the reduced cost. All the remaining dual variables, as well as the parameters associated with them will always be non-negative, and since they are subtracted when calculating the reduced cost, their contribution will always be non-positive.

Apart from the α_s dual variable and the service hosting cost H_{sp} included in the primary path cost E_k^P, all components of the reduced cost of a mapping are dependent on the arcs used for the primary path as well as potentially a backup path. These terms can be modelled as costs associated with selecting an arc (i, j) in either the primary path or backup path. Since all of these terms have a negative contribution to the reduced cost of a potential new mapping, these costs will always be non-negative. The goal is to find the primary path and backup path that each minimise the costs associated with the arcs used. In addition, the paths must not exceed the maximal allowed latency, in essence

creating a Shortest Path Problem with Resource Constraints (SPPRC) for each of the primary and backup paths. Note that the availability requirement will be dependent on both the choice of primary and backup path, and can therefore not be handled by the individual SPPRCs, and is instead handled externally to the SPPRCs in the two column generation heuristics presented.

The column generation is performed for each service provider pair (s, p), thus the α_s and the H_{sp} are considered as given. The remainder of the reduced cost calculation depends on the set of arcs selected for the mapping m's primary path k and backup path b in the up direction, referred to as A_k^{PATH} and A_b^{PATH}, respectively. Since paths are required to use the same links in both directions, costs for both directions can be inferred from the selection of arcs in one direction. The reduced cost of a mapping m, from Equation 6.41, with path k as primary path and path b as backup path, can thus be reformulated as follows:

$$\sigma_m = \alpha_s - H_{sp} - \sum_{(i,j)\in A_k^{PATH}} V_{ijs}^P - \sum_{(i,j)\in A_b^{PATH}} V_{ijsk}^B \tag{6.42}$$

Where V_{ijs}^P and V_{ijsk}^B are the costs for selecting an arc (i, j) for a service s in the primary path and backup path, respectively, in the up direction. Note that V_{ijsk}^B, is also indexed by the primary path k, is the costs associated with selecting arcs for the backup paths depends in the primary path. If an arc (i, j) is used in one direction, (i, j) is used in the opposite direction, therefore V_{ijs}^P and V_{ijs}^B includes both the cost for using (j, i) in the up direction and the cost for using (j, i) in the down direction.

The amount of primary bandwidth usage for a mapping m on an arc (i, j), U_{ijm}^M, is B_s^U if (i, j) is used in the up direction, and B_s^D if (i, j) is used in the down direction and 0 otherwise. V_{ijs}^P can therefore be expressed as:

$$\begin{aligned} V_{ijs}^P = \; & B_s^U E_{ij} + B_s^D E_{ji} \\ & + B_s^U \gamma_{ij} + B_s^D \gamma_{ji} \\ & + \sum_{(s',t)\in\mathscr{O}|s'=s\vee t=s} \Theta_{ijs't} \end{aligned} \tag{6.43}$$

Where

$$\Theta_{ijs't} = \begin{cases} \theta_{ijs't} & \text{if } i < j \\ \theta_{jis't} & \text{if } i > j \end{cases} \tag{6.44}$$

since the $\theta_{ijs't}$ is only defined for each link $(i,j) \in \mathscr{A} | i < j$, and not each arc.

The amount of backup bandwidth required by a mapping m on an arc (i,j), Q_{ijm}, depends on both the primary path k and the backup path b, as the primary path k's bandwidth usage on (i,j), U^P_{ijk}, affects the backup path's bandwidth need on (i,j) if they are overlapping. The cost of using an arc (i,j) in the backup path for a service s from customer to provider (and the opposite arc (j,i) from provider to customer), given path k as primary path, V^B_{ijsk}, is defined as:

$$\begin{aligned} V^B_{ijsk} = \quad & \max(0, B^U_s - U^P_{ijk})\phi_{ijs} + \max(0, B^D_s - U^P_{jik})\phi_{jis} \\ & + \beta \max(0, B^U_s - U^P_{ijk})\omega_{ij} + \beta \max(0, B^D_s - U^P_{jik})\omega_{ji} \\ & + \sum_{(s',t) \in \mathscr{O} | s' = s \vee t = s} (F^U_{jisk}\tau_{jis't} + F^D_{ijsk}\tau_{ijs't}) \end{aligned} \tag{6.45}$$

Where F^U_{ijsk} and F^D_{ijsk} is defined as:

$$F^U_{ijsk} = \begin{cases} 1 & \text{if } B^U_s - U^P_{ijk} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.46}$$

$$F^D_{ijsk} = \begin{cases} 1 & \text{if } B^D_s - U^P_{ijk} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.47}$$

The two column generation methods described here uses three different sub routines for solving different SPPRCs, referred to as SPPRC-L, SPPRC-LO and SPPRC-LA. The difference between these sub routines is the constrained resources of the SPPRC they solve. SPPRC-L and SPPRC-LO are used by *SPPRC Column Generation Heuristic A*, while *SPPRC Column Generation Heuristic B* is an extension of the former column generation method which introduces SPPRC-LA. For easy reference, these sub routines

and the constrained resources of the SPPRC they solve can be found in Table 6.14. Each of these sub routines, as well as the SPPRCs solved by them, is discussed further in the description of the column generation method using them.

TABLE 6.14: The resources of the SPPRCs solved by the different sub routines used by *SPPRC Column Generation Heuristic A* and *SPPRC Column Generation Heuristic B*

SPPRC Sub Routine	Resources
SPPRC-L	Latency
SPPRC-LO	Latency, Overlap
SPPRC-LA	Latency, Availability

SPPRC Column Generation Heuristic A

The column generation approach described here solves a series of SPPRCs to find a combination of primary- and backup path for a given service and provider placement. The bandwidth and latency requirements can be handled individually for the primary- and backup path. The algorithm starts by finding the cheapest primary path to the given provider placement, satisfying the requirements to bandwidth and latency. This is done by removing any arcs with insufficient bandwidth capacity and modelling the latency as a resource in an SPPRC. This SPPRC is solved by the sub routine referred to as SPPRC-L. If a primary path with sufficiently low costs is found, this path is either used alone in a new mapping, if its availability is sufficient for the requirement specified by the service; otherwise it is used as a basis for a primary and backup path combination.

If the primary path alone does not have a sufficient availability, a second type of SPPRC is used to find a suitable backup path, extending the original SPPRC with a resource indicating the number of link overlaps between the primary and backup paths. This SPPRC is solved by the SPPRC-LO sub routine. The expected availability is not explicitly calculated by this SPPRC, but by using the same assumption as in the LFM, where link-disjoint primary and backup paths will satisfy the availability requirement, this column generation method tries to guide the SPPRC towards availability feasible primary and backup paths by gradually restricting the two paths to share fewer links. The SPPRC-LO sub routine is performed with gradually fewer overlaps allowed until

the availability requirement is satisfied or until further restriction to overlap is no longer possible (the last backup path found leads to a non-profitable mapping or overlap is already fully disallowed). Pseudo code for this method is found in Algorithm 4.

Algorithm 4 SPPRC Column Generation Heuristic A

procedure SPPRC-CGA(s, p) ▷ Service s and provider p
  $a \leftarrow V_{customerFor(s)}$ ▷ Start node for up direction of path
  $t \leftarrow J_p$ ▷ End node for up direction of path
  $\mathscr{A}' \leftarrow \{(i,j) \in \mathscr{A} | I_{ij} \geq B_s^U \wedge I_{ji} \geq B_s^D\}$ ▷ Arcs usable for s in up direction

  $(k, e^P) \leftarrow$ *SPPRC-L*(
    $a,\ t,\ G_s,\ \mathscr{A}'$, ▷ Start, end, max latency, valid arcs
    $\{V_{ijs}^P \forall (i,j) \in \mathscr{A}'\}$, ▷ Costs of all arcs (i,j), from Eq. 6.43
    $\{T_{ij} + T_{ij} \forall (i,j) \in \mathscr{A}'\}$ ▷ Latency for all arcs (i,j) up
)
  if $\alpha_s - H_{sp} - e^P > 0$ **then** ▷ Reduced cost from Eq. 6.42
    if $k \neq \varnothing \wedge D_k^P > Y_s$ **then** ▷ Equation 2.1
      return *new mapping*(k) ▷ Mapping with primary only
    else
      for all $(i,j) \in \mathscr{A}'$ **do**
        if $(i,j) \in A_k^{PATH} \vee (j,i) \in A_k^{PATH}$ **then**
          $O_{ij} \leftarrow 1$ ▷ (i,j) in backup will create link overlap
        else
          $O_{ij} \leftarrow 0$ ▷ (i,j) in backup will not create link overlap
      $o \leftarrow |A_k^{PATH}| - 1$ ▷ # of allowed overlaps with k
      while $o \geq 0$ **do**
        $(b, e^B) \leftarrow$ *SPPRC-LO*(
          $a,\ t,\ G_s,\ o,\ \mathscr{A}'$, ▷ Start, end, max latency, max overlap, valid arcs
          $\{V_{ijsk}^P \forall (i,j) \in \mathscr{A}'\}$, ▷ Costs for all arcs (i,j), from Eq. 6.45
          $\{T_{ij} + T_{ij} \forall (i,j) \in \mathscr{A}'\}$, ▷ Latency for all arcs (i,j) up
          $\{O_{ij} \forall (i,j) \in \mathscr{A}'\}$ ▷ Overlap for all arcs (i,j) up
)
        if $b = \varnothing \vee \alpha_s - H_{sp} - e^P - e^B \leq 0$ **then** ▷ Reduced cost from Eq. 6.42
          break ▷ Profitable mapping not possible
        if $D_{kb}^M > Y_s$ **then**
          return *new mapping*(k, b) ▷ Mapping with primary and backup
        $o \leftarrow o - 1$
  return *none*

SPPRC Column Generation Heuristic B

For mappings only using a primary path, the availability can be modelled as a resource and added to the SPPRC where the accumulated availability must stay above an availability requirement. For some problem instances it may be likely that single

path mappings are sufficient to satisfy services' requirements. As a result, the following extension to version A of the SPPRC column generation heuristic is proposed, pseudo code presented in Algorithm 5.

Algorithm 5 SPPRC Column Generation Heuristic B

1: **procedure** SPPRC-CGB(s, p) ▷ Service s and provider p
2: $a \leftarrow V_{customerFor(s)}$ ▷ Start node for up direction of path
3: $t \leftarrow J_p$ ▷ End node for up direction of path
4: $\mathscr{A}' \leftarrow \{(i,j) \in \mathscr{A} | I_{ij} \geq B_s^U \wedge I_{ji} \geq B_s^D\}$ ▷ Arcs usable for s in up direction
5:
6: $(k, e^P) \leftarrow$ *SPPRC-LA*(
7: $a,\ t,\ G_s,\ Y_s,\ \mathscr{A}'$, ▷ Start, end, max latency, min availability, valid arcs
8: $\{V_{ijs}^P \forall (i,j) \in \mathscr{A}'\}$, ▷ Costs of all arcs (i,j), from Eq. 6.43
9: $\{T_{ij} + T_{ij} \forall (i,j) \in \mathscr{A}'\}$, ▷ Latency for all arcs (i,j) up
10: $\{D_{ij}^L \forall (i,j) \in \mathscr{A}'\}$ ▷ Link availability for all arcs (i,j)
11:)
12: **if** $k \neq \varnothing \wedge \alpha_s - H_{sp} - e^P > 0$ **then** ▷ Reduced cost from Eq. 6.42
13: **return** $m \leftarrow$ *new mapping*(k) ▷ Mapping with primary only
14: **else**
15: **return** SPPRC-CGA(s, p) ▷ Algorithm 4

The SPPRC solved by the SPPRC-LA sub routine in Algorithm 5, expands the SPPRC of SPPRC-L by including availability as a constrained resource. Availability is different from the latency resource, by having a lower bound instead of an upper bound. In addition the availability is the product of all chosen links availability. Since link availability is strictly ≤ 1.0, this product is monotonically decreasing and is therefore still applicable to the SPPRC labelling algorithm's label expansion step and label domination step, so long as the expansion step implements the availability resource consumption as a product and not a sum, and the availability dominance test handles availability in accordance to a resource restricted with a minimum value. To allow the SPPRC-LA sub routine to be implemented as a more generic labelling algorithm, the availability resource can be reformulated to a summation resource with an upper bound by the use of logarithms. The availability resource of a path k will then be formulated as follows:

$$\sum_{(i,j) \in A_k^{PATH}} -\ln D_{ji}^L \leq -\ln Y_s \tag{6.48}$$

Chapter 7

Computational Study

This chapter presents the methodology behind, and results generated from, testing the CSBQANR solution methods presented in Chapter 6. The methodology is presented in Section 7.1 and contains an explanation of the test instances, the test setup and execution and the test cases. Section 7.2 presents results obtained for the test cases, as well as a discussion of these results.

7.1 Methodology

This section presents the methodology of the testing of the CSBQANR solution methods. First the five constructed problem instances used in the testing are presented; the dimensions of the instances, the parameters used and the structure of the networks of the instances are detailed. Following this, the test setup and execution is detailed, explaining the solution methods' implementation specifics and the solver and configurations used in the testing. The section is finished with an explanation of the different test cases run in the testing of the solution methods.

7.1.1 Test Instances

The inspiration for the networks in the test instances presented in this section is the networks of a large telecommunications operator. The instances in themselves are not modelled directly after these networks, but the number and placements of the networks'

nodes and arcs are realistic based on these real world wide networks. The test instances used in this chapter were constructed using a self-produced web application, referred to as the Visualised Data Editor (VDE), with an interactive user interface enabling the user to visualise the network and to change it directly using the interface. The VDE is presented in Appendix C.

Test Instance Dimensions

A total of five different test instances are used for testing the CSBQANR solution methods. These five test instances have different dimensions with varying numbers of customers, services, internal nodes and arcs. The number of providers is kept at four, and each customer demands between one and three services. The *M-6* test instance is thought to be a minimal version of a network and is probably too small compared to any real world instance with market potential for a BC. However, it is useful for testing and validating the different solution methods efficiently. Its name indicates that it is a minimal instance with 6 customers. The naming of the rest of the test instances indicates whether their internal networks are relatively dense or sparse (compared to each other), and the number of customers each instance has. The *S-20* test instance (sparse, 20 customers) has a more realistic network structure, but without excessive capacity. The number of customers is assumed to be realistic for a small BC where customers in most cases only require one or two services each. The *S-40* test instance (sparse, 40 customers) has a larger number of customers who in average demand two services each. This instance has a relatively sparse internal network, similar to the networks of the M-6 and S-20 test instances, but is constructed with a larger number of customers to test how a limited network can support larger demands. The *D-20* test instance (dense, 20 customers) is based on the S-20 test instance, but its internal network has been extended with additional capacity both in regards to arcs and internal nodes. The *D-40* test instance (dense, 40 customers) is an extension of the D-20 test instance, with an increased number of customers and services. Table 7.1 presents the different test instances. The names of the test instances used here will be used throughout this thesis.

TABLE 7.1: Key attributes of the different test instances

	M-6	*S-20*	*S-40*	*D-20*	*D-40*
# of Customers	6	20	40	20	40
# of Services	8	24	80	25	80
# of Providers	4	4	4	4	4
# of Internal Nodes	8	17	13	30	30
# of Arcs	52	100	136	196	276
# of Internal Arcs	36	60	56	116	116

Availability and Latency

Every service in the CSBQANR problem requires a maximum latency level and a minimum availability level to be upheld. These values are chosen within a suitable range based on numbers from real world cases. The availability requirement for a service, Y_s, lie in the range between 95 % and 99.9 %. The round trip latency values accepted by a service, G_s, range between 35 ms and 105 ms. Every arc in the problem has an expected latency value and every link has an expected availability value. The latency of the arcs has a value between 2.5 ms and 7.5 ms, with the exception of a few select arcs representing connection between different geographical regions where the latency values are significantly higher. The availability of the networks' links all have a value between 99 % and 100 %, and since only failures of links are considered, both arcs representing a link are assigned the same expected availability value.

Customer Demand

The maximum number of services a customer can require is three in every test instance. Each of these services requires a specified amount of bandwidth to be reserved to support the amount of traffic it will produce. This demand can be different in the two directions; from the customer towards the provider and from the provider to the customer. The bandwidth demands are in the range of 2 to 40, but the majority lie between 10 and 20. Each service is to be provided by a provider in the network, and only a subset of the providers is eligible to provide each service. In the test instances presented here, each service can be placed at two of the four providers.

Revenue and Costs

Due to a lack of relevant information about prices and price levels in the business to business cloud computing market, all costs used in the test instances are constructed artificially and approximated by the use of common sense. The costs of reserving capacity, placing a service at a provider and the revenue generated by serving a customer are all set so that they are reasonable compared to each other. Prices on the reservation of one bandwidth unit on an arc are in the range of 0.5 to 2, the price of placing a service at a provider ranges from 100 to 1400, and the revenue from a customer is in the range of 500 to 10 000. The five test instances have different ranges in which these values are chosen, to adjust costs and revenue to each other and to enable the serving of customers for each specific test instance.

Test Instance Structure

The structure of the test instances presented in this section is based on the network of a large telecommunications operator with network structure in both Europe and Asia connected by high capacity long distance links. The providers placed in Asia have lower costs compared to providers in the European part of the network. The M-6 test instance has its customers placed in the region corresponding to Europe, but has provider nodes in both the European and Asian regions. The S-40 test instance has customers in both regions, but the majority is situated in the European region. This test instance has a limited number of connections between the regions and the customers do in many cases share a connection point to the internal network. The S-20 test instance has an equal number of customers in both regions, but few customers connected to each connection point to the internal network, compared to the S-40 test instance. The same limited number of connections between the regions is found here. The D-20 test instance is similar to the S-20 test instance in regards to customer and provider nodes and placement, but the number of connections in all parts of the network is considerably increased and the number of internal nodes is higher, providing a denser network compared to the network of the S-20 test instance. The D-40 test instance has the same internal network as the D-20 instance, but has the same number of customers and services as the S-40 test instance. The D-40 also has providers in both the European

and Asian parts of the network, and the customers are distributed evenly between the two regions. Figures 7.1 to 7.5 display the five test instances.

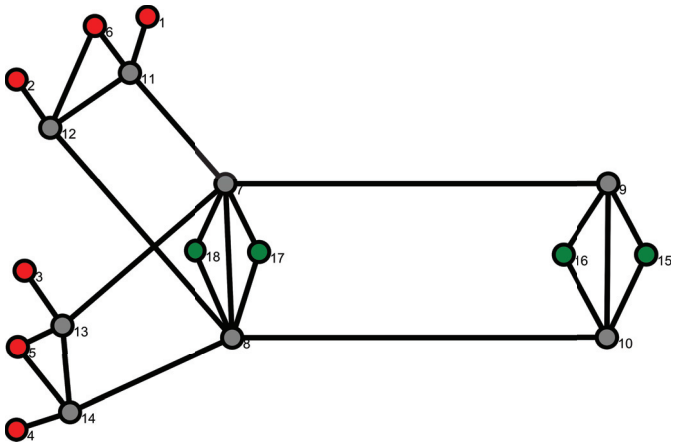


FIGURE 7.1: The M-6 test instance, the red nodes are customer nodes, the green nodes are provider nodes and the rest are internal nodes. The numbers are node numbers automatically generated by the VDE

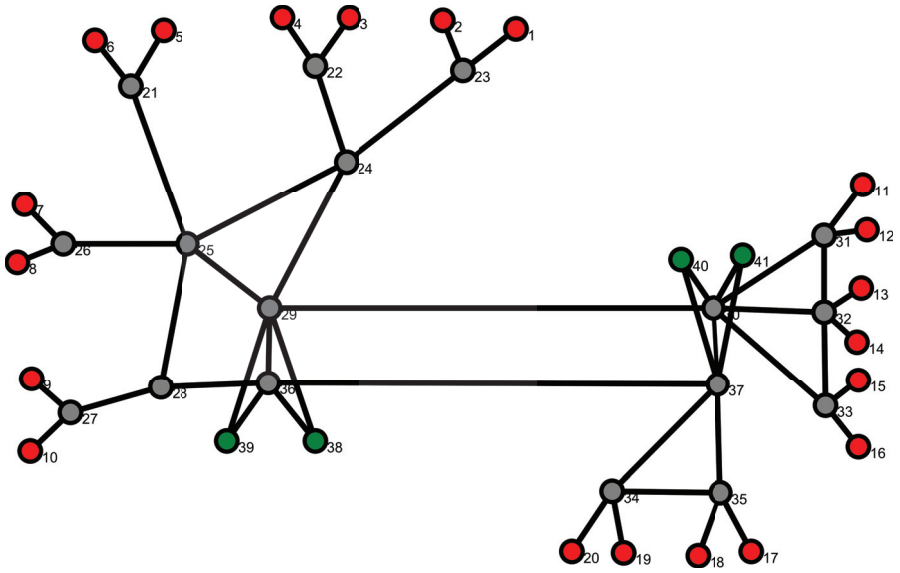


FIGURE 7.2: The S-20 test instance, the red nodes are customer nodes, the green nodes are provider nodes and the rest are internal nodes. The numbers are node numbers automatically generated by the VDE

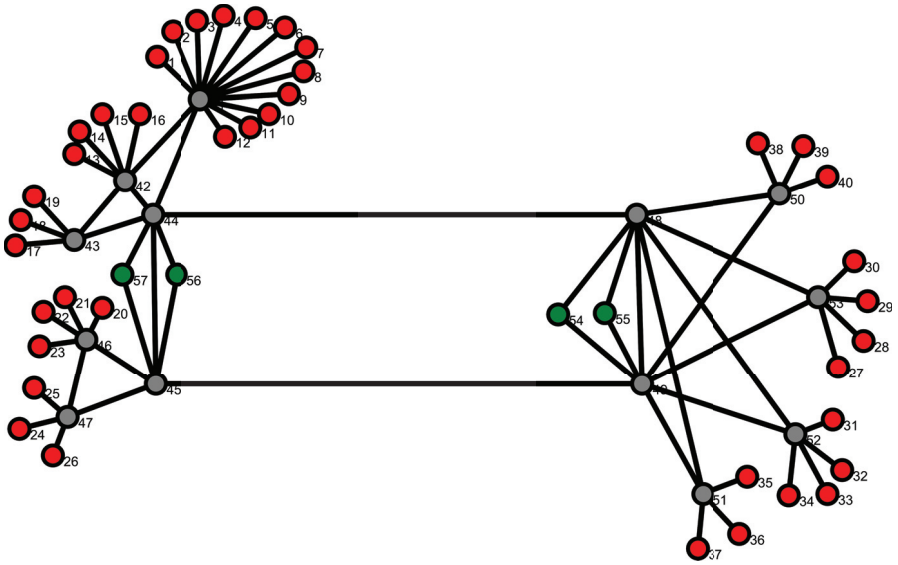


FIGURE 7.3: The S-40 test instance, the red nodes are customer nodes, the green nodes are provider nodes and the rest are internal nodes. The numbers are node numbers automatically generated by the VDE

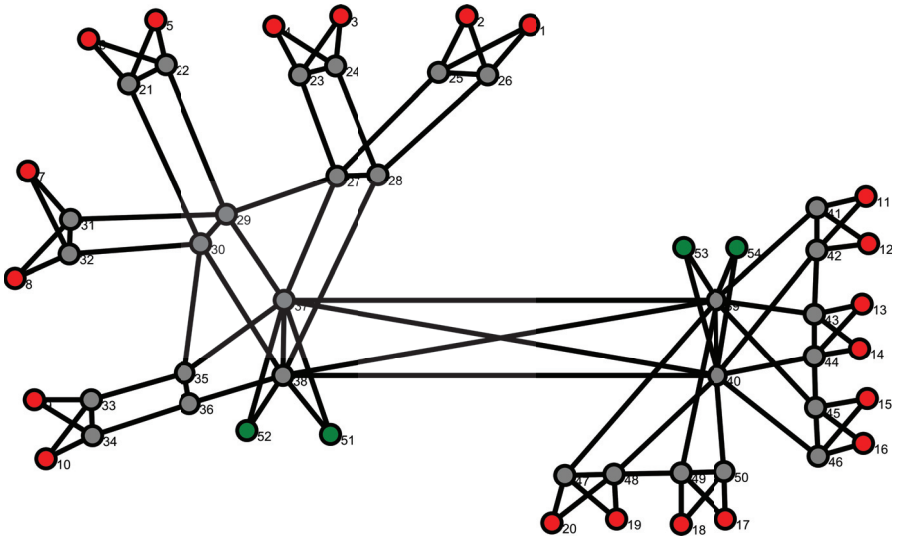


FIGURE 7.4: The D-20 test instance, the red nodes are customer nodes, the green nodes are provider nodes and the rest are internal nodes. The numbers are node numbers automatically generated by the VDE

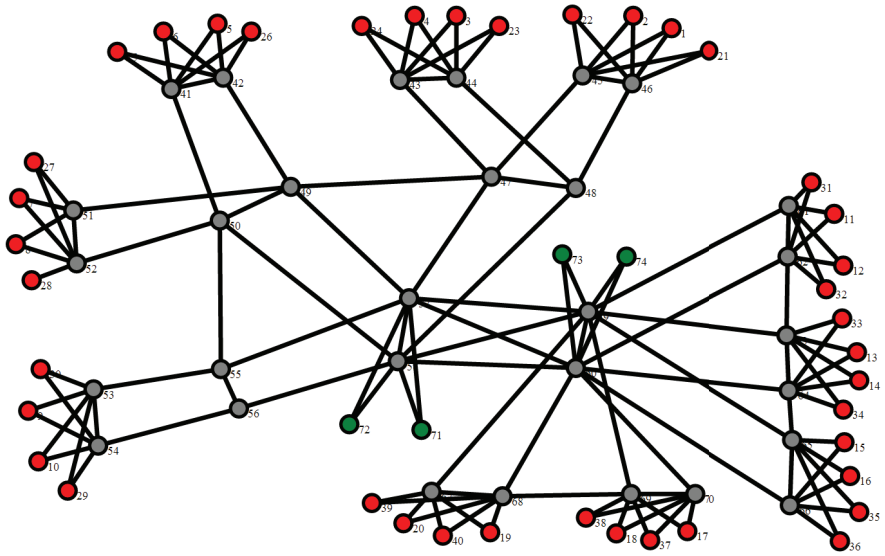


FIGURE 7.5: The D-40 test instance, the red nodes are customer nodes, the green nodes are provider nodes and the rest are internal nodes. The numbers are node numbers automatically generated by the VDE

Validity of Test Instances and Results

Because of the assumptions and estimations used to construct the test instances used in the testing of the CSBQANR solution methods, the results obtained from solving these test instances are not intended to be used for decision making in a real world setting. The focus is rather to test the methods' applicability, compare the different solution methods to each other and test how well they handle large and complex problem instances. As mentioned in Section 8.2, an interesting extension to this work could be to find more realistic data to test the models and solutions methods on, but this is left for future work.

7.1.2 Test Setup and Execution

Implementation

The LFM presented in Section 6.2 and PFM presented in Section 6.3 were implemented in the Mosel modelling language, while the mapping model presented in Section 6.4 was implemented in C++ using the Xpress-BCL Builder Component Library. This model implementation was combined with a custom C++ component implementing each of the column generation heuristics from Section 6.5.

The pre-generation of paths and mappings, described in Section 6.3.1 and Section 6.4.1, respectively, were implemented in C++. The output from these procedures can either be stored to data files compatible with Mosel implementations of the PFM and mapping model, or handed directly to the BCL implementation of the mapping model, or the column generation method of Section 6.5.1, as C data structures.

The C++ components have been bundled into a single application taking problem instances in JSON-format as input. The application can pre-generate paths and mappings for the loaded problem instance, then either output the problem instance including the pre-generated paths and mappings to a Mosel compatible data file, or hand the problem instance directly to one of the C++ solution methods. This application is explained further in Appendix D.

The VDE web application for visualising, creating and modifying problem instances, mentioned in Section 7.1.1, can export problem instances to either Mosel data format, compatible with the LFM Mosel implementation, or to a JSON data format, compatible with the C++ application. Appendix C includes additional information about the VDE.

MIP Solver and Configuration

All optimisation models are solved using FICO® Xpress Optimisation Suite v7.6.0 64-bit on machines with 126.13GiB of memory, 64 AMD Opteron^{TM} 6274 processor cores (@ 2.2GHz) and running CentOS 6.5 x86 64-bit Linux distributions. Default settings were used for the Xpress solver apart from a maximum time for the optimisation process set to 10 hours and selecting primal simplex as the LP solution method when performing LP optimisation as well as when solving the LP relaxation in the root node of the branch and bound tree when performing MIP-optimisation.

The default setting for LP solver algorithms is to perform concurrent solve using dual simplex, primal simplex and Newton Barrier algorithms in the branch and bound root node, and terminating all other concurrent solves once one of them finds LP optimum. The recommended setting is to keep the concurrent solves, as for multi-core systems, the different algorithms are run in parallel without affecting each other, and minimal solution

time is obtained. Due to issues experienced with concurrent solve not terminating once optimum is found by one of the methods, the LP solution process was set to use one algorithm only. In general, dual simplex tends to perform better for problems that are not infeasible or near infeasible, but for the models presented in this thesis, this is not the case. For all test cases where more than 0.5 seconds were needed to find LP optimum, the primal simplex proved to be most efficient, and was therefore chosen as the solution method. For test cases where LP optimum was found in less than 0.5 seconds, the dual simplex method tended to be faster, but this is likely due to the concurrent solves being initialised in fixed order with dual simplex being first, giving dual simplex a small, but sufficient, head start. By selecting to only use the primal simplex for these instances, the optimum was found equally fast as with dual simplex in concurrent solve.

In addition to the Xpress configurations described, the custom built column generation component was also configured to limit the time spent on column generation to one hour. This time limit was implemented so that the column generation is terminated *after* the first full column generation iteration bringing the total column generation time consumption above the time limit, thus the actual time spent may exceed the configured time limit.

7.1.3 Test Cases

The Mosel implemented solution methods of the LFM and PFM (Section 6.2 and Section 6.3) are henceforth referred to as m1 and m2, respectively, the mapping model implementation using all possible pre-generated mappings, is referred to as m3. The different column generation heuristics based on the mapping model are referred to as cg1, cg2 and cg3. Here, cg1 represents the solution method using *Column Generation by Brute Force Search*, presented in Section 6.5.1, while cg2 and cg3 represent the solution methods using the two column generation methods from Section 6.5.2, *SPPRC Column Generation Heuristic A* and *SPPRC Column Generation Heuristic B*, respectively. The term test case is here used for a single run using one solution method with a specific problem instance as input and a certain β value. Every test instance presented in Section 7.1.1 was solved using m1, m2, m3, cg1, cg2 and cg3, and each combination of these were run using β values of 0.0, 0.25, 0.5, 0.75 and 1.0.

The pre-generation of input for use in m2 and m3 for the test instances D-20 and D-40 produce too many paths and mappings making it impossible to run these combinations without exceeding the memory available on the systems used for testing if all paths and mappings are included. The size of the pre-generated input is therefore reduced by restricting the number of paths generated for each combination of service and provider placement, as described in Section 6.3.1. This maximum path limit for each service provider pair is set separately for each combination of solution methods m2 and m3 for the test instances D-20 and D-40. These path limits are set so that runs using a higher path limit well exceed the available memory before the 10 hour limit is reached, and are derived by the use of trial and error. A similar problem with the size of the pre-generated data appears for cg1 for test instance D-40, where including the full size of the pre-generated data results in excessive time consumption on each column generation iteration. A maximum path limit is therefore also set for cg1 for the D-40 instance, using a limit that brings the number of paths generated down to a manageable size for the service provider pairs with excessive amounts of potential paths. The path limits for m2, m3 and cg1 for the different solution methods, used in the instances D-20 and D-40 can be found in Table 7.2.

A second set of such path limits is introduced for running m3 with D-20 and D-40, based on the number of mappings the column generation methods produce for the same test instances. This set of path limits is used to compare the solutions obtained from the different column generation solution methods with the results from solving m3 with a restricted set of mappings similar in size to the sets of mappings the column generation methods produce. Tables including the numbers of mappings produced by the column generation methods, and the resulting numbers of mappings for each path limit are presented in Appendix B (Table B.12, Table B.13 and Table B.14). Two types of path limits are defined for each of the two test instances. The first type aims at producing a set of mappings with a size as close as possible to the average number of mappings produced by all column generation solution methods for all β values for that specific test instance, referred to as the AVG path limit. The second path limit is set to produce the smallest set of mappings for that specific test instance that is higher than the largest number of mappings produced by column generation, referred to as the MAX path limit. The appropriate path limits for each of the two path limit types for D-20 and D-40 is

obtained by iteratively performing mapping pre-generation for increasing path limits, noting the number of mappings produced, until the appropriate limits are identified. This set is presented in Table 7.3

TABLE 7.2: Maximum limits for paths and the resulting number of mappings for running cg1, m2 and m3 on D-20 and D-40

		D-20	*D-40*
cg1	Path Limit	-	100
m2	Path Limit	5	4
m3	Path Limit	20	10
	Resulting # of mappings	4788	1272

TABLE 7.3: Average and maximum number of mappings of the column generation methods, comparison path limits and the resulting number of mappings for running m3 on D-20 and D-40

		D-20	*D-40*
cg1-3	Average # of mappings	298.1	609.7
	Max # of mappings	400	730
m3	AVG Path Limit	5	4
	Resulting # of mappings	294	555
m3	MAX Path Limit	6	5
	Resulting # of mappings	448	818

Lastly a set of test cases is run by increasing and decreasing the availability requirements of each service in the S-20 test instance to respectively 0.995 and 0.95, and running these extended instances on every solution method with β values of 0.0 and 0.5. This is done to investigate the effect a change in the input to the solution methods will have on solution values and solution times. The choice of what parameter to use fell on the availability requirements because availability is such a central parameter in the CSBQANR problem and because of the difference in modelling of the availability in the solution methods. The effect of other parameters are discussed for a similar problem in Braaten and Holmen (2013). To limit running time used, only two β values were selected, the two β values are chosen to cover the diversification of the β range to some degree.

7.2 Results

This section presents some of the results of testing the CSBQANR solution methods and provides a discussion of these results. First, some general results are presented, followed by a discussion of the performance of each of the solution methods. Following this are discussions of the effect the β value and varying of the availability requirement have had on the complexity of the problem and the solutions obtained. Thereafter, the scalability of the solution methods is discussed, and the section finishes with a discussion of possible sources of errors found in the solution values and times derived from the testing. For the full results, see Appendix B.

Every test instance presented in Section 7.1.1 was solved using m1, m2, m3, cg1, cg2 and cg3, and each combination of these were run using β values of 0.0, 0.25, 0.5, 0.75 and 1.0. Solution values, the time at which these were found and total times for the different combinations of the CSBQANR problem solution methods and test instances run with $\beta = 0.25$ are presented in table 7.4. For M-6 the time at which best solution was found is excluded because of the small margins between this time and the total time. A maximum limit of 10 hours for the MIP process was used; in the cases where this limit was reached, the solution process was terminated and the best found solution within that time was recorded as the corresponding solution value. In the table these entries have a total time of $> 10h$ and the best bound found at termination is also given. The D-20 test instance run with solution methods m2 and m3 and the D-40 test instance run on cg1, m2 and m3 generate too many paths and mappings to provide results with the test setup used. The values given in the table are from running these combinations by restricting the number of paths generated for each combination of service and provider, as described in Section 7.1.3. The total time values reported are the total times spent on the solution process, including any pre-generation steps. Table 7.5 presents the average solution times spent solving the different test instances for each solution method.

TABLE 7.4: Results for every test instance with β value of 0.25 solved by every solution method, time when best solution was found and total time consumption in seconds. For the cases where optimality was not proved within 10 hours, best bound at termination is also given. Because of the small margins, time when the solution was found is not given for M-6.

$\beta = 0.25$	Solution Method:	cg1	cg2	cg3	m1	m2	m3
M-6	Solution Value	1550	1550	1550	930	1550	1550
	Total Time	0.3	0.1	0.1	0.2	343	0.5
S-20	Solution Value	10970	10840	11130	4780	11130	11170
	Time Found	3.2	8.7	17.6	1.3	1498	22.4
	Total Time	3.3	9.0	18.3	1.5	$> 10h$	27.0
	Best Bound					14495	
S-40	Solution Value	20588	20128	20173	9610	22583	21358
	Time Found	130	134	130	18.4	17022	1908
	Total Time	157	151	203	19.8	$> 10h$	$> 10h$
	Best Bound					32375	25279
D-20	Solution Value	9787	9216	9267	4990	9382	9941
	Time Found	3279	181	83.5	64.6	618	484
	Total Time	5467	187	88.1	14066	$> 10h$	$> 10h$
	Best Bound					9576	10060
D-40	Solution Value	16122	15022	15221	5926	13459	16279
	Time Found	18100	6107	13281	1972	15612	29054
	Total Time	$> 10h$	16977	13812	$> 10h$	$> 10h$	$> 10h$
	Best Bound	16970			17803	15939	17208

TABLE 7.5: Average total time across β values spent solving the different test instances for each solution method in seconds or hours where specified (h)
*solution process terminated due to time limit of 10 hours for one or more β values

Solution Method:	cg1	cg2	cg3	m1	m2	m3
M-6	0.3	0.1	0.1	0.1	225.1	0.5
S-20	2.9	8.7	9.6	22.7	$10h^*$	25.3
S-40	105.4	128.2	125.1	1.6	$10h^*$	$8.8h^*$
D-20	4258	103	84.5	$4.4h$	$9.6h^*$	$5h^*$
D-40	$9.6h^*$	7800	$4.4h^*$	$10h^*$	$10h^*$	$8.4h^*$

The following figures (7.6 - 7.8) show the solution values obtained from solving the S-20, D-20 and D-40 test instances using every solution method and varying the β value. The S-40 test instance is not shown here, as it has very similar behaviour to the S-20, while M-6 is not included since its solutions is not seemingly affected by changing the β values. Similar figures for these test instances can be found in Appendix B as Figure B.2 and Figure B.1, respectively.

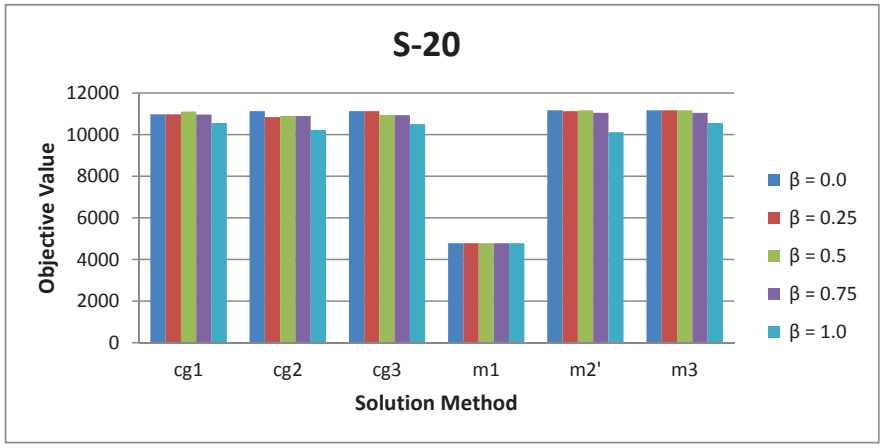


FIGURE 7.6: Solution values for the S-20 test instance run on every combination of solution method and β
'The m2 solution method did not solve the instances to optimality within the time limit of 10 hours

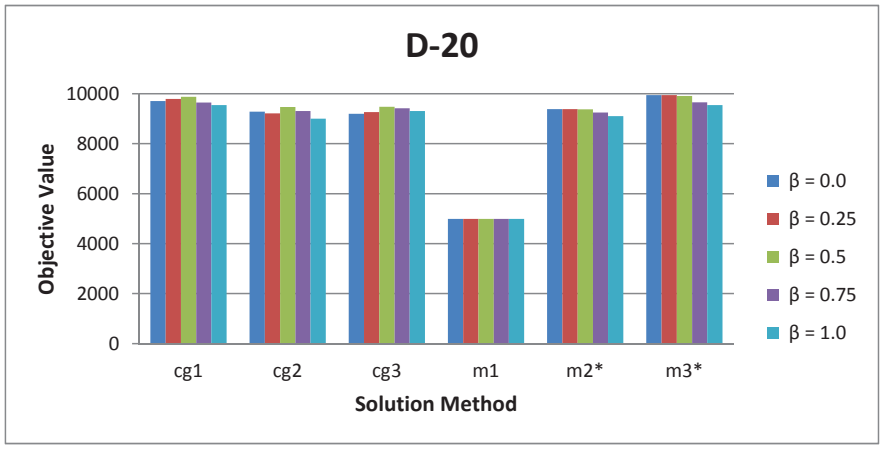


FIGURE 7.7: Solution values for the D-20 test instance run on every combination of solution method and β
*The m2 and m3 methods are run with a maximum limit on the number of paths pre-generated of respectively 5 and 20, as mentioned in Section 7.1.3

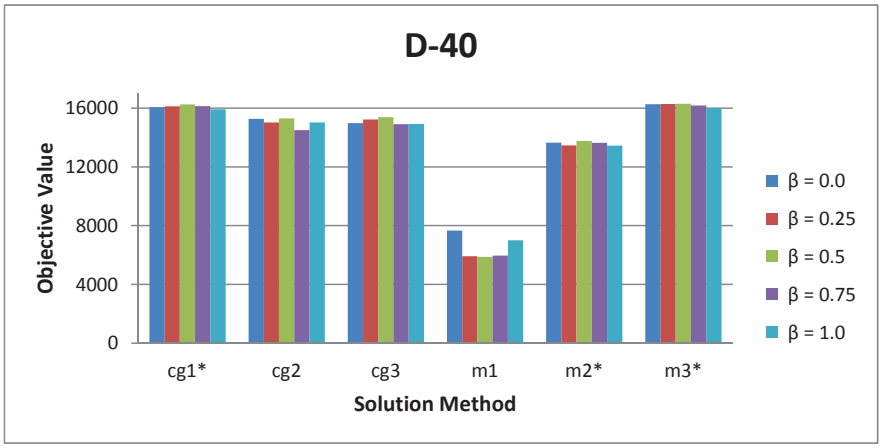


FIGURE 7.8: Solution values for the D-40 test instance run on every combination of solution method and β
*The cg1, m2 and m3 methods are run with a maximum limit on the number of paths pre-generated of respectively 100, 4 and 10, as mentioned in Section 7.1.3

7.2.1 Performance of the CSBQANR Solution Methods

This section will present some results, and discuss the performance of each solution method based on these and the results presented above. The complete set of solution values and solution times can be found in Appendix B.

The m1 Solution Method

Figures 7.6 to 7.8 and Table 7.4 show clearly that the LFM solution method m1 generally provides a lower solution value compared to the other solution methods. As this the LFM includes a link-disjoint requirement, for a service's primary and backup paths, m1 does not solve precisely the same problem as the other solution methods and for some problem instances this requirement proves to be very restricting. For relatively sparse networks, with little possibility of providing disjoint paths between customer and provider nodes, the solution will mainly consist of services that do not require backup paths. This is the case with the S-20 test instance, where eight customers each with one service only using a primary path are served. No backup capacity is reserved and so the solution does not vary across the different β values, this is also seen in the S-40 test instance. The same tendency of a relatively low solution values is seen for all the test instances. As seen in Figure 7.9, m1 generally chooses to serve fewer customers and thereby also place fewer services than other solution methods, in the figure, m3 is shown

for comparison. The number of services with backup is comparably low as well, as seen in the same figure.

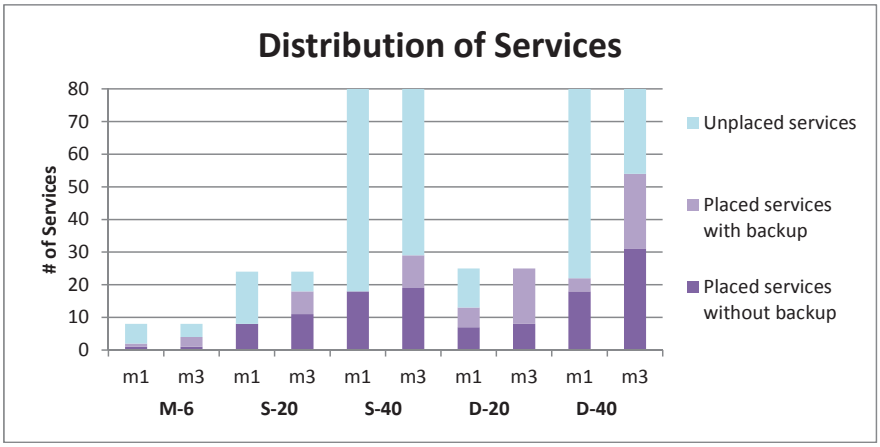


FIGURE 7.9: The service distribution of running m1 and m3 on every test instance with a β value of 0.0

The test instances with solutions including backup capacity reservation are M-6, D-20 and D-40, who all have relatively dense networks compared to their number of services and can therefore provide total link-disjoint paths from the customers to the providers. The varying β value has seemingly no effect on the solution values obtained from the m1 solution method when run to optimality. The m1 solution method requires totally disjoint primary and backup paths for a service as well as no overlapping of both primary and backup paths between two services. This results in very few overlapping backup paths and as the β is used to decide the amount of capacity to be reserved on highly shared backup links; it has limited effect on the solutions.

The strict disjoint requirement also effects the solution time of the m1 solution method. The number of possible network routings is greatly reduced, and the modelling of the amount of backup capacity required is made simpler. In the S-40 and S-20 test instances the reduced size of the solution space and simpler modelling seems to make the computation noticeably simpler, decreasing the solution time compared to the other test instances and solution methods. However, as seen in Table 7.4, the time to find the optimal solution for m1 is quite small compared to the total time it uses to prove that this solution in fact is the optimal. The number of complete network routings valid in

an integer solutions is reduced for m1, compared to the other methods, but as the link-disjoint requirement is less restricting without binary requirements, the LP produces a lot of non-integer solutions m1 has to investigate to be able to prove optimality.

The m2 and m3 Solution Methods

The solution methods m2 and m3 consist of specialised pre-generation algorithms (Section 6.3.1 and Section 6.4.1) and respectively a PFM and a mapping model presented in Section 6.3 and Section 6.4. The difference between m2 and m3 is that m2 tries to build network resource mappings from sets of pre-generated paths for each service, while m3 selects network resource mappings from pre-generated sets for each service. Apart from this, the two models should produce equally optimal solutions when all pre-generated paths and mappings are included, solving equivalent problem instances and allowed to complete their solution processes.

The most apparent difference between the m2 and m3 solution methods is thereby their pre-generation methods. The pre-generated mappings to m3 contain data at a higher abstraction level where more decisions have already been made compared to the pre-generated input to m2. As one can see in Table 7.4 and in the complete result tables found in Appendix B, the solution values of m2 and m3 run with equivalent input data are identical when solved to optimality and very close to each other even when not solved to optimality, with the exception of the S-40 test instance. For S-40, m2 has found a better solution than m3 for all β values except $\beta = 0.0$. For $\beta = 1.0$, m2 even finds a better solution than m3's optimal solution. In every solution produced by m2 for the S-40 instance one or more path combinations that are not allowed by m3 are used. This leads to m2 producing higher solution values than m3. This behaviour is due to small numerical differences in the input data for m2 and m3, and is discussed further in Section 7.2.4. The results for S-40 are therefore disregarded when comparing m2 and m3.

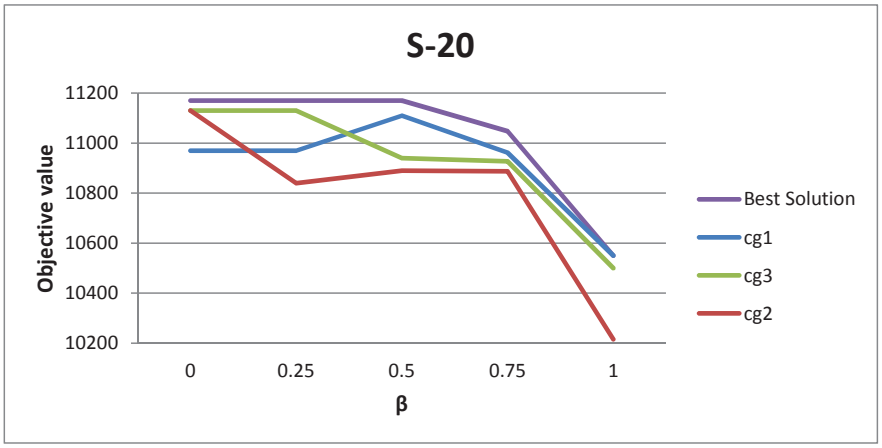
While the solution values are either identical or very similar, the total solution times of m2 and m3 are quite different, as well as the time at which the best solution is found. For M-6, m2 uses a total time between 0.6 and 5.7 minutes, while m3 uses less than 2.5

seconds for all β values. Additionally, m3 consistently finds a closer best bound than m2, as seen for cases where both solution methods are terminated due to the time limit in Table 7.4 . In addition, the time at which the best solution is found is much less for m3 compared to m2. For the S-20 test instance m3 finishes in a matter of seconds while m2 is terminated after running for 10 hours. In this case, m2 finds the same solution as m3 with three of five β values, but fails to prove its optimality within the time limit. For the D-20 and D-40 test instances, m2 has a best bound lower than the solution provided by m3. As these solution methods are supposed to solve the same problem, this might seem wrong. However, the restriction of number of paths pre-generated is much more strict for m2 than for m3 (see Table 7.2), giving m2 a more restricted solution space. Comparing m2 and m3, it seems that m3 is a better choice for all test cases, when S-40 is disregarded due to misleading results. The extra level of abstraction and early decision making in the m3 solution method seems to ease the solution process considerably compared to the m2 solution method.

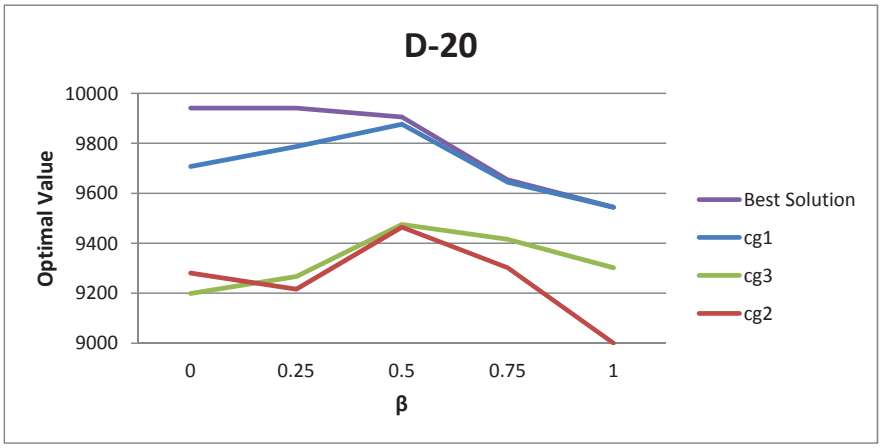
Column Generation Solution Methods

For the M-6 test instance all the column generation solution methods produce solutions equal to the optimal solution (β equal to 0.0, 0.25, 0.5 and 1.0) or very close (within 99.98% for $\beta = 0.75$). For other test instances the different column generation solution methods produce different results. Solution values obtained with the different column generation solution methods, along with the best found solution across all solution methods, for test instances S-20 and D-20 are shown in Figure 7.10. In relation to the best solution found, one can see that the column generation solution methods follow this quite close, especially for cg1 and $\beta \geq 0.5$. In general, cg1 seems to produce better solutions compared to cg2 and cg3, but this is not always the case, which demonstrates the heuristic nature of the column generation solution methods.

The minimum and average solution values, relative to the best found solution for that specific test instance and β value, for the different methods is presented in Table 7.6. When only regarding the solution values produced, cg1 is the better choice of method for column generation for the instances tested, having significantly better solution values than cg2 and cg3, where cg3 is slightly better than cg2. On average, all three column



(a) Solution Values of cg1, cg2 and cg3 run with S-20



(b) Solution Values of the Column Generation Solution Methods run with D-20

FIGURE 7.10: The solution values obtained from running the column generation solution methods on the test instances S-20 and D-20 plotted against each other and against the best found solution, of all solution methods, per β value

generation solution methods perform quite well, cg1 produces on average a solution within 98.1% of the best found solution, cg2 is within 94.7% and cg3 within 95.4%. Excluding the special results from the S-40 instance, where m2 produces better solutions than allowed by m3 or any of the column generation solution methods, these numbers are even better. The respective average relative solution values are then 99.4%, 96.3% and 96.9% for cg1, cg2 and cg3. It is worth noting that even though the column generation solution methods produce good solutions, the solutions found with m3 are still better for instances where m3 was terminated due to the time limit and optimum is unknown, albeit at the cost of higher solution times.

TABLE 7.6: Minimum and average solution values of cg1, cg2 and cg3 relative to the best solution found, across all test instances. A set of results excluding S-40 is included due to m2's behaviour for S-40, discussed in Section 7.2.4

Test Instances	Solution Method:	cg1	cg2	cg3
All	Minimum % of best	91.2	85.4	86.5
	Average % of best	98.1	94.7	95.4
All Except S-40	Minimum % of best	97.7	89.6	92.1
	Average % of best	99.4	96.3	96.9

As seen in Table 7.5, the different column generation solution methods behave quite similar in terms of average solution time for test instances M-6, S-20 and D-40, with cg1 slightly ahead of cg2 and cg3 for the latter two instances. For the test instances D-20 and D-40, the total solution time differs significantly. For D-20, cg1 spends considerably more time on the solution process, compared to cg2 and cg3. D-20, having a dense network, creates a high amount of potential paths in the pre-generation step for cg1, causing the majority of total time being spent on cg1's process of performing exhaustive searches through all potential mappings for each service provider pair for each column generation iteration. Although, for β values 0.0 and 0.25, the time spent on the MIP process alone for cg1 far exceeds the total time consumption by cg2 and cg3. For the D-40 test instance, the number of paths used to generate columns for cg1 is restricted, as mentioned in Section 7.1.3, making the time spent searching through all potential mappings a smaller portion of the total time consumption, compared to D-20, resulting in a total time consumption similar to cg2 and cg3 for these cases.

Column Generation Methods Compared to m3

To evaluate the mapping columns generated by the column generation solution methods, their solutions are compared to the solutions obtained from solving m3 using the AVG and MAX path limits, defined in Section 7.1.3, on the D-20 and D-40 test instances. Table 7.7 shows the solution values and solution times obtained from solving m3 with the AVG path limit. The results from both the AVG and MAX path limits can be found in Table B.7. The solution values and times of the column generation methods with $\beta = 0.25$ can be found in Table 7.4, for all other β values, see Appendix B.

TABLE 7.7: Results from m3 with maximum paths per service provider pair limit reduced to generate number of mappings approximately equal to Column Generation Methods average number of mappings

Instance	AVG Path Limit	β	Solution Value	Solution Time
D-20	5	0.00	9382.0	9.8
		0.25	9382.0	9.1
		0.50	9372.0	9.5
		0.75	9245.5	9.1
		1.00	9102.0	3.9
D-40	4	0.00	13907.0	1408.3
		0.25	13907.0	1741.2
		0.50	13881.5	1456.8
		0.75	13691.0	1342.1
		1.00	13475.0	583.2

For solution values obtained for the D-20 test instance, m3 with the AVG path limit outperforms both cg2 and cg3 for β values 0.0 and 0.25, and cg2 for $\beta = 1.0$, while cg2 and cg3 produces the best solution for the remaining β values. This implies that the mapping columns generated by cg2 and cg3 are not necessarily better than a similarly sized set of mapping columns obtained from simply limiting the number of paths produced in the pre-generation by only accepting the first N paths produced for each service provider combination, resulting in the N shortest paths in terms of number of links traversed, as described in Section 6.3.1. cg1, on the other hand, outperforms m3 with AVG path limit for all β values for the D-20 test instance. When D-20 is solved with m3 MAX path limit, cg2 is outperformed for all β values and cg3 for β values 0.0, 0.25 and 0.5, but cg1 still produces the best solution. Note that m3 with the MAX path limit, produces 448 mappings, considerably more than cg2 and cg3, their largest number of mappings across all β values are respectively 275 and 337 (see Table B.13 and Table B.12). For this reason results in this case will be skewed in favour of m3 compared to cg2 and cg3.

For the D-40 test instance, Table 7.7 shows that m3, with the AVG path limit, is outperformed by all column generation solution methods for all β values. When the MAX path limit is used for m3, the tendency remains the same, with the exception of cg2 performing slightly worse than the MAX path limit restricted m3 for $\beta = 0.75$. For this larger and more complex test instance, it is evident that the mapping columns added by column generations do improve upon similarly sized subsets of mapping columns

obtained from simply restricting number of paths to the N-shortest in terms of number of links traversed.

Looking at solution times for the restricted m3 compared to the column generation methods, they are considerably lower than the solution times for the column generation solution methods, for both D-20 and D-40, when using the AVG path limit. This difference in total solution time is in part due to time spent on the column generation steps for cg1, cg2, and cg3, but the time spent on the MIP solution process in the column generation solution methods is also considerably larger compared to the MIP solution process of m3 with the AVG path limit. This may be due to the column generation solution methods being performed for the LP-relaxation of the problem only, thus resulting in mapping columns specialised for an LP-optimal solution, making the process of finding a MIP optimal solution harder compared to other similarly sized set of mappings. However, for D-40, when m3 uses the MAX path limit, the total solution time of the restricted m3 is very similar to the total solution times for cg2 and cg3, where both cg2 and cg3 produces better solution for all β values, except β 0.75 for cg2.

7.2.2 Effects of Varying β and Availability Requirements

Firstly, this section presents some results and discusses the effect of using the β scaling parameter and indicating its influence on the solutions obtained and the solution process. Thereafter the effect of making the availability requirements of each service both higher and lower on the resulting solution values and solution times is discussed.

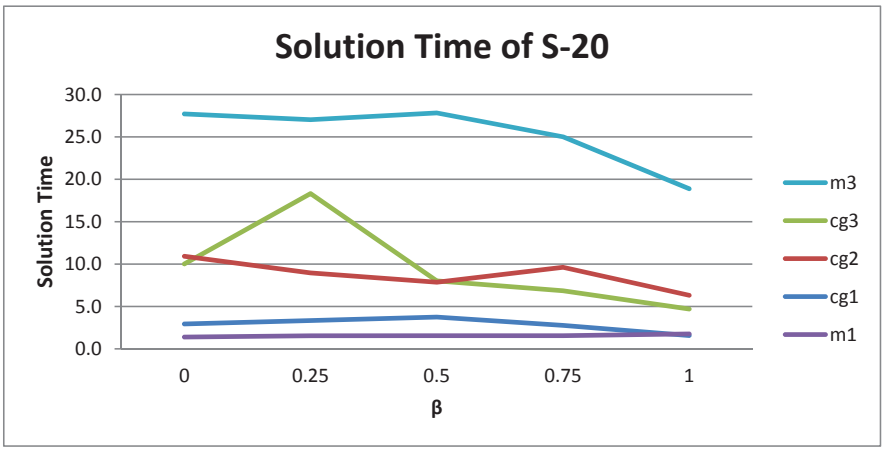
Effect of the Use of β on Solutions and Performance

The expected effect β would have on the solution values was that the values would decrease with an increase in the β value, as the backup requirements would become stricter. This seems to be the general tendency in the solution methods m1, m2 and m3 when these are allowed to run their full course. The solutions provided by the column generation solution methods are heuristic and as such they cannot be expected to always display the same behaviour as m1, m2 and m3 with the variation of β values. Figures 7.6 to 7.8 show that the solutions from these solution methods vary more randomly

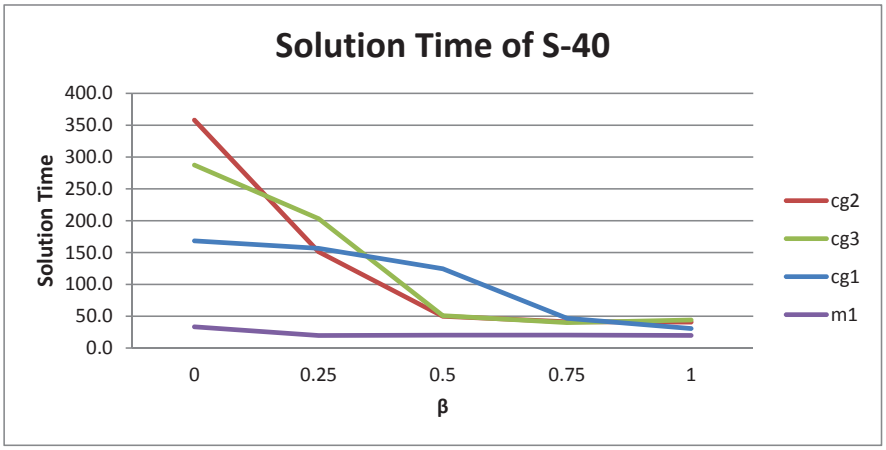
with an increase in β compared to the pre-generation based methods and m1. However, the general tendency points towards a decrease in solution value for higher β values, at least for β equal to one. This apparent randomness displayed by the column generation solution methods can be explained by the fact that the column generation methods produce different set of heuristic mappings for each β value and the quality of these can differ. The difference between the heuristic solution values and the optimal values for each β value can be different depending on how good the set of mappings produced by the column generation methods is for that β value.

Figure 7.11 displays the development in solution time across the β values for two of the five test instances. The two instances are chosen to show the spread of development tendencies in the different test instances. For the S-20 test instance, the m2 solution method is not included in the figure as those cases were all terminated by the time limit of 10 hours and so has an solution time $> 10h$. For the same reason both m2 and m3 are excluded from the S-40 figure (Figure 7.11(b)). However, there is one exception, the m3 solution method ran on S-40 with $\beta = 1.0$ reaches optimum after 14394 seconds, indicating that this test case is easier to solve than S-40 with lower β values.

When regarding the solution times across different β values, especially in the S-40 test instance (see Figure 7.11(b)), one can see that there is a specific tendency. As the value of β increases it seems that the problem gets increasingly easier to solve and the solution time decreases. This tendency is not equally obvious in every test instance, for the S-20 instance it is less apparent, but still present to some degree, see Figure 7.11(a). β is used in the constraint regulating the amount of backup capacity to be reserved for an arc and the changing of its value changes the problem to be solved. For a β equal to zero the problem allows every arc in the network to be shared unbounded. For a β equal to one, the problem only supports dedicated protection, disallowing any sharing of backup resources. For values between zero and one, different degrees of sharing is allowed, and the larger the β, the fewer mapping possibilities occur. This decreases the problem size and solving the problem takes less time. For low β values, sharing of backup links is less restricted, and so the LP solution tries to include as much sharing as possible to increase the profit. The problem requires disjoint primary paths for overlapping backup paths and vice versa, i.e. restricting the amount of sharing possible, but in the LP solution



(a) Solution Time for S-20



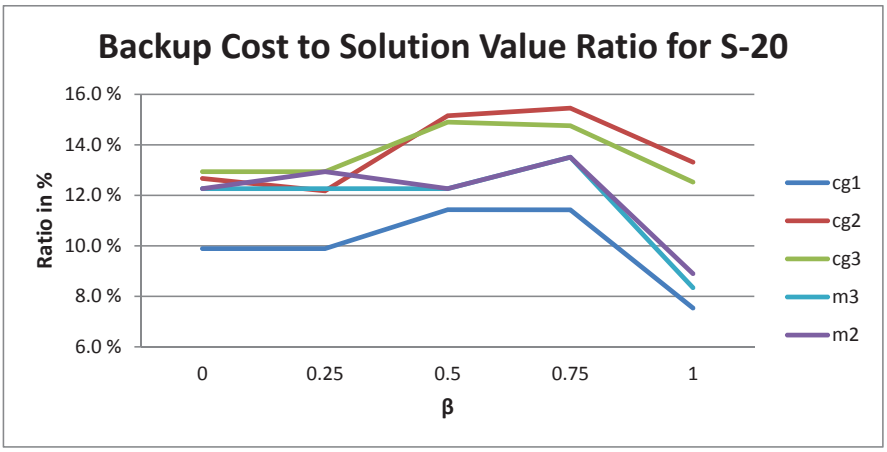
(b) Solution Time for S-40

FIGURE 7.11: Solution time in seconds for S-20 and S-40, for S-40 ran with cg1, cg2, cg3 and m1, for S-20 with m3 as well, for every β. The m2 solution method for both test instances and m3 solution method for S-40 are excluded as they are terminated by the time limit (solution time $> 10h$).

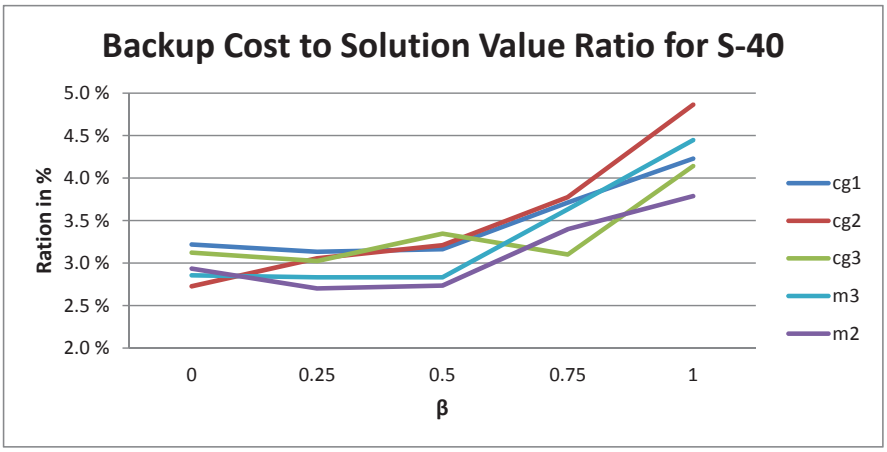
this can be circumvented by partial overlapping. This increases the difference between the LP and MIP solutions and leads to more time being used on finding the optimal integer solution for low β values compared to high β values.

Figure 7.12 shows the ratio between backup costs and solution value obtained from S-20 and S-40 run with every solution method, except m1, for every β. The m1 solution method is excluded as its solutions are the same for every β value. From the figure one can see that for the S-40 test instance, as the β value increases, so does the backup cost ratio. The β value was intended to regulate the degree of sharing and the amount of backup capacity needed to be reserved, so an increase in the relative backup reservation was expected with increasing β values. As the backup cost is a good indicator of the

amount of backup reserved, Figure 7.12(b) indicates that β has had its expected effect. However, regarding the equivalent figure for S-20, Figure 7.12(a), this effect seems to not be as evident. Here, the backup cost ratio increases until $\beta = 1.0$, where every solution method produces solutions with lower backup cost ratio. Consulting the details of the solutions, or the values in Table B.9, one can see that for this β value every solution value decreases considerably as well as the backup costs, indicating that a number of services requiring backup is excluded from the solution when β becomes one and dedicated backup is required. This in turn can explain this fall in backup cost ratio, and one can see that β also in this case has had the expected effect.



(a) Backup Cost to Solution Value Ratio for S-20



(b) Backup Cost to Solution Value Ratio for S-40

FIGURE 7.12: Backup Cost in % of solution values for S-20 and S-40 run with cg1, cg2, cg3, m2 and m3, for every β. Solution method m1 is excluded as its solution is the same across every β value.

Effect of Varying the Availability Requirements

The S-20 test instance was modified to create two additional test instances, one with lower availability requirements for every service and one with higher availability requirements for every service, compared to the original S-20, as mentioned in Section 7.1.3. The results of running these test instances with every solution method for a β value of 0.0 is shown in Table 7.8. The test instances were also run with $\beta = 0.5$, the results of those cases can be found in Table B.8 in Appendix B.

TABLE 7.8: Results for test instance S-20 with low, original and high availability requirements run with every solution method for $\beta = 0.0$
*solution times indicate the time at which the best solution was found, the solution values of these cases are not proved to be optimal within the 10 hours' time limit

$\beta = 0.0$	Solution Method:	cg1	cg2	cg3	m1	m2	m3
Low	Solution Value	16270	16270	16270	16290	16290	16290
	Total Solution Time	1.3	1.6	2.5	1.4	6.3	37.6
	MIP Solution Time	0.0	0.1	0.1	1.1	5.8	3.8
Original, S-20	Solution Value	10970	10840	11020	4780	11170	11170
	Total Solution Time	2.9	38.8	26.6	1.4	$> 10h$	27.7
	MIP Solution Time	1.1	1.4	0.9	1.1	2677*	11.3
High	Solution Value	1970	1625	1625	480	2090	2909
	Total Solution Time	0.8	0.9	1.2	1.2	$> 10h$	1.2
	MIP Solution Time	0.1	0.1	0.1	0.9	1690*	0.9

Regarding the solution values obtained, one can see that lower availability requirements allow for the serving of a larger number of, or more profitable, customers, increasing the solution value obtainable. The increase in availability requirements has the opposite effect; as one can see from the table, the solution values obtained from this case are considerably lower than the values obtained from the original test instance. This effect is similar for every solution method, but is especially large in the m1 solution method. For the low availability instance, the column generation methods are very close to m2 and m3 in regards of solution value, but for the two other instances it seems like this difference becomes larger and m2 and m3 provide the best solution values. The identical solution values of m2 and m3 are expected and support the results presented in Section 7.2.1. The column generation methods provide different, but similar solution

values for the original and high availability test instances, which can be expected due to their heuristic nature.

For the m1 solution method, the solution values obtained in the high availability and original case are considerably lower than the values obtained from the other solution methods. This is in compliance with the effect characteristic for m1 discussed in Section 7.2.1, where the strict link-disjoint requirement to a service's primary and backup paths used in m1 limits the number of services with backup paths it is possible to serve when combined with a sparse network. However, the solution value obtained with m1 from running the case with lower availability requirements is actually better than the value obtained from the column generation solution methods and as high as the values obtained from m2 and m3, implying that when backup paths can be disregarded, m1 is equivalent to m2 and m3. Regarding the solution times of m1, these are almost exactly the same for the three variations of S-20. This indicates that the difference in availability requirements does not affect m1's ability to solve the problem. As m1 does not use any kind of pre-generating, the size of the problem is the same for any values of the parameters. In addition, with its simplified availability approximation m1 does not consider the availability of the combination of primary and backup path; which can indicate that solving m1 is less influenced by changing the availability requirements.

When regarding the solution time values presented in Table 7.8 for the solution methods cg1, cg2 and cg3, these are about the same for both the low and high availability instances, but for the original S-20 the total solution time is considerably higher. As the MIP solution times do not increase as much, it is the column generation in the root node itself that is more time-consuming in the original S-20. The reason for this can lie in the in-between nature of the original test instance. The high availability instance gives a smaller solution space because the high availability requirements exclude a lot of possible mappings and as a result, the column generation methods finishes its search for new columns earlier compared to the original problem. With the low availability instance, the decrease in the number of services requiring backup paths will simplify the process of finding new mapping columns, as only a primary path is needed. The original S-20 has a high number of possible mappings and a large portion of these mappings require

a primary and a backup path, so the solution time of this instance is not restricted by either of the two properties.

Solution methods m2 and m3 react quite differently to changes in the availability requirements. While lowering the availability requirements reduces the total solution time to m2 considerably, m3 shows a slight increase in total solution time as the availability requirement is lowered. The m3 uses input from a pre-generating method where the resulting valid mappings are already validated in regards to availability, this makes the number of mappings produced heavily dependent on the availability requirement, with higher availability requirements reducing the number of possible mappings. While m2 takes its input from the path pre-generating method providing possible paths regardless of their availability levels, it has to evaluate the availability levels of primary- and backup path combinations within its MIP optimisation process instead. This requires m2 to evaluate the same large number of path combinations regardless of the actual availability requirement, and a high requirement makes finding a valid combination harder. For m3, on the other hand, a much lower number of mappings have to be considered in its MIP solution process as the availability requirements are increased, and explains why m3 behaves opposite to m2 with respect to changes in the availability requirements. Note that the time spent by the path and mapping pre-generating methods is included in the total solution time presented in Table 7.8.

7.2.3 Scalability of the CSBQANR Solution Methods

Comparing the solution times of running the m1 solution method on the test instances M-6, S-20 and S-40 (see Appendix B for the complete set of results) to the other solution methods it seems to be the most efficient. However, when moving on to problem instances with denser networks as D-20 and D-40, the column generation methods solve these faster than m1. The denseness of the networks in the test instances run seems to impact m1's solution time considerably, which can be explained by the fact that m1 does not use any column or pre-generating methods to produce its input and therefore has to evaluate every possible routing during its solution process. As the denseness of the networks increase, the number of possible routings increase. Effective use of m1 is thus restricted to problem instances with sparse networks or few services to be placed.

In any case, as mentioned in Section 7.2.1, m1 does not solve exactly the same problem as the other solution methods and to provide good solutions should thereby only be used when total link-disjoint primary and backup paths are essential.

Regarding the solution times resulting from running the m2 solution method, one can see in Table 7.5 that the only test instance it is able to solve within the 10 hour time limit is the minimal test instance M-6 and the single case of D-20 for $\beta = 1.0$, where the number of pre-generated paths is restricted considerably. Even with the M-6 test instance it uses 3.75 minutes on average, compared to the rest of the solution methods having average solution times of less than 1 second. For D-20 with $\beta = 1.0$, the solution time is 8.2 hours, while for all other instances, the solution process is terminated after 10 hours before optimum can be proven. The path pre-generating step of the m2 solution method is only accountable for a minimal part of the total solution time, and is therefore not the reason for m2's high solution times. However, the time spent on pre-generation grows considerably as the size of the instance increases, 2.8 seconds for D-20 vs. 545.6 seconds for D-40, and is expected to scale badly to even larger instances (see Table B.10). Additionally, m2 has to be run with a maximum path limit for the D-20 and D-40 test instances, as described in Section 7.1.3. This means that the solutions derived from these test instances cannot guarantee an optimal solution to the problem even if m2 were to run until completion of the solution process. The majority of time m2 uses is spent on the MIP process and so this part of m2 can be held accountable for its bad scalability. For m2, the availability requirement validation is done outside the pre-generation, this means that m2 has to search through a solution space including every path combination, valid or not, in its solution process. As a result, m2 has to consider a lot more possibilities for each problem instance, compared to m3, which can exclude all availability-invalid path combinations in its pre-generating step. Even though m2 may not be the most efficient solution method to the CSBQANR problem, the solutions it produces with its limitations on both input and time are quite close to the column generation methods and m3 in test instances M-6, S-20, S-40 and D-20, as seen in figures 7.6 and 7.7. In the D-40 test instance, the solutions resulting from m2 are not as good as the column generation methods and m3, as seen in Figure 7.8.

The m3 solution method solves small and medium problem instances within the 10 hour time limit, as seen by its average time consumption in Table 7.5. Runs with m3 for all β values for the M-6 and S-20 test instances are solved to optimum within the time limit, and some of the highest β values for the other instances are solved within 10 hours, but for D-20 and D-40 optimality cannot be guaranteed. As for the m2 solution method, m3 has a maximum path limit when solving D-20 and D-40, described in Section 7.1.3. The maximum path limits handled by m3 is notably higher than the maximum path limits for m2 (see Table 7.2), thus m3 appears to scale better than m2. Even though m3, like m2, is not able to guarantee optimal solutions to the most complex test instances, m3 still produces better solutions than any other solution method for all instances, including the test instances where the pre-generated number of mappings is restricted. For larger or more complex problem instances than the D-40 test instance, m3 will require further restrictions to the number of pre-generated mappings, and the value of the solutions m3 is able to produce is expected to suffer accordingly.

The column generation solution methods solves every test instance except D-40 within the 10 hour limit. The solution times of cg1 show that it handles all test instances up to S-40 very well, but solving the D-20 test instance the solution time of cg1 increases considerably. A limit to the maximum number of paths for each service provider pair for D-40 is needed for cg1 to function in practice, as described in Section 7.1.3. When cg1 is run with this restriction, the column generation is completed within 1 hour for all β values except $\beta = 0.25$. The full solution process is completed for the case with $\beta = 1.0$, but for every other β value it is terminated upon reaching the 10 hour time limit.

Disregarding the D-40 test instance, the solution times of cg2 and cg3 are very close and in the same range, every test case is then solved within 6 minutes. For the D-40 test instance, cg2 is the only solution method able to finish within the 10 hour MIP time limit for every β value, although cg2 does exceed the 1 hour column generation time limit for some of the β values. The solution method cg3 is terminated because of the time limit for $\beta = 0.0$, but manages to finish with the time limit for the rest of the β values. For $\beta = 0.75$ and $\beta = 1.0$ both cg2 and cg3 use less than 1 hour to solve the

D-40 test instance. Figure 7.13 shows the solution time of cg2 and cg3 when run with test instances S-40 and D-20.

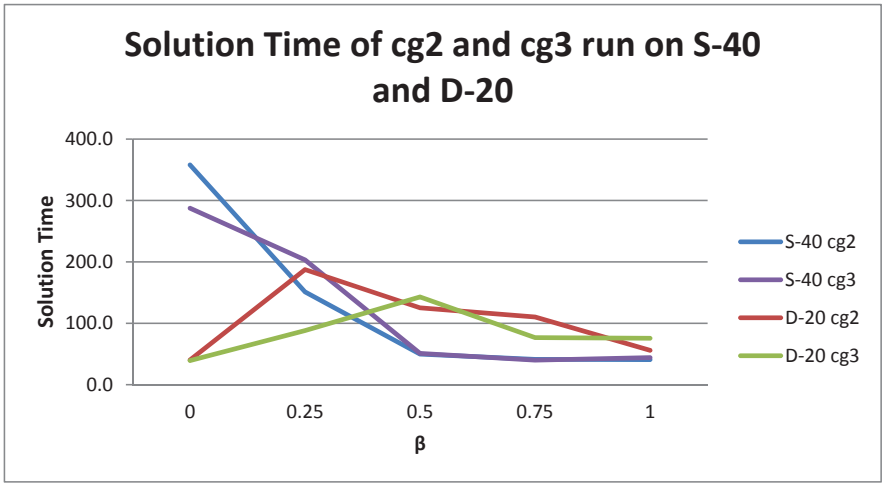


FIGURE 7.13: Solution time for running cg2 and cg3 with S-40 and D-20 for every value of β

For $\beta = 0.0$ the solution times for cg2 and cg3 when run with the D-20 instance are much lower than when they are run on the S-40 test instance, unlike every other solution method. At $\beta = 0.25$ the solution times differ, but they are closer than with $\beta = 0.0$. For the higher β values, the solution times from solving the D-20 test instance is generally larger than from solving S-40, as is the tendency of the rest of the solution methods. The number of possible disjoint paths in the S-40 test instance is low because of its sparse quality, this in the combination with its high number of demanded services makes the disjoint requirements very restricting for this instance. As the D-20 test instance can provide many more disjoint paths and has a much lower number of demanded services, the disjoint requirements are less restricting in this instance. As mentioned in the β discussion in Section 7.2.2, low β values encourage sharing in the LP solution, which can increase the solution time if the disjoint requirements are very restricting. Considering the mentioned properties of S-40 and D-20, this can explain the large differences in solution time for these instances with low β values.

Comparing all the solution methods, as seen in Table 7.5 the column generation solution methods solves every test instance in average faster than m2 and m3 and reviewing the complete results presented in Appendix B one can see that the column generation solution methods are always faster than m3 when m3 is run with unlimited number of

paths. For all test instances, except D-40, all the column generation solution processes are able to finish within the 10 hour time limit. For the larger test instances D-20 and D-40, the solution times of cg2 and cg3 are considerably lower than those of cg1 for D-20, while the solution time is similar for D-40 only if using limited pre-generation for cg1. For the other test instances the solution times are in the same range. However, as discussed in Section 7.2.1, cg1 stands out and always provides better values than cg2 and cg3. Solution values of cg1 is generally close to the values provided by the m3 solution method, as seen in figures 7.6 to 7.8.

A considerable part of the solution time when running cg1, cg2 and cg3 originates from the column generation part of the solution methods, prior to solving the MIP with the generated columns, see Table B.11 and Table B.5. The cg1 solution method has the lowest times spent on column generation for S-20 and S-40, and is close to cg2 and cg3 for M-6. However for D-20, cg1 spends considerably more time than cg2 and cg3, approaching the 1 hour time limit set for column generation. For D-40, the number of pre-generated paths needs to be restricted for cg1 to produce meaningful results within a reasonable amount of time. With restricted pre-generated input to cg1, the time consumption is slightly lower than those of cg2 and cg3, only exceeding the 1 hour time limit for $\beta = 1.0$, while cg2 and cg3 exceed this time limit for β values 0.0, 0.25 and 0.5. For cg1 to finish within the same time as cg2 and cg3, as it does for D-40, it is expected that the path limit for pre-generation for cg1 has to be even stricter as the problem instances get larger and more complex. For D-40, the restriction imposed still allows cg1 to produce better solutions than cg2 and cg3, but this is not expected to be the general tendency for larger and more complex instances. The solution methods cg2 and cg3 are the only solution methods that manage to finish the solution process for D-40 for some β values without any added restrictions, and is therefore assumed to offer the best scalability as problem instances get larger and more complex.

The pre-generation of paths and mappings performed for solution methods m2, m3 and cg1, is only accountable for a small portion of the solution methods' total time consumption for solving the test instances, but there are still some results worth noting regarding the scalability for the pre-generation methods. For M-6, S-20, S-40 and D-20, the time spent on pre-generation never exceeds 3 seconds for m2 and m3, and is

a maximum of 18.3 seconds for cg1. For D-40, the time spent on pre-generation is considerably longer, approximately 545 seconds, 960 seconds and 2808 seconds for the pre-generation of input for m2, m3, and cg1, respectively (see Table B.10). This indicates that, even though pre-generation of data is a small portion of the total solution time for the test instances used, this portion is expected to grow considerably as problem instances get larger.

Both the pre-generation of paths and mappings, as well as the different column generation methods are all done separately for each service provider pair, and can therefore be done in parallel for each such pair. For instances M-6, S-20, S-40 and to some extent D-20, little gain is expected from implementing parallelisation, as the total time spent for pre-generation and column generation is considered small compared to the total solution time. However, for D-40, the time spent on both pre-generation of paths and column generation grows considerably. As problem instances get larger and more complex, the proportion of time spent for pre-generation or column generation is expected to increase, and parallel implementation of the pre-generation and column generation methods are expected to improve scalability considerably. Table B.10 found in Appendix B shows the time used on pre-generation for m2, m3 and cg1 and Table B.11 shows the time used for column generation for cg1, cg2 and cg3.

7.2.4 Sources of Errors

The different solution methods are implemented using different programming languages and also use different kinds of input files. Where m3 uses the raw JSON files and calculates paths and mappings directly, m2 uses the JSON files as input for a pre-generating step, similar to the one used by m3, but then exports the data to a Mosel data file before handing it to the PFM implementation of m2. This affects the availability requirement validation of the different solution methods, the result being that one solution method, m2, accepts solutions that other methods do not.

This behaviour appears in the S-40 instance, where m2's best found solution is better than the optimal solution found by m3. Since m2 and m3 solve the same problem, this should not be possible. Careful study of the solutions produced show that m2

uses two combinations of primary and backup paths that are not included as possible mappings for m3. The two combinations connect the same customer and provider, but are allocated to two different services. The path combinations use the same links through the network, and are therefore identical in terms of availability. Additionally, the two services both have an expected availability requirement of 0.995. The path combinations give an expected availability of exactly 0.995, when using the input data pre-generated for m2. However, during m3's mapping pre-generating step, the expected availability for the equivalent mappings is calculated to 0.994999833409, and those mappings are therefore not included as possible mappings for the m3 solution method. Hard-coding the implementation of m3 to accept the two mapping equivalents of the infeasible path combinations from m2 does produce a solution with a value between m2's best found solution value and best bound, implying that these kinds of borderline solutions is the root cause of m2's behaviour for this instance. Similarly, hard-coding m2 to not allow these two path combinations, as well as similar borderline path combinations, does indeed make m2 produce a solution in line with m3's original solution, with a best found solution value below, and a best bound above, the optimal solution found by m3. These results can be found in Table B.15 in Appendix B.

All tests done in this thesis are run on machines on the Solstorm cluster, which is a cluster shared with other students and staff associated to the Department of Industrial Economics and Technology Management. This means that there is no guarantee that one gets all of the computing power and memory available at a node during the full duration of a test. It is possible to roughly monitor resource usage on computing nodes in the cluster, but small to medium sized tasks, run simultaneously, may have gone undetected and had a small effect on the solution times obtained. However, every possible precaution has been made to prevent collisions on the cluster. Using a remote cluster in this manner can include influence from uncontrollable external factors as well. This all can affect the solution times obtained through the tests executed in the work with this thesis, so solution times presented here might be imprecise.

The column generation solution methods are implemented so that they keep the optimal basis found for each iteration, reloading the same basis once new mapping columns have been found and added before starting the next iteration. When the LP and MIP

root node solution methods are restricted to use primal simplex only, Xpress reports the warning *'BCL: Warning 1570: XPRS: ?140 Warning: Basis lost - recovering'*, when the MIP process is started following the initial column generation. The solutions produced seem to be unaffected by this warning, but the cause of this warning message may have had minor effects on the total solution time reported for solution methods cg1, cg2 and cg3.

Chapter 8

Conclusion and Future Work

This chapter starts with some concluding remarks of the work presented in this thesis, followed by some suggested future work related to this thesis is presented.

8.1 Conclusions

The next generation of brokers will be the QoS-aware broker with cloud connectivity that takes requirements from the customers. Today's telecommunications companies, with emerging initiatives as cloud brokers, can benefit greatly from implementing this approach. Such a broker must not only decide placement of services, but also how to provide the connectivity between services and customers. This approach also provides the opportunity to offer cloud services with connectivity guarantees, bundled with generic Internet connectivity, as telecommunication companies already act as Internet service providers in multiple regions.

There are multiple existing solutions solving parts of the challenges involved in acting as a BC, like network routing handling simple QoS attributes as path latency or bandwidth flow. Some approaches reward low average latency to some extent, but do not consider individual services' requirements, and other approaches have strict latency requirements, while others allow high latency routings. Approximating availability in network routing has been attempted and explored extensively in literature, but no known

solutions perform both cloud resource location (e.g. service to provider) and connectivity with more advanced QoS attributes including availability requirements for each individual service.

In this thesis, three MIP models attempting to address the optimisation problem where cloud service allocation and connectivity challenges are met by a joined broker and carrier role, considering three intrinsically different QoS requirements (bandwidth, latency and availability), have been presented. Derived from the models are six solution methods, these have been tested using five test instances and different values of a scaling parameter β. The solution method m1 is a simple link-flow model implemented in Mosel with no pre-generation. The solution method m2 is a path-flow model implemented in Mosel with a path pre-generation method providing paths as input to the Mosel model. The solution method m3 is a mapping model implemented in C++ with BCL using a mapping pre-generation method to generate path combinations as input. The three column generation methods cg1, cg2 and cg3 use different column generation methods to provide good mappings to the C++ implementation of the mapping model. The β parameter is used to vary the degree of sharing of backup paths allowed and consequently the amount of backup capacity to be reserved. The five test instances are constructed to be used in all solution methods and have different number of customers and services, and varying network structure.

As discussed in Section 7.2.1, the m1 solution method solves a more restricting problem compared to the other solution methods and so produces lower solution values and serves fewer customers. In problem instances with sparse networks, m1 will avoid reserving capacity for backup use. Because of no or little backup capacity provisioning, solutions obtained by m1 are not effected by β value differences for the test instances used. However, solution times of m1 are low, compared to the other solution methods, because of smaller solution space due to added restrictions to the allowed network routing for each service, as well as the simpler modelling of the availability requirement.

Using path and mapping pre-generation methods, m2 and m3 are the implementations of the PFM and mapping model presented in Chapter 6. The solution methods are supposed to be equivalent solution models, the difference being what decision are made

in the pre-generating algorithms and in the MIP models of the two solution methods. When solved to optimality m2 and m3 provide the same solutions, and when terminated prematurely m3 provides equal or higher solution values and consistently provides a smaller gap. The exception is the S-40 test instance, where m2 produces better solutions than m3, but for this instance, m2 includes combinations of primary and backup paths not allowed by m3. This is due to numerical differences in rounding of availability values when storing input data for m2 and the availability values used in the mapping pre-generating for m3.

The column generation solution methods are heuristic solution methods shown to provide quite good solutions. The best solutions are provided by cg1, having solution values on average within 98.1% of the best found solution for each test instance and β value. The solution method cg1 depends on possible paths being pre-generated, while cg2 and cg3 do not. The solution times for cg1 run on smaller test instances are low, but the time spent on generating columns increases considerably when the problem instances become larger and more complex. In these cases, cg2 and cg3 are the faster solution methods, and cg1 is required to use a limited set of pre-generated paths to offer solution times in the same range. The quality of the mapping columns generated by the column generation solution methods was tested against the columns generated by the pre-generating of m3, when the pre-generating was restricted to generate approximately the same number of mappings, by selecting the shortest paths in terms of number of links. The column generation methods are shown to produce better mapping columns than the restricted pre-generation for the D-40 test instance. For D-20, which has less customers and services, only cg1 consistently produces better sets of mapping columns than produced by the restricted pre-generation. For D-20, cg2 and cg3 produces solutions similar to m3 with restricted pre-generation.

The use of β was intended to regulate how the amount of backup capacity needed to be reserved is affected by multiple overlapping backup paths. The general tendency is that increasing value of β leads to problems easier to solve, having on average lower solution time values than with lower β values. This effect is less apparent in some test instances and for some cases with the column generation solution methods, but can be seen very clearly in the m1, m2 and m3 for most test instances. Reviewing the ratio between

backup costs and solution values for a selection of the test cases one can see that using β has the expected effect as the need for backup increases as its value increases. When $\beta = 1.0$ and dedicated backup is required, it seems that for some cases the need for backup decreases instead of increases, but this effect can be attributed to the exclusion of customers, no longer being profitable with increased backup needs, from the overall solution.

Changing the values of input parameters as the availability requirements of each service in a problem instance affects the complexity of the problem considerably. Solution values of every solution method increases as the availability requirements become less strict, reducing the need for backup paths and reducing the amount of services excluded due to availability requirements. Similarly, increasing the availability requirements disqualifies a lot of services from being included in the solution and reduces the solution values for every solution method. Solution method m1 is especially sensitive to the variations in availability requirement, as the strict link-disjoint requirement to a service's primary and backup paths used in m1 limits the number of services with backup needs it is possible to serve. For the column generation solution methods solution times are reduced both when lowering and increasing the availability requirements. Lowering the availability requirement removes backup considerations from both the column generation and the MIP model, while making the requirements higher greatly reduces the number of columns generated. For m2, the number of pre-generated paths is independent of the availability requirement, thus high requirements increase the solution time, as it gets harder to find combinations of paths fulfilling the availability requirements, while lowering the requirements makes it easier. For m3, low availability requirements will increase the solution time, while higher requirements decrease the solution time. This is due to the availability requirements being handled by the pre-generating for m3 and the number of mappings generated depends on the availability requirements. For high availability requirements, fewer possible mappings are found, reducing the complexity of the problem, while for low availability requirements, more possible mappings are found, increasing the number of mappings to be evaluated by m3. For m1, no noticeable effect in terms of solution time for the different levels of availability requirements is evident, which can be attributed to its simplified availability handling and to the fact that no pre-generating is used.

Regarding the scalability of the solution methods presented in this thesis, one finds that m1 is efficient in problem instances with sparse networks or low number of services, but is outperformed, in terms of solution time, for the test instances larger or more complex than the S-40 test instance. The solution times of m2 are quite high all-over and m2 seems to scale very badly, and despite having quite good solution values, is not suitable on problem instances larger or more complex than M-6 using more than 10 hours solving every other test instance. Solution method m3 scales better than m2, but still has trouble solving test instances larger or more complex than S-40 without restricting the number of paths generated. However, restricting the number of paths to match the number of mappings produced by the column generation methods, m3 solves quite fast and still produces good solutions. Regarding cg1, it scales well up till the size and complexity of S-40, and has tolerable solution time values with the D-20 test instance. However, to be able to find a meaningful solution for D-40, cg1 needs to restrict the path pre-generating because of its column generation step otherwise using several hours on each iteration. The solution methods cg2 and cg3 scales better than cg1, being the only solution methods able to solve D-40 within the time limit for most of the β values. Actually, cg2 is the only solution method that manages to solve every test case within the 10 hour limit for the MIP process, but does exceed the 1 hour column generation time limit for some β values.

All in all, m3, with restrictions for larger problem instances or one of the column generation methods cg1, cg2 and cg3 seem to be the solution methods to recommend. For larger problem instances than presented in this thesis, cg2 and cg3 are expected to be a better choice than cg1.

8.2 Future Work

In this section some suggestions are presented for continuation of the work regarding the CSBQANR problem presented in this thesis, the models derived to represent it, the solution methods derived to solve it, and other areas which could improve its use.

8.2.1 Problem Extensions

As the CSBQANR problem presented in this thesis only considers the case where one service is connected to one provider location, which makes it most applicable for the SaaS service model which is the area most telecommunication companies are focusing on today. For the PaaS and IaaS service models, having multiple components included in a service that require some form of interoperability is more common, so an extension to the problem allowing separate placements of these components might be interesting in addressing these service models. This adds the need to handle connectivity between components as well as between customers and providers. The resulting problem including these extensions would resemble a quadratic assignment problem where the assignment of one component affects the assignment of others.

Three QoS attributes have been included in the work presented in this thesis, all differing in their nature and behaviour. These attributes have been chosen due to their importance to business customers, frequency of occurrence in the literature and their difference in behaviour and modelling approaches. An extension to the problem could include other QoS attributes directly or by generalising the problem so different attributes can be handled and thus increasing the problem's flexibility and area of application. Other attributes can be jitter (important for streaming and buffer size), error rate, node processing requirements (e.g. many small packages vs. a few big), redundancy, security considerations etc.

8.2.2 Extensions to the Solution Methods

The column generation solution methods proposed in this thesis can only produce heuristic solutions to the CSBQANR problem. By incorporating column generation in the MIP solution process, optimal solutions may potentially be achieved, depending on the column generation method used. Although performing reasonably well, the column generation methods of cg2 and cg3 are heuristic methods, and cannot guarantee that an optimal set of mapping columns is found. The solution method cg1, on the other hand, can guarantee that an LP-optimal set of mapping columns is found, but is dependent on having all possible paths pre-generated, and does not scale as well as cg2 and cg3. Extending the column generation methods presented in this thesis, or developing new

column generation methods, offering mapping columns that are optimal or close to optimal and not dependent on pre-generation, could be interesting in the continuation of this work.

In the pre-generation and column generation methods, the generation of columns is done independently for each service provider pair, and thus these methods contain considerable potential for parallelisation. For large problem instances, parallel implementations of pre-generation and column generation are expected to produce noticeable reductions in total solution time, and could be part of a continuation of this work.

8.2.3 Numerical Study of the Effect of β

The β parameter used by all the solution methods presented in this thesis determines the degree of sharing of the protection scheme, from fully shared protection ($\beta = 0.0$) to dedicated protection ($\beta = 1.0$). With unlimited sharing scheme, the actual expected availability of a network connection will depend on all other connections having a backup path overlapping with that connection's backup path, as the overlapping part of the backup path may be unavailable if a link failure affects the primary path of one of the sharing connections. When there are few such overlapping paths, their effect on the expected availability is very small, but if the number of overlapping connections is large, the effect may be significant. The test instances used in this thesis do not produce solutions where the number of overlapping backup paths is large, and the models presented in this thesis do not explicitly handle this part when estimating the expected availability. However, a non-zero β value can be used to reduce the effect from other overlapping network connections, increasing the amount of bandwidth reserved for backup as the number of overlapping backup paths increases, reducing the probability of a backup path's resources being unavailable. A continuation of the work presented in this thesis could include a numerical study of the effect different β values would have for problem instances where the number of overlapping backup paths is large. The effect different β values has on the actual availability of primary and backup path mappings would be especially interesting in such a study.

8.2.4 Real World Implementation and Applications

The CSBQANR problem presented in this thesis is designed to be of help for a BC in the tactical and possibly strategic decision making process. The obvious application is to decide what customers to choose and how to fulfil their demands, but the model could also be used with other intentions. The output of the solution methods is dependent on the input data, and the solutions derived are highly dependent on the accuracy of this data, as indicated by the variation in solutions by varying the availability requirement, as seen in Section 7.2.2. Constructing problem instances with different network structures or other aspects, and running these using one or several of the presented solution methods can be used to see how solutions change conditional on the input used, i.e. using the solution methods to perform a kind of simulation. In this fashion the BCs could use the CSBQANR solution methods to explore aspects as placement of provider locations, where further development, improvement and expansion of the existing network is needed, what kind of services are the most profitable and what effect different price structures could have.

The CSBQANR problem presented in this thesis has a highly simplified structure and complexity compared to the real world problem, where only three QoS attributes are considered, and availability is only approximated. The use of more and better information about the present cloud computing market and how it is expected to evolve could be valuable in improving this problem and the solving of it. Other important aspects for the experienced QoS attributes may be identified and incorporated in the problem's network routing. Prices, network structure, availability and latency values, customer placement and demand and so forth are all estimated in this thesis. To be able to support real world implementation, real world data on these aspects is needed to provide more realistic input to the solution methods.

Technology in the cloud computing and virtualisation areas is developing fast, but as of today some of the aspects of the presented problem and solutions may not be implementable. In the future, one should explore different network virtualisation approaches to implement the network embedding produced by the results of solving the problem. The CSBQANR problem presented may produce network embedding relying on rapid backup switching, which is not supported by all approaches.

The development of SLAs to ensure the guarantees of the QoS attributes proposed in this thesis is a challenge the actors in the market have to address to be able to utilise the CSBQANR solution methods to their full extent. If such SLAs are not possible to attain, the models should be altered to fit the actual setting and match what is possible to obtain.

Bibliography

Bari, M. F., Boutaba, R., Esteves, R., Granville, L. Z., Podlesny, M., Rabbani, M. G., Zhang, Q., and Zhani, M. F. Data center network virtualization: A survey. *Communications Surveys Tutorials, IEEE*, 15(2):909–928, 2013. ISSN 1553-877X. doi: 10.1109/SURV.2012.090512.00043.

Baroncelli, F., Martini, B., and Castoldi, P. Network virtualization for cloud computing. *Annals of telecommunications - Annales des télécommunications*, 65(11-12):713–721, 2010.

Braaten, S. and Holmen, M. Cloud brokering with quality aware network routing. NTNU, 2013. Project Thesis in the subject TIØ4500 Managerial Economics and Operations Research, Specialization Project.

Clemente, R., Bartoli, M., Bossi, M. C., D'orazio, G., and Cosmo, G. Risk management in availability sla. In *Design of Reliable Communication Networks, 2005.(DRCN 2005). Proceedings. 5th International Workshop on*, pages 8–pp. IEEE, 2005.

Cui, W., Stoica, I., and Katz, R. H. Backup path allocation based on a correlated link failure probability model in overlay networks. In *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*, pages 236–245. IEEE, 2002.

Gonzalez, A. personal communication, 2014.

Guo, T., Wang, N., Moessner, K., and Tafazolli, R. Shared backup network provision for virtual network embedding. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–5. IEEE, 2011.

Han, L. Market acceptance of cloud computing: An analysis of market structure, price models and service requirements. Technical report, Bayreuther Arbeitspapiere zur Wirtschaftsinformatikk, No. 42, 2009.

Irnich, S. and Desaulniers, G. Shortest path problems with resource constraints. In *G. Desaulniers, J. Desrosiers, M. M. Solomon, eds. Column Generation*, pages 33–65. Springer, 2005.

Józsa, B. G. and Orincsay, D. Shared backup path optimization in telecommunication networks. In *Proc. of Design of Reliable Communication Networks Workshop (DRCN)*, pages 251–257, 2001.

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., and Leaf, D. Nist cloud computing reference architecture. In *Services (SERVICES), 2011 IEEE World Congress on*, pages 594–596, 2011. doi: 10.1109/SERVICES.2011.105.

Mell, P. and Grance, T. The NIST definition of cloud computing (draft). *NIST special publication*, 800(145):7, 2011.

Schupke, D. A. Guaranteeing service availability in optical network design. In *Asia-Pacific Optical Communications Conference (APOC)*. International Society for Optics and Photonics, 2005.

Schupke, D. A. and Rambach, F. A link-flow model for dedicated path protection with approximative availability constraints. *IEEE communications letters*, 10(9):679–681, 2006.

Svaet, S. W., Undheim, A., and Castejón, H. N. Cloud carrier: Present and future, Telenor research internal report. January 2013.

Undheim, A., Swaet, S. W., and Solsvik, F. H. SLA requirement for cloud brokers, Telenor research internal report. September 2012.

ITU, I. T. U. ITU focus group on cloud computing - part 1. 2012.

VMWare Staff. Virtualization overview. *White Paper, http://www.vmware.com/pdf/virtualization.pdf*, 2012, Accessed: 2014-02-03.

Zhang, J., Zhu, K., Zang, H., and Mukherjee, B. A new provisioning framework to provide availability-guaranteed service in wdm mesh networks. In *Communications, 2003. ICC'03. IEEE International Conference on*, volume 2, pages 1484–1488. IEEE, 2003.

Zhou, L. and Grover, W. A theory for setting the ”safety margin” on availability guarantees in an sla. In *Design of Reliable Communication Networks, 2005.(DRCN 2005). Proceedings. 5th International Workshop on*, pages 7–pp. IEEE, 2005.

Appendix A

Complete MIP Models

This appendix provides a summary of the MIP models introduced in Chapter 6.

A.1 Link-Flow Model

A.1.1 Indices, Sets, Parameters and Variables

Table A.1: Summary of the indices to the link-flow model

Index	Definition
c	Customer
i, j	Network node
p	Provider
s	Service

TABLE A.2: Summary of the sets to the link-flow model

Set	Definition
$\mathscr{A}$	Set of all arcs (i, j)
$\mathscr{C}$	Set of all customers c
$\mathscr{I}_c$	Set of internal nodes from the perspective of customer c
$\mathscr{Q}_j$	Set of every node reachable from node j
$\mathscr{O}$	Set with every combination of different services (s, t) where $s < t$
$\mathscr{P}_s$	Set of all providers who can host service s
$\mathscr{S}_c$	Set of services demanded by customer c
$\mathscr{W}_j$	Set of every node that can reach node j

The nodes defining the arcs in $\mathscr{A}$ are numbered increasingly, so when a reference to links is needed, a link being the undirected link between two nodes or both arcs connecting the nodes, this is defined as $\forall(i, j) \in \mathscr{A}|i < j$. Similarly, the services are ordered, so when requiring all pairs of different services $\mathscr{O}$, the set of all combinations of services (s, t) where $s < t$ is used.

TABLE A.3: Summary of the parameters to the link-flow model

Parameter	Definition
B_s^U, B_s^D	Bandwidth requirement for service s in each direction (U: up, customer to provider, D: down, provider to customer)
D_{ij}^L	(Expected) availability for link between i and j, where $D_{ij}^L = D_{ji}^L$
E_{ij}	Price per capacity unit used on arc between nodes i and j
G_s	Maximum round trip latency allowed by service s
H_{sp}	Cost of placing service s at provider p
I_{ij}	The amount of capacity available for reservation on arc (i,j)
J_p	Node for provider p
R_c	Revenue from serving customer c
T_{ij}	(Expected) latency on arc from node i to node j
V_c	Node for customer c
Y_s	Required minimum availability level for service s
β	Factor regulating the amount of extra capacity that must be reserved for an increased number of services sharing a backup arc

TABLE A.4: Summary of the decision variables of the link-flow model

Variable	Definition
b_{ijs}	Binary variables that indicate if service s uses arc (i, j) as a part of the backup path from customer node to selected provider node (and (j, i) is used for the backup path in the opposite direction). The variables are not defined for arcs going into the customer node corresponding to service s
ℓ_{st}	Binary variable indicating if the chosen primary paths of services s and t overlap anywhere
r_{sp}	Binary variable indicating if a backup path is needed for service s to provider p
u_{ijs}	Binary variables that indicate if service s uses arc (i, j) as part of its primary path from customer node to selected provider node (and (j, i) is used for the primary path in the opposite direction). The variables are not defined for arcs going into the customer node corresponding to service s
x_{sp}	Binary variable that indicates if service s is placed at provider p
y_c	Binary variable that indicates if customer c is being served
λ_{ij}	Continuous variable indicating how much capacity to be reserved on arc (i, j) for backup use

A.1.2 Constraints

$$\begin{aligned} \max z \quad = \quad & \sum_{c \in \mathscr{C}} R_c y_c \\ & - \sum_{c \in \mathscr{C}} \sum_{s \in \mathscr{S}_c} \sum_{p \in \mathscr{P}_s} H_{sp} x_{sp} \\ & - \sum_{(i,j) \in \mathscr{A}} E_{ij} (\lambda_{ij} + \sum_{c \in \mathscr{C}} \sum_{s \in \mathscr{S}_c} (B_s^U u_{ijs} + B_s^D u_{jis})) \end{aligned} \tag{6.1}$$

$$y_c - \sum_{p \in \mathscr{P}_s} x_{sp} = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.2}$$

$$\sum_{c \in \mathscr{C}} \sum_{s \in \mathscr{S}_c} (B_s^U u_{ijs} + B_s^D u_{jis}) + \lambda_{ij} \leq I_{ij}, \quad \forall (i,j) \in \mathscr{A} \tag{6.3}$$

$$\sum_{i \in \mathscr{W}_j} u_{ijs} - \sum_{i \in \mathscr{Q}_j} u_{jis} = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall j \in \mathscr{I}_c \tag{6.4}$$

$$\sum_{i \in \mathscr{W}_j} b_{ijs} - \sum_{i \in \mathscr{Q}_j} b_{jis} = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall j \in \mathscr{I}_c \tag{6.5}$$

$$\sum_{j \in \mathscr{Q}_{V_c}} u_{V_c j s} - y_c = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.6}$$

$$\sum_{j \in \mathscr{Q}_{V_c}} b_{V_c j s} - \sum_{p \in \mathscr{P}_s} r_{sp} = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.7}$$

$$\sum_{i \in \mathscr{W}_{J_p}} u_{i J_p s} - \sum_{j \in \mathscr{Q}_{J_p}} u_{J_p j s} - x_{sp} = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s \tag{6.8}$$

$$\sum_{i \in \mathscr{W}_{J_p}} b_{i J_p s} - \sum_{j \in \mathscr{Q}_{J_p}} b_{J_p j s} - r_{sp} = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s \tag{6.9}$$

$$\sum_{(i,j)\in\mathscr{A}} T_{ij}(u_{ijs}+u_{jis}) \le G_s, \quad \forall c\in\mathscr{C},\ \forall s\in\mathscr{S}_c,\ \forall p\in\mathscr{P}_s \tag{6.10}$$

$$\sum_{(i,j)\in\mathscr{A}} T_{ij}(b_{ijs}+b_{jis}) \le G_s, \quad \forall c\in\mathscr{C},\ \forall s\in\mathscr{S}_c,\ \forall p\in\mathscr{P}_s \tag{6.11}$$

$$\sum_{p\in\mathscr{P}_s} M_s^A r_{sp} + \sum_{(i,j)\in\mathscr{A}} \ln(D_{ij}^L)\ u_{ijs} - \ln Y_s \ge 0, \quad \forall c\in\mathscr{C},\ \forall s\in\mathscr{S}_c \tag{6.12}$$

$$M_s^A = \ln Y_s - \sum_{(i,j)\in\mathscr{A}} \ln D_{ij}^L, \quad \forall c\in\mathscr{C},\ \forall s\in\mathscr{S}_c$$

$$B_s^U b_{ijs} + B_s^D b_{jis} - \lambda_{ij} \le 0, \quad \forall c\in\mathscr{C},\ \forall s\in\mathscr{S}_c,\ \forall (i,j)\in\mathscr{A} \tag{6.13}$$

$$\sum_{c\in\mathscr{C}}\sum_{s\in\mathscr{S}_c} \beta(B_s^U b_{ijs} + B_s^D b_{jis}) - \lambda_{ij} \le 0, \quad \forall (i,j)\in\mathscr{A} \tag{6.14}$$

$$b_{ijs} + u_{ijs} \le 1, \quad \forall c\in\mathscr{C},\ \forall s\in\mathscr{S}_c,\ \forall (i,j)\in\mathscr{A} \tag{6.15}$$

$$r_{sp} - x_{sp} \le 0, \quad \forall c\in\mathscr{C},\ \forall s\in\mathscr{S}_c \forall p\in\mathscr{P}_s \tag{6.16}$$

$$u_{ijs} + u_{jis} + u_{ijt} + u_{jit} - \ell_{st} \le 1, \quad \forall (i,j)\in\mathscr{A} | i<j,\ \forall (s,t)\in\mathscr{O} \tag{6.17}$$

$$b_{ijs} + b_{jis} + b_{ijt} + b_{jit} + \ell_{st} \leq 2, \quad \forall (i,j) \in \mathscr{A} | i < j, \ \forall (s,t) \in \mathscr{O}$$

(6.18)

$$\begin{aligned}
y_c &\in \{0,1\}, \quad \forall c \in \mathscr{C} \\
x_{sp} &\in \{0,1\}, \quad \forall c \in \mathscr{C}, \ \forall s \in \mathscr{S}_c, \ \forall p \in \mathscr{P}_s \\
r_{sp} &\in \{0,1\}, \quad \forall c \in \mathscr{C}, \ \forall s \in \mathscr{S}_c, \ \forall p \in \mathscr{P}_s \\
u_{ijs} &\in \{0,1\}, \quad \forall c \in \mathscr{C}, \ \forall s \in \mathscr{S}_c, \ \forall (i,j) \in \mathscr{A} \mid j \neq V_c \\
b_{ijs} &\in \{0,1\}, \quad \forall c \in \mathscr{C}, \ \forall s \in \mathscr{S}_c, \ \forall (i,j) \in \mathscr{A} \mid j \neq V_c \\
\ell_{st} &\in \{0,1\}, \quad \forall (s,t) \in \mathscr{O} \\
\lambda_{ij} &\geq 0, \quad \forall (i,j) \in \mathscr{A}
\end{aligned}$$

(6.19)

A.2 Path-Flow Model

A.2.1 Indices, Sets, Parameters and Variables

Table A.5: Summary of the indices to the Path-Flow Model

Index	Definition
c	Customer
i, j	Network node
k, b	Path
p	Provider
s, t	Service

Table A.6: Summary of the sets to the Path-Flow Model

Set	Definition
$\mathscr{A}$	Set of all arcs (i, j)
$\mathscr{C}$	Set of all customers c
$\mathscr{K}$	Set of all paths k
$\mathscr{K}_{sp}^{SP}$	Set of all paths k where service s is placed at provider p
$\mathscr{L}_{ij}$	Set of all paths k using the arc (i, j)
$\mathscr{O}$	Set with every combination of different services (s, t) where $s < t$
$\mathscr{P}_s$	Set of all providers p who can host service s
$\mathscr{S}_c$	Set of services s demanded by customer c

The nodes defining the arcs in $\mathscr{A}$ are numbered increasingly, so when a reference to links is needed, a link being the undirected link between two nodes or both arcs connecting the nodes, this is defined as $\forall(i, j) \in \mathscr{A} | i < j$. Because paths in this model use the same links in both directions, the set $\mathscr{L}_{ij}$ is symmetric, i.e. $\mathscr{L}_{ij} = \mathscr{L}_{ji}$. Additionally, the services are ordered, so when requiring all pairs of different services $\mathscr{O}$, the set of all combinations of services (s, t) where $s < t$ is used.

TABLE A.7: Summary of the parameters to the Path-Flow Model

Parameter	Definition
D_k^P	Availability for path k
D_{kb}^C	Probability of both primary path k and backup path b being available simultaneously
E_{ij}	The cost of reserving a unit of capacity for backup on arc (i, j)
E_k^P	The cost of using a path k as primary path
I_{ij}	The amount of capacity available for reservation on arc (i,j)
R_c	Revenue from serving customer c
U_{ijk}^P	The amount of capacity used by path k on arc (i,j)
Y_s	Required minimum availability level for service s
β	Factor regulating the amount of extra capacity that must be reserved for an increased number of services sharing a backup arc

TABLE A.8: Summary of the variables to the Path-Flow Model

Variable	Definition
f_{ijs}	Binary variable indicating if there is a need for backup capacity reservation on arc (i, j) for service s
ℓ_{st}	Binary variable indicating if the chosen primary paths of services s and t overlap anywhere
o_{kb}	Binary variable indicating if the primary and backup path combination (k, b) is chosen
q_{ijs}	Positive continuous variable displaying the amount of capacity needed to be reserved for backup on arc (i, j) for service s. Equals zero if f_{ijs} is zero
u_k	Binary variable indicating if path k is chosen as a primary path
v_k	Binary variable indicating if path k is chosen as a backup path
x_{sp}	Binary variable indicating if service s is provided by provider p
y_c	Binary variable indicating if customer c is being served
λ_{ij}	Positive continuous variable displaying the amount of capacity reserved on arc (i, j)

A.2.2 Constraints

$$\max z \quad = \quad \sum_{c \in \mathscr{C}} R_c y_c - \sum_{k \in \mathscr{K}} E_k^P u_k - \sum_{(i,j) \in \mathscr{A}} E_{ij} \lambda_{ij} \tag{6.20}$$

$$y_c - \sum_{p \in \mathscr{P}_s} x_{sp} = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.21}$$

$$x_{sp} - \sum_{k \in \mathscr{K}_{sp}^{SP}} u_k = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s \tag{6.22}$$

$$\sum_{k \in \mathscr{L}_{ij}} U_{ijk}^P u_k + \lambda_{ij} \leq I_{ij}, \quad \forall (i,j) \in \mathscr{A} \tag{6.23}$$

$$\sum_{p \in \mathscr{P}_s} \sum_{k \in \mathscr{K}_{sp}^{SP}} D_k^P (u_k + v_k) - \sum_{p \in \mathscr{P}_s} \sum_{k \in \mathscr{K}_{sp}^{SP}} \sum_{b \in \mathscr{K}_{sp}^{SP}} D_{kb}^C o_{kb} - Y_s y_c \geq 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.24}$$

$$q_{ijs} - M_{ijs}^B f_{ijs} \leq 0, \quad \forall (i,j) \in \mathscr{A},\ \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.25}$$

$$M_{ijs}^B \geq U_{ijk}^P, \quad \forall (i,j) \in \mathscr{A},\ \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s,\ \forall k \in \mathscr{K}_{sp}^{SP} \cap \mathscr{L}_{ij}$$

$$\sum_{p \in \mathscr{P}_s} \sum_{k \in \mathscr{K}_{sp}^{SP} \cap \mathscr{L}_{ij}} U_{ijs}^P (v_k - u_k) - q_{ijs} \leq 0, \quad \forall (i,j) \in \mathscr{A},\ \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.26}$$

$$q_{ijs} - \lambda_{ij} \leq 0, \quad \forall (i,j) \in \mathscr{A},\ \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.27}$$

$$\beta \sum_{c \in \mathscr{C}} \sum_{s \in \mathscr{S}_c} q_{ijs} - \lambda_{ij} \leq 0, \quad \forall (i,j) \in \mathscr{A} \tag{6.28}$$

$$u_k + v_b - o_{kb} \leq 1, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s,\ \forall k \in \mathscr{K}_{sp}^{SP},\ \forall b \in \mathscr{K}_{sp}^{SP} \tag{6.29}$$

$$\sum_{p \in \mathscr{P}_s} \sum_{k \in \mathscr{K}_{sp}^{SP} \cap \mathscr{L}_{ij}} u_k + \sum_{p \in \mathscr{P}_s} \sum_{k \in \mathscr{K}_{tp}^{SP} \cap \mathscr{L}_{ij}} u_k - \ell_{st} \leq 1, \quad \forall (i,j) \in \mathscr{A} | i < j,\ \forall (s,t) \in \mathscr{O} \tag{6.30}$$

$$f_{ijs} + f_{ijt} + \ell_{st} \leq 2, \quad \forall (i,j) \in \mathscr{A},\ \forall (s,t) \in \mathscr{O} \tag{6.31}$$

$$\begin{aligned}
y_c &\in \{0,1\}, \quad \forall c \in \mathscr{C} \\
x_{sp} &\in \{0,1\}, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s \\
u_k &\in \{0,1\}, \quad \forall k \in \mathscr{K} \\
v_k &\in \{0,1\}, \quad \forall k \in \mathscr{K} \\
o_{kb} &\in \{0,1\}, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c,\ \forall p \in \mathscr{P}_s,\ \forall k \in \mathscr{K}_{sp}^{SP},\ \forall b \in \mathscr{K}_{sp}^{SP} \\
\ell_{st} &\in \{0,1\}, \quad \forall (s,t) \in \mathscr{O} \\
f_{ijs} &\in \{0,1\}, \quad \forall (i,j) \in \mathscr{A},\ \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \\
q_{ijs} &\geq 0, \quad \forall (i,j) \in \mathscr{A},\ \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \\
\lambda_{ij} &\geq 0, \quad \forall (i,j) \in \mathscr{A}
\end{aligned} \tag{6.32}$$

A.3 Mapping Model

A.3.1 Indices, Sets, Parameters and Variables

Table A.9: Summary of the indices to the mapping model

Index	Definition
c	Customer
i, j	Network node
m	Mapping
k, b	Path
s, t	Service

Table A.10: Summary of the sets to the mapping model

Set	Definition
$\mathscr{A}$	Set of all arcs (i, j)
$\mathscr{C}$	Set of all customers c
$\mathscr{K}$	Set of all paths k
$\mathscr{L}_{ij}$	Set of all paths using the arc (i, j)
$\mathscr{M}$	Set of all mappings m
$\mathscr{M}_k^B$	Set of all mappings m using path k as backup path
$\mathscr{M}_k^P$	Set of all mappings m using path k as primary path
$\mathscr{M}_s^S$	Set of all mappings m that can be used by a service s
$\mathscr{O}$	Set with combination of all different services (s, t) where $s < t$
$\mathscr{S}_c$	Set of services demanded by customer c

The nodes defining the arcs in $\mathscr{A}$ are numbered increasingly, so when links are needed, a link being the undirected link between two nodes or both arcs connecting the nodes, this is defined as $\forall(i, j) \in \mathscr{A} | i < j$. Because paths in this model use the same links in both directions, the set $\mathscr{L}_{ij}$ is symmetric, i.e. $\mathscr{L}_{ij} = \mathscr{L}_{ji}$. Additionally, the services are ordered, so when requiring all pairs of different services $\mathscr{O}$, the set of all combinations of services (s, t) where $s < t$ is used.

TABLE A.11: Summary of the parameters to the mapping model

Parameter	Definition
E_{ij}	The cost of reserving a unit of capacity for backup on arc (i, j)
E_k^P	The cost of using a path k
F_{ijm}	Indicates whether mapping m needs backup capacity on arc (i,j) or not. Equal to 1 if Q_{ijm} is larger than 0, and 0 otherwise
I_{ij}	The amount of capacity available for reservation on arc (i,j)
Q_{ijm}	The amount of capacity needed to be reserved for backup on arc (i,j) by mapping m. Equal to the maximum of zero and the difference between backup and primary capacity demand if backup and primary paths overlap on (i,j)
R_c	Revenue from serving customer c
U_{ijm}^M	The amount of capacity used by mapping m on arc (i,j)
β	Factor regulating the amount of extra capacity that must be reserved for an increased number of services sharing a backup arc

TABLE A.12: Summary of the variables to the mapping model

Variable	Definition
ℓ_{st}	Binary variable indicating if the chosen primary paths of services s and t overlap anywhere
w_m	Binary variable indicating if mapping m is chosen
y_c	Binary variable indicating if customer c is being served
λ_{ij}	Positive continuous variable showing the amount of capacity reserved on arc (ij)

A.3.2 Constraints

$$\max z \quad = \quad \sum_{c \in \mathscr{C}} R_c y_c - \sum_{k \in \mathscr{K}} E_k^P \sum_{m \in \mathscr{M}_k^P} w_m - \sum_{(i,j) \in \mathscr{A}} E_{ij} \lambda_{ij} \tag{6.33}$$

$$y_c - \sum_{m \in \mathscr{M}_s^S} w_m = 0, \quad \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.34}$$

$$\sum_{m \in \mathscr{M}_s^S} U_{ijm}^M w_m + \lambda_{ij} \leq I_{ij}, \quad \forall (i,j) \in \mathscr{A} \tag{6.35}$$

$$\sum_{m \in \mathscr{M}_s^S} Q_{ijm} w_m - \lambda_{ij} \leq 0, \quad \forall (i,j) \in \mathscr{A},\ \forall c \in \mathscr{C},\ \forall s \in \mathscr{S}_c \tag{6.36}$$

$$\beta \sum_{m \in \mathscr{M}} Q_{ijm} w_m - \lambda_{ij} \leq 0, \quad \forall (i,j) \in \mathscr{A} \tag{6.37}$$

$$\sum_{k \in \mathscr{L}_{ij}} \sum_{m \in \mathscr{M}_k^P \cap \mathscr{M}_s^S} w_m + \sum_{k \in \mathscr{L}_{ij}} \sum_{m \in \mathscr{M}_k^P \cap \mathscr{M}_t^S} w_m - \ell_{st} \leq 1, \quad \forall (i,j) \in \mathscr{A} | i < j,\ \forall (s,t) \in \mathscr{O} \tag{6.38}$$

$$\sum_{k \in \mathscr{L}_{ij}} \sum_{m \in \mathscr{M}_k^B \cap \mathscr{M}_s^S} F_{ijm} w_m + \sum_{k \in \mathscr{L}_{ij}} \sum_{m \in \mathscr{M}_k^B \cap \mathscr{M}_t^S} F_{ijm} w_m + \ell_{st} \leq 2, \quad \forall (i,j) \in \mathscr{A},\ \forall (s,t) \in \mathscr{O} \tag{6.39}$$

$$\begin{aligned} y_c &\in \{0,1\}, \quad \forall c \in \mathscr{C} \\ w_m &\in \{0,1\}, \quad \forall m \in \mathscr{M} \\ \ell_{st} &\in \{0,1\}, \quad \forall (s,t) \in \mathscr{O} \\ \lambda_{ij} &\geq 0, \quad \forall (i,j) \in \mathscr{A} \end{aligned} \tag{6.40}$$

Appendix B

Computational Test Results

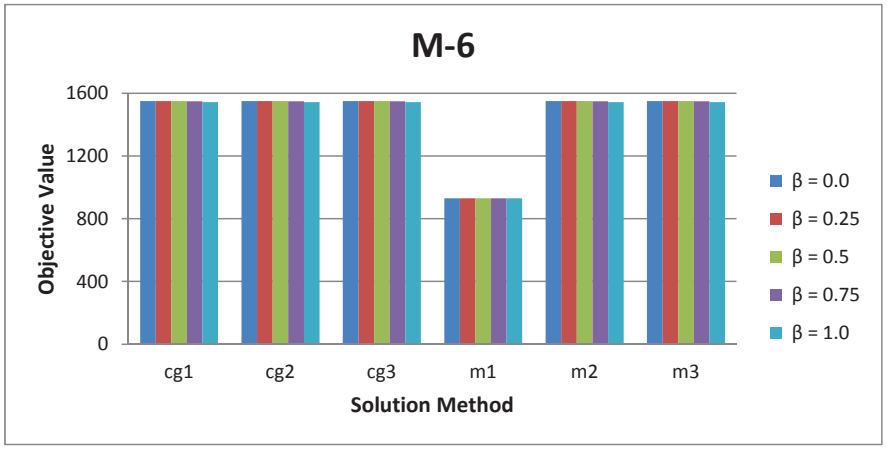


FIGURE B.1: Solution values for the M-6 test instance run on every combination of solution method and β

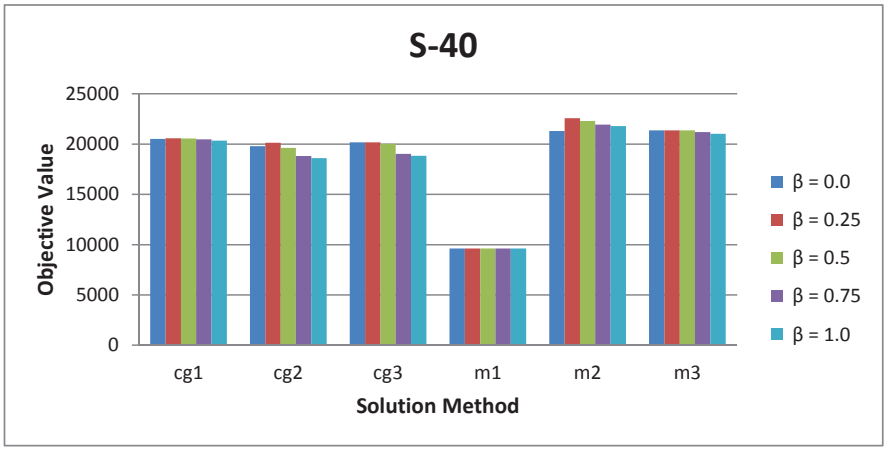


FIGURE B.2: Solution values for the S-40 test instance run on every combination of solution method and β

TABLE B.1: Complete set of results for every solution method and every test instance for $\beta = 0.0$
*These solution methods are run with a maximum limit on the number of paths pre-generated presented in Table B.6

		$\beta = 0.0$		
Test Instance	Method	Solution Value	Total Time	MIP Time
M-6	cg1	1550	0.2	0.1
	cg2	1550	0.1	0.0
	cg3	1550	0.1	0.0
	m1	930	0.1	0.1
	m2	1550	457.8	457.7
	m3	1550	0.6	0.5
S-20	cg1	10970	2.9	1.2
	cg2	11130	10.9	1.0
	cg3	11130	10.0	0.8
	m1	4780	1.4	1.1
	m2	11170	36001.9	36001.3
	m3	11170	27.7	11.3
S-40	cg1	20508	168.4	26.5
	cg2	19803	358.0	108.4
	cg3	20173	287.4	29.1
	m1	9610	33.4	29.7
	m2	21298	36003.9	36001.3
	m3	21358	36025.5	36001.6
D-20	cg1	9707	5089.1	1923.0
	cg2	9281	40.3	4.6
	cg3	9199	39.3	7.4
	m1	4990	2549.0	2548.4
	m2*	9382	36004.3	36001.3
	m3*	9941	36159.5	36001.1
D-40	cg1*	16052	42345.6	36000.8
	cg2	15257	10970.5	7292.3
	cg3	14980	39690.7	36002.4
	m1	7665	37508.6	37502.0
	m2*	13651	36550.4	36001.2
	m3*	16267	36898.9	36003.4

TABLE B.2: Complete set of results for every solution method and every test instance for $\beta = 0.25$
*These solution methods are run with a maximum limit on the number of paths pre-generated presented in Table B.6

		$\beta = 0.25$		
Test Instance	Method	Solution Value	Total Time	MIP Time
M-6	cg1	1550	0.3	0.1
	cg2	1550	0.1	0.0
	cg3	1550	0.1	0.0
	m1	930	0.2	0.1
	m2	1550	343.4	343.4
	m3	1550	0.5	0.5
S-20	cg1	10970	3.3	1.2
	cg2	10840	9.0	1.3
	cg3	11130	18.3	0.8
	m1	4780	1.5	1.3
	m2	11130	36002.4	36001.8
	m3	11170	27.0	10.6
S-40	cg1	20588	156.6	37.5
	cg2	20128	151.2	27.4
	cg3	20173	203.3	81.4
	m1	9610	19.8	16.4
	m2	22582.5	36003.7	36001.1
	m3	21357.5	36025.3	36001.8
D-20	cg1	9787	5466.9	2570.1
	cg2	9216	187.4	25.5
	cg3	9267	88.1	17.6
	m1	4990	14065.6	14065.0
	m2*	9382	36003.3	36000.3
	m3*	9941	36096.2	36005.0
D-40	cg1*	16122	42423.2	36002.5
	cg2	15022	16976.9	13346.1
	cg3	15221	13812.1	6089.9
	m1	5926	36013.7	36006.7
	m2*	13459	36550.9	36001.7
	m3*	16279	36885	36002.7

TABLE B.3: Complete set of results for every solution method and every test instance for $\beta = 0.5$
*These solution methods are run with a maximum limit on the number of paths pre-generated presented in Table B.6

		$\beta = 0.5$		
Test Instance	Method	Solution Value	Total Time	MIP Time
M-6	cg1	1550	0.3	0.1
	cg2	1550	0.1	0.0
	cg3	1550	0.1	0.0
	m1	930	0.2	0.2
	m2	1550	138.1	138.0
	m3	1550	0.7	0.6
S-20	cg1	11110	3.8	1.1
	cg2	10890	7.9	1.1
	cg3	10940	8.0	1.1
	m1	4780	1.6	1.3
	m2	11170	36003.5	36002.9
	m3	11170	27.8	12.0
S-40	cg1	20548	124.5	63.1
	cg2	19633	49.8	7.8
	cg3	20028	50.9	6.8
	m1	9610	20.2	16.6
	m2	22307.5	36003.0	36000.4
	m3	21357.5	36026.1	36001.6
D-20	cg1	9876	3764.2	164.4
	cg2	9464	125.2	14.8
	cg3	9476	142.9	12.6
	m1	4990	30607.0	30606.4
	m2*	9372	36003.2	36000.2
	m3*	9905	9537.5	9438.8
D-40	cg1*	16241.5	42193.9	36000.5
	cg2	15290	6130.5	2380.6
	cg3	15380	19027.3	15358.0
	m1	5876	36032.2	36025.5
	m2*	13761	36550.6	36001.5
	m3*	16293.5	36900.7	36003.2

TABLE B.4: Complete set of results for every solution method and every test instance for $\beta = 0.75$
*These solution methods are run with a maximum limit on the number of paths pre-generated presented in Table B.6

		$\beta = 0.75$		
Test Instance	Method	Solution Value	Total Time	MIP Time
M-6	cg1	1549	0.4	0.3
	cg2	1549	0.1	0.0
	cg3	1549	0.1	0.0
	m1	930	0.2	0.1
	m2	1549	64.8	64.8
	m3	1549	0.4	0.4
S-20	cg1	10963	2.8	0.9
	cg2	10888	9.6	1.2
	cg3	10928	6.9	0.9
	m1	4780	1.6	1.3
	m2	11048	36001.7	36001.1
	m3	11048	25.0	8.6
S-40	cg1	20478	46.7	10.1
	cg2	18808	41.2	4.9
	cg3	19028	40.0	5.1
	m1	9610	20.4	17.0
	m2	21928	36003.3	36000.7
	m3	21198	36024.8	36001.4
D-20	cg1	9644	2985.8	19.2
	cg2	9302	110.3	5.5
	cg3	9416	76.7	4.2
	m1	4990	27059.7	27059.1
	m2*	9246	36003.3	36000.3
	m3*	9654	8132.1	7973.1
D-40	cg1*	16124	41550.2	36000.8
	cg2	14496	2877.6	541.7
	cg3	14901	3527.7	1276.1
	m1	5966	36060.6	36053.6
	m2*	13638.5	36550.4	36001.2
	m3*	16172	37389.8	36003.7

TABLE B.5: Complete set of results for every solution method and every test instance for $\beta = 1.0$
*These solution methods are run with a maximum limit on the number of paths pre-generated presented in Table B.6

		$\beta = 1.0$		
Test Instance	Method	Solution Value	Total Time	MIP Time
M-6	cg1	1543	0.1	0.0
	cg2	1543	0.1	0.0
	cg3	1543	0.1	0.0
	m1	930	0.1	0.1
	m2	1543	121.4	121.4
	m3	1543	0.2	0.1
S-20	cg1	10550	1.6	0.2
	cg2	10215	6.3	0.3
	cg3	10500	4.7	0.2
	m1	4780	1.8	1.5
	m2	10115	36001.7	36001.1
	m3	10550	18.9	2.8
S-40	cg1	20338	30.6	6.7
	cg2	18608	40.8	4.5
	cg3	18838	44.1	4.2
	m1	9610	19.9	16.2
	m2	21787.5	36003.7	36001.2
	m3	21028	14394.0	14369.5
D-20	cg1	9544	3986.9	0.9
	cg2	9000	56.0	0.5
	cg3	9302	75.6	0.7
	m1	4990	4634.3	4633.7
	m2*	9102	29558.9	29555.9
	m3*	9544	109.3	5.9
D-40	cg1*	15917	4426.7	80.6
	cg2	15020	2046.9	77.3
	cg3	14913	2605.8	49.2
	m1	7003	36018.6	36011.6
	m2*	13451	36549.9	36000.8
	m3*	15992	3033.8	1672.95

TABLE B.6: Maximum limits for paths for running cg1, m2 and m3 on D-20 and D-40

	D-20	*D-40*
cg1	-	100
m2	5	4
m3	20	10

TABLE B.7: All results from m3 with maximum path limit per service provider pair, which are set so the pre-generation generates the number of mappings equal to column generation methods' average and maximum number of mappings

Instance	Path Limit	β	Solution Value	Solution Time
D-20	5	0.00	9382.0	9.8
		0.25	9382.0	9.1
		0.50	9372.0	9.5
		0.75	9245.5	9.1
		1.00	9102.0	3.9
	6	0.00	9524.0	16.0
		0.25	9524.0	28.1
		0.50	9514.0	14.2
		0.75	9380.5	9.2
		1.00	9206.0	5.1
D-40	4	0.00	13907.0	1408.3
		0.25	13907.0	1741.2
		0.50	13881.5	1456.8
		0.75	13691.0	1342.1
		1.00	13475.0	583.2
	5	0.00	14915.0	$> 10h$
		0.25	14955.0	$9.1h$
		0.50	14920.5	$> 10h$
		0.75	14778.2	9214.3
		1.00	14587.0	601.0

TABLE B.8: Results for test instance S-20 with low, original and high availability requirements run with every solution method for $\beta = 0.0$ and $\beta = 0.5$

		$\beta = 0.0$			$\beta = 0.5$		
	Method	Sol. Value	Tot. Time	MIP Time	Sol. Value	Tot. Time	MIP Time
Low	cg1	16270	1.3	0.0	16270	1.3	0.0
	cg2	16270	1.6	0.1	16270	1.6	0.1
	cg3	16270	2.5	0.1	16270	2.5	0.1
	m1	16290	1.4	1.1	16290	1.4	1.1
	m2	16290	6.3	5.8	16290	4.5	4.0
	m3	16290	37.6	3.8	16290	38.4	3.7
S-20	cg1	10970	2.9	1.1	11110	3.8	1.1
	cg2	10840	38.8	1.4	10990	14.5	1.2
	cg3	11020	26.6	0.9	10950	17.6	0.9
	m1	4780	1.4	1.1	4780	1.6	1.3
	m2	11170	36001.9	36001.3	11130	36001.6	36001.0
	m3	11170	27.7	11.3	11170	27.8	12.0
High	cg1	1970	0.8	0.1	2020	1.4	0.5
	cg2	1625	0.9	0.1	2063	2.3	0.1
	cg3	1625	0.9	0.1	2063	2.3	0.1
	m1	480	1.2	0.9	480	1.3	1.0
	m2	2090	36001.3	36000.8	2090	36003.8	36003.3
	m3	2090	1.2	0.9	2090	0.9	0.6

TABLE B.9: Profit (solution value) and Cost assosiated with backup (backup cost) for S-20 and S-40 run with every solution method and for every β value

		$\beta = 0.0$		$\beta = 0.25$		$\beta = 0.5$		$\beta = 0.75$		$\beta = 1.0$	
		Profit	Cost	Profit	Cost	Profit	Cost	Profit	Cost	Profit	Cost
S-20	cg1	10970	1085	10970	1085	11110	1270	10963	1253	10550	795
	cg2	11130	1410	10840	1320	10890	1650	10888	1683	10215	1360
	cg3	11130	1440	11130	1440	10940	1630	10928	1613	10500	1315
	m2	11170	1370	11130	1440	11170	1370	11048	1493	10115	900
	m3	11170	1370	11170	1370	11170	1370	11048	1493	10550	880
S-40	cg1	20508	660	20588	645	20548	650	20478	760	20338	860
	cg2	19803	540	20128	615	19633	630	18808	710	18608	905
	cg3	20173	630	20173	610	20028	670	19028	590	18838	780
	m2	21298	625	22583	610	22308	610	21928	745	21788	825
	m3	21358	610	21358	605	21358	605	21198	770	21028	935

TABLE B.10: Time spent on pre-generation of paths for m2, mappings for m3 and paths for cg1 for every test instance in seconds

	Path pre-gen m2	Mapping pre-gen m3	Path pre-gen cg1
M-6	0.01	0.01	0.00
S-20	0.12	0.13	0.11
S-40	0.06	0.06	0.06
D-20	2.78	2.70	18.29
D-40	545.56	960.92	2808.01

TABLE B.11: Time used in the column generation steps of cg1, cg2 and cg3 for every test instance for $\beta = 1.0$

	$\beta = 1.0$		
	cg1	cg2	cg3
M-6	0.12	0.05	0.05
S-20	1.72	7.62	8.71
S-40	74.75	95.85	98.06
D-20	3304.52	93.41	75.74
D-40	2959.23	3069.32	3974.07

TABLE B.12: The number of mappings produced with cg1, cg2 and cg3 for the D-20 test instance for every value of β, as well as the maximum number of mappings for each β and solution method, and the maximum number of mappings for each β

		$\beta = 0.0$	$\beta = 0.25$	$\beta = 0.5$	$\beta = 0.75$	$\beta = 1.0$	Average
D-20	cg1	329	385	400	314	352	356.0
	cg2	208	269	275	246	248	249.2
	cg3	239	279	337	286	305	289.2
Average		258.7	311.0	337.3	282.0	301.7	298.1
Max		329	385	400	314	352	

TABLE B.13: The number of mappings produced by the pre-generation of m3 for test instance D-20 by restricting the number of paths for each service provider pair

D-20	Path Limit	4	5	6	7	8
	Number of Mappings	180	294	448	675	826

TABLE B.14: The number of mappings produced by the pre-generation of m3 for test instance D-40 by restricting the number of paths for each service provider pair

D-40	Path Limit	4	5	6
	Number of Mappings	555	818	1184

TABLE B.15: Solution values and best bounds for the S-40 test instance run with m2 and m3, as well as with the hard-coded version of m2 not allowing the illegal path combinations found in the original m2 solution, and with the hard-coded version of m3 accepting the illegal path combinations found in the original m2 solution

	$\beta = 1.0$	
Solution Method	Solution Value	Best Bound
m2	21787.5	28231.6
Hard-Coded m2	20217.5	28598.4
m3	21027.5	21028.9
Hard-Coded m3	22582.5	26216.5

Appendix C

Visualised Data Editor

The test instances used for testing of the solution methods presented in Chapter 6 for solving the CSBQANR problem described in Chapter 5, have been created and edited using a web-application, referred to as the Visualised Data Editor (VDE), developed specifically for that purpose. The VDE allows for creating, editing, exporting, and importing of problem instances.

Implementation

The VDE is developed using HTML5 and javascript, and supports current versions of all major web browsers. The VDE uses two third party javascript libraries, *jQuery* and *knockout.js*, both open source and released under the *MIT license*.

Data Formats

Two data formats are supported for import and export by the VDE; JSON and Mosel data format. Exported Mosel data is compatible with the Mosel implementation of the LFM, presented in Section 6.2. Exported JSON data is taken as input to an independent application (implemented in C++), implementing pre-generation of paths and mapping, with export to Mosel data format, as well as the BCL version of the mapping model in Section 6.4 and the column generation methods of Section 6.5. Importing and exporting of data files is done through copy/paste from/to a text box due to technical limitations in a pure HTML5 and javascript application.

Usage

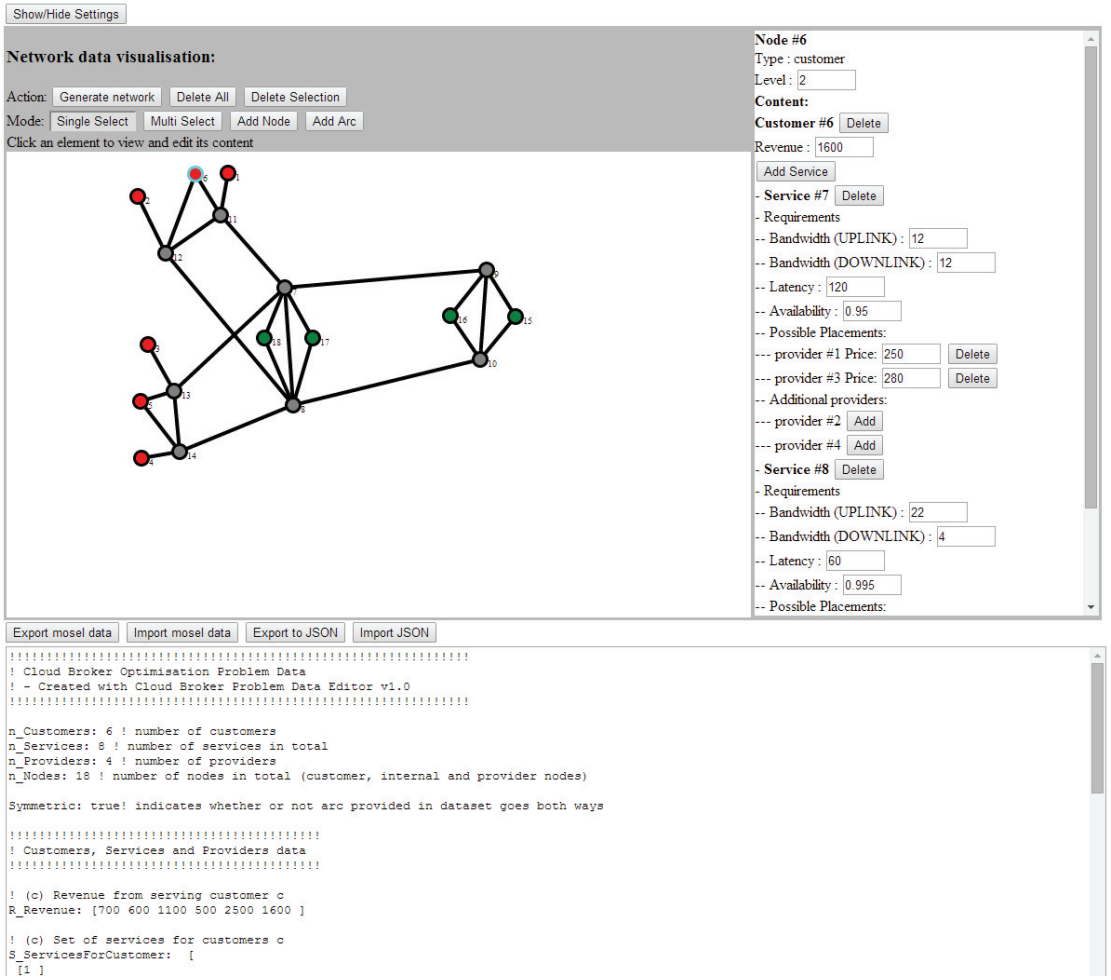


FIGURE C.1: Screenshot of the Problem Data Editor displaying the different regions when a customer node is selected

The application can be launched by opening the 'index.htm' file of the data editor in your web browser of choice. As seen in Figure C.1, the interface of the VDE consists of 4 main regions. The upper left region functions as a toolbar for performing functions such as deleting all or selected data, generating new random data and changing the current mode of the editor window. The middle left region is the visual editor of the data editor. Here a visual representation of the instance is presented, where arcs and nodes can be selected, added, removed and repositioned. The lower left region includes the text box for import and export of data, as well as buttons for import and export actions. The right region lists - and allows editing of - information about the network elements selected in the visual editor. For arcs this is simply the specific characteristics of that arc, for nodes this information indicates if the specified node represents a provider, customer or neither,

and displays info of the node including all info of services. This information region can also be used for editing the parameters of the network with buttons for adding the customer or provider property to a node, adding services to a customer, adding eligible providers to a service and changing values as revenue, bandwidth requirements, latency and availability requirements. For links in the network its bandwidth capacity, expected latency, expected availability and the bandwidth price can be altered.

In addition to the 4 main regions of the user interface, there is one additional region, which is hidden as default, consisting of settings for random data generation. Here, the number of node levels, nodes per cluster, customers and providers can be set as well as the maximum number of services per customer, proportion of eligible providers per service, the average arc latency and the average service latency requirement. By the use of the button *Generate network*, an auto-generated network is output to the visualisation area, corresponding to the generation settings with the use of a random function to provide some diversification in the network parameters.

Appendix D

Cloud Broker Application

D.1 Implementation

The application has been developed with the C++ programming language. Two non-standard C libraries have been used. An open source library released under the *FreeBSD* license, *libjson*, is used for importing of json data files to the application. To interact with the Xpress Optimisation Suite, the proprietary *Xpress-BCL Builder Component Library* (BCL) has been used.

The main functionality of the application can be divided in to three main components; the Pre-generation-, Mapping Model- and Column Generation Components. In addition to these components, the application also includes logic for file input and output, task execution and configuration logic, but is not considered main components of the application.

The pre-generation component takes a data object, representing a problem instance, as input and can generate paths and mappings according to Algorithm 1 and Algorithm 2, respectively. This component is independent of the other application components, and its output can be used for the Mapping Model Component or to output Mosel data files compatible with Mosel implementations of the PFM and the mapping model.

The Mapping Model Component builds and solves the mapping model of Section 6.4 using BCL. The model can be solved either as LP or MIP. This component can also return dual values of its current solution as well as output a textual representation of the solution. This component is dependent on being used in combination with either the Pre-generation component or the Column Generation Component to produce meaningful results.

The Column Generation Component implements three different column generation methods for the Mapping Model Component; Brute Force, SPPRC Heuristic A and SPPRC Heuristic B. This component depends on the Mapping Model Component for solving the LP of the problem and provide dual values of the LP-solution.

D.2 Compiling

An executable of the application is available in the delivery of this thesis. This executable is compiled for CentOS 6.5 x86 64-bit Linux with Xpress Suite version 7.6.0. For systems with different specifications, a recompile may be needed.

The application is developed for Linux operating systems and does contains some platform specific code, and as a result a Linux required for compiling and using the application. In addition, the mapping model implementation (Section 6.4) is developed for Xpress 7.6.0, but should be compatible with other Xpress versions supporting the same BCL API. However, a different version of Xpress will require a recompile specific for that system.

Compilation of the application is done through the use of a *Makefile*. From the source code root directory, simply perform the command *make* to compile the application. The default executable name is *cloudbroker-bcl*. For some system configurations, a recompile of the *libjson* library may also be needed. This is done by performing the *make* command in the libjson sub directory of the source code.

D.3 Usage

The application supports three different actions, listed in Table D.1 for easy reference.

TABLE D.1: The actions supported by the BCL application

Action	Description
mdata	Generates a path and mapping model Mosel implementation compatible data file for the provided input JSON data
solve	Solves the BCL implementation of the mapping model for the provided input JSON data, with mappings pregenerated
cgsolve	Solves the BCL implementation of the mapping model for the provided input JSON data, using column generation to provide mappings

The application uses the following command pattern for execution

```
<executable name> <action> <options>
```

Note that the application will need the appropriate permissions to be executed. This can be done by running the following command from the directory of the application.

```
chmod 755 cloudbroker-bcl
```

Assuming an open terminal window located at the same folder as the executable, using the default name, with a problem instance in JSON format data.json. The following command will generate a Mosel compatible data file with pre-generated paths and mappings with the name moseldata.txt for the input JSON data data.json.

```
./cloudbroker-opt mdata -i data.json -o moseldata.txt
```

Similarly, the following command executed to solve the problem using the mapping model with all mapping pre-generated and storing the result to result.txt:

```
./cloudbroker-opt solve -i data.json -o result.txt
```

To solve the problem using column generation, and storing the result to cgresult.txt and building the model with a β model parameter value of 0.5, the following command is executed:

```
./cloudbroker-opt cgsolve -i data.json -o cgresult.txt -beta 0.5
```

Note that the specification of the location of the executable, as seen here with the './' prefix, is not needed if the executable is added to the operating system's path.

The application has a number of supported optional settings to alter the behaviour, a complete list is provided in Table D.2. The settings may be provided in any order, should any setting be given multiple times for a single action, the last value given for the setting is used.

TABLE D.2: Complete list of all supported options for the bcl application.
* optional for actions *solve* and *cgsolve*, required for action *mdata*

Option	Type	Description	Default	Required
-i	string	Name of input data file	n/a	Yes
-o	string	Name of output data file for any result from selected action	n/a	No*
-beta	real	β value used in the mapping model	0.25	No
-maxtime	integer	Maximum allowed time spent on the MIP or LP solution process in seconds. Ignored if set to $= 0$	0	No
-maxpaths	integer	Maximum allowed number of paths pre-generated for each service provider pair. Ignored if set to ≤ 0	-1	No
-cgmaxtime	integer	Maximum allowed time spent on the column generation process in seconds. Ignored if set to ≤ 0	-1	No
-cgmaxiters	integer	Maximum allowed number of iterations for the column generation process. Ignored if set to ≤ 0	-1	No
-cgmaxcount	integer	Maximum allowed number of mappings generated from the column generation process. Ignored if set to ≤ 0	-1	No
-cgalg	integer	Column generation method: 1: Brute Force 2: SPPRC heuristic A 3: SPPRC heuristic B	3	No
-nomapping	n/a	If flag is set, the pre-generation only generates paths and not mappings	n/a	No
-alg	string	Algorithm for LP/intial MIP LP: ' ': default (concurrent solve) 'd': dual simplex algorithm 'p': primal simplex algorithm 'b': Newton barrier algorithm	' '	No

Appendix E

Mosel Source Code

This appendix includes the source code of the Mosel implementation of the LFM presented in Section 6.2 and the PFM presented in Section 6.3.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!
!! Carrier Broker Optimisation: Link Flow Model
!!
!!- The following script implements the LFM defined in Section 6.1
!!  of Mari Holmen's and Sindre Møgster Braaten's masters thesis
!!
!!- Authors: Mari Holmen and Sindre Møgster Braaten
!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

model CarrierBrokerLFM

options explterm
options noimplicit
uses "mmxprs", "mmsystem";

parameters
    ! Data file to read from
    Data = 'data/s2-mosel-link.txt';
    ! Minimum proportion of total backup requirement reserved on an arc (aka. beta)
    MinBackupProportion = 0.25;
    ! Time limit for runtime,  maximum number of seconds for optimisation
    TimeLimit = -1;
end-parameters

writeln("Model Parameters:");
writeln("Data:", Data);
writeln("MinBackupProportion(beta):", MinBackupProportion);
writeln("TimeLimit:", TimeLimit);

declarations
    timetracker:    real; ! used to log timestamps for time consumption output
end-declarations

writeln("Building model...");
timetracker := timestamp;

!setparam("XPRS_presolve", 0);  ! uncomment to turn of presolve
if(TimeLimit>0.0) then
    setparam("XPRS_maxtime", TimeLimit);
end-if

setparam("XPRS_verbose", true); ! Turn on message printing
setparam("XPRS_MIPLOG", 2); ! 2: print information for each solution found
                            !(ALT: 0: no log, 1: summary in end, 3: log each node, -N: log every Nth node)
!!!!!!!!!!!!!!!!!!!!!!!!
! SETS
!!!!!!!!!!!!!!!!!!!!!!!!

declarations
! Set sizes
    n_Customers:            integer; ! number of customers
    n_Services:             integer; ! number of services
    n_Providers:            integer; ! number of providers
    n_Nodes:                integer; ! number of nodes in total

! Sets
    Customers:              set of integer;
    Providers:              set of integer;
    ! Used as shorthand for 'cc in Customers, ss in S_ServicesForCustomer(cc)' when cc is not needed
    Services:               set of integer;
    ! Set of nodes in the network.
    ! - First we have the customer nodes, then the internal nodes, the the provider nodes.
    Nodes:                  set of integer;
    ! Set of internal nodes in the network + all customer nodes
    ! - usage for internal nodes for each customer cc: 'nn in I_Nodes | nn<>cc'
    I_Nodes:                set of integer;
end-declarations

initializations from Data
    n_Customers;
    n_Services;
    n_Providers;
    n_Nodes;
end-initializations

Customers:= 1..n_Customers;
Services:= 1..n_Services;
Providers:= 1..n_Providers;
```

```
Nodes:= 1..n_Nodes;
I_Nodes:= 1..(n_Nodes-n_Providers);

finalize(Customers);
finalize(Services);
finalize(Providers);
finalize(Nodes);
finalize(I_Nodes);

! INDEXED SETS

declarations
    ! set of services of for each customer cc
    S_ServicesForCustomer:      set of set of integer;
end-declarations

initialisations from Data
    S_ServicesForCustomer;
end-initialisations

!!!!!!!!!!!!!!!!!!!!!!
! PARAMETERS
!!!!!!!!!!!!!!!!!!!!!!

declarations
! Parameters
    ! Price per used capacity between nodes
    K_CapPrice:                 dynamic array(Nodes,Nodes) of real;
    ! R_Revenue from serving each customer
    R_Revenue:                  dynamic array(Customers) of real;
    ! Price of placing a service at a provider
    H_PlacePrice:               dynamic array(Services,Providers) of real;
    ! Latency requirement for each service from customer to provider
    G_LatencyReq:               array(Services) of real;
    ! Bandwidth requirement for each service from customer to provider
    B_BandwidthReqUp:               array(Services) of real;
    ! Bandwidth requirement for each service from provider to customer
    B_BandwidthReqDown:         array(Services) of real;
    ! Minimum avarage availability for each service
    Y_AvailabilityReq:          array(Services) of real;
    ! Lateny between each pair of nodes
    T_LinkLatency:              dynamic array(Nodes,Nodes) of real;
    ! Bandwidth capacity between each pair of nodes
    F_BandwidthCap:             dynamic array(Nodes,Nodes) of real;
    ! Expected availability for each owned link between each pair of nodes
    D_AvailabilityExp:          dynamic array(Nodes,Nodes) of real;
    ! Node for each provider
    E_ProviderNode:             set of integer;

! Network data interpretation configuration
    Symmetric:              boolean;

end-declarations

initialisations from Data
    K_CapPrice;
    R_Revenue;
    H_PlacePrice;
    G_LatencyReq;
    B_BandwidthReqUp;
    B_BandwidthReqDown;
    Y_AvailabilityReq;
    T_LinkLatency;
    F_BandwidthCap;
    D_AvailabilityExp;

    Symmetric;
end-initialisations

! Provider nodes are the n_Providers last nodes in network
E_ProviderNode:=(n_Nodes-n_Providers+1)..n_Nodes;
finalize(E_ProviderNode);

! If Symmetric is set to true in provided dataset
! - duplicate all arcs in dataset in its opposite direction if opposite not already specified
if(Symmetric) then
    forall(nn in Nodes, mm in Nodes) do

        if(exists(K_CapPrice(nn,mm)) and not exists(K_CapPrice(mm,nn))) then
            create(K_CapPrice(mm,nn));
            K_CapPrice(mm,nn):=K_CapPrice(nn,mm);
```

```
        end-if

        if(exists(T_LinkLatency(nn,mm)) and not exists(T_LinkLatency(mm,nn))) then
            create(T_LinkLatency(mm,nn));
            T_LinkLatency(mm,nn):=T_LinkLatency(nn,mm);
        end-if

        if(exists(F_BandwidthCap(nn,mm)) and not exists(F_BandwidthCap(mm,nn))) then
            create(F_BandwidthCap(mm,nn));
            F_BandwidthCap(mm,nn):=F_BandwidthCap(nn,mm);
        end-if

        if(exists(D_AvailabilityExp(nn,mm)) and not exists(D_AvailabilityExp(mm,nn))) then
            create(D_AvailabilityExp(mm,nn));
            D_AvailabilityExp(mm,nn) :=D_AvailabilityExp(nn,mm);
        end-if
    end-do
end-if

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! VARIABLES
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declarations
    !Variables
    ! - x: binary, placement of service at provider
    x_Placement:        dynamic array(Services, Providers)          of mpvar;
    ! - u: binary, use of arc for service for uplink (and oppsite arc for downlink)
    u_UsePrimary:       dynamic array(Nodes,Nodes,Services)         of mpvar;
    ! - y: binary, serving of a customer
    y_Serve:            dynamic array(Customers)                    of mpvar;
    ! - b : binary, use of arc for service for backup uplink (and opposite arc for downlink)
    b_UseBackup:        dynamic array(Nodes,Nodes,Services)         of mpvar;
    ! - r : binary, is service s needs backup on its path to provider p
    r_RequireBackup:    dynamic array(Services,Providers)           of mpvar;
    ! - l (lambda): continuous, bandwidth reserved for backup on a (owned) link
    l_BackupRes:        dynamic array(Nodes,Nodes)                  of mpvar;
    ! - l: binary, indicates if two services have overlapping primary paths
    l_Overlap:          dynamic array(Services,Services)            of mpvar;
end-declarations

! - for all valid combinations of service and provider
forall (ss in Services, pp in Providers | exists(H_PlacePrice(ss,pp))) do
    create (x_Placement(ss,pp));
    x_Placement(ss,pp) is_binary;
    create(r_RequireBackup(ss,pp));
    r_RequireBackup(ss,pp) is_binary;
end-do

! - for evary arc in network
forall(ii in Nodes, jj in Nodes) do
    create(l_BackupRes(ii,jj));
end-do

! - for every service
forall(cc in Customers, ss in S_ServicesForCustomer(cc)) do
    ! - for every arc in network
    ! -- EXCEPT: arcs in to customer node of service, as paths from customer to provider will
    !            never traverse these links
    forall(ii in Nodes, jj in Nodes | jj<>cc and exists(F_BandwidthCap(ii,jj))) do
        create(u_UsePrimary(ii,jj,ss));
        u_UsePrimary(ii,jj,ss) is_binary;
        create(b_UseBackup(ii,jj,ss));
        b_UseBackup(ii,jj,ss) is_binary;
    end-do
end-do

! - for all customers
forall(cc in Customers) do
    create(y_Serve(cc));
    y_Serve(cc) is_binary;
end-do

! - for every distinct pair of two services
forall (ss in Services, tt in Services | ss < tt) do
    create(l_Overlap(ss,tt));
end-do

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! CONSTRAINTS
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
declarations
! Objective function
    Total_Profits:                                                          linctr;

! Constraints
    ServeCustomer:              dynamic array(Services)                      of linctr;
    ArcCapacity:                dynamic array(Nodes,Nodes)                   of linctr;
    PrimaryStartRequirement:    dynamic array(Services)                      of linctr;
    BackupStartRequirement:     dynamic array(Services)                      of linctr;
    BandwidthFlowPrimary:       dynamic array(Services,I_Nodes)              of linctr;
    BandwidthFlowBackup:        dynamic array(Services,I_Nodes)              of linctr;
    PrimaryEndRequirement:      dynamic array(Services,Providers)            of linctr;
    BackupEndRequirement:       dynamic array(Services,Providers)            of linctr;
    PrimaryLatencyRequirement:  dynamic array(Services,Providers)            of linctr;
    BackupLatencyRequirement:   dynamic array(Services,Providers)            of linctr;
    AllocateBackupPath:         dynamic array(Services,Providers)            of linctr;
    AvailabilityRequirement:    dynamic array(Services)                      of linctr;
    SumBackupLimit:             dynamic array(Nodes,Nodes)                   of linctr;
    SingleBackupLimit:          dynamic array(Nodes,Nodes,Services)          of linctr;
    LinkDisjoint:               dynamic array(Nodes,Nodes,Services)          of linctr;
    PrimaryOverlap:             dynamic array(Nodes,Nodes,Services,Services) of linctr;
    BackupOverlap:              dynamic array(Nodes,Nodes,Services,Services) of linctr;
end-declarations

! OBJECTIVE FUNCTION
! - total profits from serving customers
Total_Profits := (
        sum (cc in Customers) (
            ! R_Revenue from serving customer (if served)
            R_Revenue(cc)*y_Serve(cc)
            -
            ! costs associated with customer's required services
            sum (ss in S_ServicesForCustomer(cc)) (
                ! placement cost
                sum (pp in Providers) (
                    H_PlacePrice(ss,pp)*x_Placement(ss,pp)
                )
                +
                ! network usage cost
                sum (nn in Nodes, mm in Nodes | exists(K_CapPrice(nn,mm))) (
                    K_CapPrice(nn,mm)
                    *
                    (
                        B_BandwidthReqUp(ss)*u_UsePrimary(nn,mm,ss)
                        +
                        B_BandwidthReqDown(ss)*u_UsePrimary(mm,nn,ss)
                    )
                )
            )
        )
        -
        !Backup use cost
        sum (nn in Nodes, mm in Nodes | exists(K_CapPrice(nn,mm))) (
            K_CapPrice(nn,mm)*l_BackupRes(nn,mm)
        )
    );

! SERVE CUSTOMER CONSTRAINT
! Customers can only be served if all services for customer is provided
forall(cc in Customers) do
    forall(ss in S_ServicesForCustomer(cc)) do
        ServeCustomer(ss) := sum (pp in Providers) x_Placement(ss,pp) - y_Serve(cc) = 0;
    end-do
end-do

! ARC TOTAL CAPACITY CONSTRAINT
! Use of an arc must not exceed its capacity (primary + backup cap)
forall (nn in Nodes, mm in Nodes| exists(F_BandwidthCap(nn,mm)))do
    ArcCapacity(nn,mm) := (
        sum (ss in Services) (
            B_BandwidthReqUp(ss)*u_UsePrimary(nn,mm,ss)
            +
            B_BandwidthReqDown(ss)*u_UsePrimary(mm,nn,ss)
        )
        +
        l_BackupRes(nn,mm)
        <=
        F_BandwidthCap(nn,mm)
    );
end-do
```

```
! CUSTOMER NODE ROUTING START CONSTRAINTS:
! primary / backup must select arc from customer node if chosen
forall (cc in Customers, ss in S_ServicesForCustomer(cc)) do
    PrimaryStartRequirement(ss) := (
            sum(mm in Nodes | mm<>cc) (u_UsePrimary(cc,mm,ss) )
             - y_Serve(cc)
        )= 0;
    BackupStartRequirement(ss) := (
            sum(mm in Nodes | mm<>cc) b_UseBackup(cc,mm,ss)
            -
            sum(pp in Providers) r_RequireBackup(ss,pp)
        ) =0;
end-do

! ROUTING FLOW CONSTRAINTS
! - routing in to a node for a service must be equal to the routing out
!   (unless a it is a provider node or the service's customer node)
forall (cc in Customers, ss in S_ServicesForCustomer(cc), nn in I_Nodes | nn<>cc ) do
    BandwidthFlowPrimary(ss,nn) := (
            sum (mm in Nodes | exists(F_BandwidthCap(nn,mm))) u_UsePrimary(nn,mm,ss)
            -
            sum(mm in Nodes | exists(F_BandwidthCap(mm,nn))) u_UsePrimary(mm,nn,ss)
        ) = 0;
    BandwidthFlowBackup(ss,nn) := (
            sum (mm in Nodes | exists(F_BandwidthCap(nn,mm))) b_UseBackup(nn,mm,ss)
            -
            sum(mm in Nodes | exists(F_BandwidthCap(mm,nn))) b_UseBackup(mm,nn,ss)
        ) = 0;
end-do

! PLACEMENT SIDE ROUTING END CONSTRAINTS
! primary / backup must select arc in to placement node if chosen, or act as transit node
! if not selected / not able to be selected
! and primary and backup routing must end at same provider
forall (ss in Services, pp in Providers) do
    PrimaryEndRequirement(ss,pp) := (
            sum (nn in Nodes | nn<>E_ProviderNode(pp)) (
                u_UsePrimary(nn, E_ProviderNode(pp),ss)
            )
            -
            sum(mm in Nodes | mm<>E_ProviderNode(pp)) (
                u_UsePrimary(E_ProviderNode(pp),mm,ss)
            )
            -
            x_Placement(ss,pp)
        ) = 0;
    BackupEndRequirement(ss,pp) := (
            sum (nn in Nodes | nn<>E_ProviderNode(pp)) (
                b_UseBackup(nn, E_ProviderNode(pp),ss)
            )
            -
            sum(mm in Nodes | mm<>E_ProviderNode(pp)) (
                b_UseBackup(E_ProviderNode(pp),mm,ss)
            )
            -
            r_RequireBackup(ss,pp)
    ) = 0;

    if(exists(H_PlacePrice(ss,pp))) then
        ! can only have backup paths to same provider as primary
        AllocateBackupPath(ss,pp):= r_RequireBackup(ss,pp) - x_Placement(ss,pp) <= 0;
    end-if
end-do

! LATENCY REQUIREMENT CONSTRAINTS
! - user -> placement: for each service, latency for any used path must meet latency requirements
forall (ss in Services, pp in Providers) do
    PrimaryLatencyRequirement(ss,pp) :=
            sum(nn in Nodes, mm in Nodes) (
                T_LinkLatency(nn,mm)*(u_UsePrimary(nn,mm,ss) + u_UsePrimary(mm,nn,ss))
            )
            <= G_LatencyReq(ss);

    BackupLatencyRequirement(ss,pp) :=
            sum(nn in Nodes, mm in Nodes) (
                T_LinkLatency(nn,mm)*(b_UseBackup(nn,mm,ss) + b_UseBackup(mm,nn,ss))
            )
            <= G_LatencyReq(ss);
end-do
```

```
! AVAILABILITY CONSTRAINTS
! Primary path must have sufficient availability or a link disjoint backup path must be provided
! - linearised by using logarithms
forall (ss in Services) do
    AvailabilityRequirement(ss) := (
        sum( nn in Nodes, mm in Nodes | exists(D_AvailabilityExp(nn,mm))) (
            ln(D_AvailabilityExp(nn,mm)) * u_UsePrimary(nn,mm,ss)
        )
        +
        sum(pp in Providers) r_RequireBackup(ss,pp)
        >=
        ln(Y_AvailabilityReq(ss)));
end-do

! SUM BACKUP REQUIREMENT
! must reserve a certain proportion of the sum of backup requirements on an arc
forall (nn in Nodes, mm in Nodes) do
    SumBackupLimit(nn,mm) := (
        MinBackupProportion*
        sum(ss in Services) (
            B_BandwidthReqUp(ss)*b_UseBackup(nn,mm,ss)
            +
            B_BandwidthReqDown(ss)*b_UseBackup(mm,nn,ss)
        )
        -
        l_BackupRes(nn,mm)
        <= 0);
end-do

forall (ii in Nodes, jj in Nodes, ss in Services ) do
    ! MAXIMUM BACKUP CONTRAINT
    ! Must reserve backup capacity at least as high as the maximal single backup requirement
    SingleBackupLimit(ii,jj,ss) := (
        B_BandwidthReqUp(ss)
        *b_UseBackup(ii,jj,ss)  ! 1 if ii,jj is used in way UP
        +
        B_BandwidthReqDown(ss)
        *b_UseBackup(jj,ii,ss)  ! 1 if ii,jj is used in way DOWN
        -
        l_BackupRes(ii,jj)
        <= 0
    );

    ! LINK DISJOINT CONSTRAINTS
    ! The primary and backup path (if given) for a service must be link disjoint
    LinkDisjoint(ii,jj,ss) := b_UseBackup(ii,jj,ss) + u_UsePrimary(ii,jj,ss) <=1;
end-do

! SERVICE PATH OVERLAP CONSTRAINTS
! To services have overlapping main paths if for any arc both paths are represented
! for every service combination
forall(ss in Services, tt in Services | ss < tt) do
    ! for every LINK ( (i,j) in A | i < j)
    forall(ii in Nodes, jj in Nodes | ii < jj and exists(F_BandwidthCap(ii,jj))) do

        ! PRIMARY PATH OVERLAP CONSTRAINTS
        ! Two services have overlapping primary paths if for any LINK both services
        ! has selected one of the link's two arcs
        PrimaryOverlap(ii, jj, ss, tt):=
            u_UsePrimary(ii,jj,ss)+u_UsePrimary(jj,ii,ss) ! 1 if ss uses link
            +
            u_UsePrimary(ii,jj,tt)+u_UsePrimary(jj,ii,tt) ! 1 if tt uses link
            -
            l_Overlap(ss, tt)
            <= 1;

        ! BACKUP PATH OVERLAP CONSTRAINT
        ! backup paths may not overlap at any LINK if their primary paths overlap anywhere
        BackupOverlap(ii, jj, ss,tt):=
            b_UseBackup(ii,jj,ss)+b_UseBackup(ii,jj,ss) ! 1 if ss uses link
            +
            b_UseBackup(ii,jj,tt)+b_UseBackup(ii,jj,tt) ! 1 if tt uses link
            +
            l_Overlap(ss, tt)
            <= 2;
    end-do
end-do

writeln("Model building completed in ", timestamp - timetracker, " seconds");

writeln("Solving model...");
```

```
timetracker := timestamp;
maximize(XPRS_PRI, Total_Profits);

if (getprobstat=XPRS_OPT) then
    writeln("Model solved in ", timestamp - timetracker," seconds");
else
    writeln("Model was not solved after ", timestamp - timetracker," seconds");
end-if

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!   Solution output:
! - this following part contains logic for outputting the solution as human
!   readable text and is not part of the model itself.
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

writeln("\nTotal Profits: ", getobjval);
writeln("\nBackup Costs: ",
    sum(nn in Nodes, mm in Nodes) getsol(l_BackupRes(nn,mm))*K_CapPrice(nn,mm)
);

! for all customers being served
forall(cc in Customers | getsol(y_Serve(cc)) > 0.1) do
    ! output customer information and generated profits (excluding backup costs)
    writeln("\n\nCustomer ", cc, " (node ",cc,") is being served\n - R_Revenue: ",
        R_Revenue(cc)*getsol(y_Serve(cc)),"\n - profits: ",(
            R_Revenue(cc)*getsol(y_Serve(cc))
            -
            ! costs associated with customer's required services
            sum (ss in S_ServicesForCustomer(cc))
                (
                    ! placement cost
                    sum (pp in Providers)
                        H_PlacePrice(ss,pp)*getsol(x_Placement(ss,pp))
                    +
                    ! network usage cost
                    sum (nn in Nodes, mm in Nodes) (
                            K_CapPrice(nn,mm)
                            *(
                                getsol(u_UsePrimary(nn,mm,ss)*B_BandwidthReqUp(ss))
                                +getsol(u_UsePrimary(mm,nn,ss)*B_BandwidthReqDown(ss)))
                            )
                )
            )
        );

    ! for all services of the served customer
    forall(ss in S_ServicesForCustomer(cc)) do
        ! for the provider selected for the service (x only > 0.1 for one)
        forall(pp in Providers | getsol(x_Placement(ss,pp)) > 0.1) do
            ! output information about service and placement
            writeln(
                "\n - Service ",ss,":\n  - Costs: ",
                (   ! Calculate costs for this specific service
                    H_PlacePrice(ss,pp)*getsol(x_Placement(ss,pp))
                    +
                    sum (nn in Nodes, mm in Nodes) (
                        K_CapPrice(nn,mm)
                        *(
                            getsol(u_UsePrimary(nn,mm,ss))*B_BandwidthReqUp(ss)
                            +getsol(u_UsePrimary(mm,nn,ss))*B_BandwidthReqDown(ss)
                        )
                    )
                ),
                "\n  - placement: provider #", pp, " (node ",(n_Nodes-n_Providers+pp),
                ") - Cost: ",H_PlacePrice(ss,pp),
                "    -Availability without backup: ",
                exp(
                    sum( nn in Nodes, mm in Nodes | (exists(D_AvailabilityExp(nn,mm))) ) (
                        getsol(u_UsePrimary(nn,mm,ss))*ln(D_AvailabilityExp(nn,mm))
                    )
                ),
                "    -Availability requirement: ", Y_AvailabilityReq(ss)
            );
        end-do

        ! output primary path network routing information for service
        ! - up-link
        writeln("   - ARCS:\n      - primary usage up:");
        forall(nn in Nodes, mm in Nodes) do
```

```
            if (getsol(u_UsePrimary(nn,mm,ss)) > 0.1) then
                writeln("      - (", nn, ",", mm, ") : ",
                        B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss)),
                        "    (",
                        K_CapPrice(nn,mm)*B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss)),")" );
            end-if
        end-do
        ! - down-link
        writeln("      - primary usage down:");
        forall(nn in Nodes, mm in Nodes) do
            if (getsol(u_UsePrimary(nn,mm,ss)) > 0.1) then
                writeln("        - (", mm, ",", nn, ") : ",
                        B_BandwidthReqDown(ss)*getsol(u_UsePrimary(nn,mm,ss)),
                        "    (",
                        K_CapPrice(nn,mm)*B_BandwidthReqDown(ss)*getsol(u_UsePrimary(nn,mm,ss)),")" );
            end-if
        end-do

        ! if this service requires a backup path (given its primary path routing)
        if (
            sum( nn in Nodes, mm in Nodes | (exists(D_AvailabilityExp(nn,mm))) ) (
                getsol(u_UsePrimary(nn,mm,ss))*ln(D_AvailabilityExp(nn,mm))
            )
            < ln(getsol(Y_AvailabilityReq(ss)))
        ) then
            ! output backup path network routing information
            ! - up-link
            writeln("      - backup usage up:");
            forall (nn in Nodes, mm in Nodes) do
                if (getsol(b_UseBackup(nn,mm,ss))=1) then
                    writeln("        - (",nn,",",mm,"): ", getsol(l_BackupRes(nn,mm)));
                end-if
            end-do
            ! - down-link
            writeln("      - backup usage down:");
            forall (nn in Nodes, mm in Nodes) do
                if (getsol(b_UseBackup(nn,mm,ss))=1) then
                    writeln("        - (",mm,",",nn,"): ", getsol(l_BackupRes(mm,nn)));
                end-if
            end-do
        end-if
    end-do
end-do

!! Output information about total bandwidth usage on arcs
! arcs with high bandwidth usage
writeln("\n\nArcs with high utilisation of capacity (>=90%):");
forall(nn in Nodes, mm in Nodes | exists(F_BandwidthCap(nn,mm))) do
    if (sum(ss in Services) B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss))) >=
F_BandwidthCap(nn,mm)*0.9 then
        writeln(
            " - (",nn,",",mm,") ",
            (
                100*sum(ss in Services) (
                    B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss))
                ) / F_BandwidthCap(nn,mm)
            ),
            " %"
        );
    end-if
end-do
! arcs with medium bandwidth usage
writeln("\n\nArcs with medium utilisation of capacity (< 10%, < 90%):");
forall(nn in Nodes, mm in Nodes | exists(F_BandwidthCap(nn,mm))) do
    if ((sum(ss in Services)B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss))) >
F_BandwidthCap(nn,mm)*0.1 and
        (sum(ss in Services)B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss))) <
F_BandwidthCap(nn,mm)*0.9) then
        writeln(
            " - (",nn,",",mm,") ",
            (
                100*sum(ss in Services)(
                    B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss))
                ) / F_BandwidthCap(nn,mm)
            ),
            " %"
        );
    end-if
end-do

end-model
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!
!! Carrier Broker Optimisation: Path Flow Model
!!
!!- The following script implements the PFM defined in Section 6.2
!!  of Mari Holmen's and Sindre Møgster Braaten's masters thesis
!!
!!- Authors: Mari Holmen and Sindre Møgster Braaten
!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

model CarrierBrokerPFM

uses "mmxprs"; !gain access to the Xpress-Optimizer solver
options explterm
options noimplicit

uses "mmxprs", "mmsystem";

parameters
    ! Data file to read from
    Data = 'data/ml_multi.txt';
    ! Minimum proportion of total backup requirement reserved on an arc
    MinBackupProportion = 0.25;
    ! Time limit for runtime,  maximum number of seconds for optimisation
    TimeLimit = -1;
end-parameters

writeln("Model Parameters:");
writeln("Data:", Data);
writeln("MinBackupProportion(beta):", MinBackupProportion);
writeln("TimeLimit:", TimeLimit);

declarations
    timetracker:    real; ! used to log timestamps for time consumption output
end-declarations

writeln("Building model...");
timetracker := timestamp; ! assigns current "timestamp" to timetracker

!setparam("XPRS_presolve", 0);  ! uncomment to turn of presolve
if(TimeLimit>0.0) then
    setparam("XPRS_maxtime", TimeLimit);
end-if

setparam("XPRS_verbose", true); ! Turn on message printing
setparam("XPRS_MIPLOG", 2); ! 2: print information for each solution found
                            !(ALT: 0: no log, 1: summary in end, 3: log each node, -N: log every Nth node)

!!!!!!!!!!!!!!!!!!!!!!!!
! SETS
!!!!!!!!!!!!!!!!!!!!!!!!

declarations
! Set sizes
    n_Customers:            integer; ! number of customers
    n_Services:             integer; ! number of services
    n_Providers:            integer; ! number of providers
    n_Nodes:                integer; ! number of nodes in total
    n_Paths:                integer; ! number of paths

! Sets
    Customers:              set of integer;
    Providers:              set of integer;
    ! Used for shorthand for 'cc in Customers, ss in S_ServiceForCustomer(cc)' when cc is not needed
    Services:               set of integer;
    ! Set of nodes in the network.
    ! - First we have the customer nodes, then the internal nodes, the the provider nodes.
    Nodes:                  set of integer;
    Paths:                  set of integer;
end-declarations

initializations from Data
    n_Customers;
    n_Services;
    n_Providers;
    n_Nodes;
    n_Paths;

end-initializations
```

```
Customers:= 1..n_Customers;
Services:= 1..n_Services;
Providers:= 1..n_Providers;
Nodes:= 1..n_Nodes;
Paths := 1..n_Paths;

finalize(Customers);
finalize(Services);
finalize(Providers);
finalize(Nodes);
finalize(Paths);

! INDEXED SETS

declarations
    ! set of services of for each customer
    S_ServicesForCustomer:      set of set of integer;
    ! paths for each pair of service and provider
    K_PathsServiceProvider:     dynamic array(Services,Providers)   of set of integer;
    ! set of paths using each link
    L_PathsUsingArc:            dynamic array(Nodes,Nodes)          of set of integer;
end-declarations

initialisations from Data
    S_ServicesForCustomer;
    K_PathsServiceProvider;
    L_PathsUsingArc;
end-initialisations

!!!!!!!!!!!!!!!!!!!!!!!
! PARAMETERS
!!!!!!!!!!!!!!!!!!!!!!!

declarations
    ! R_Revenue from serving each customer cc
    R_Revenue:                  dynamic array(Customers)            of real;
    ! Bandwidth capacity between each pair of nodes ii,jj
    F_BandwidthCap:             dynamic array(Nodes,Nodes)          of real;
    ! bandwidth usage on arc ii,jj for path kk
    U_PathBandwidthUsage:       dynamic array(Nodes,Nodes,Paths)    of real;
    ! cost of using path kk
    C_PathCost:                 dynamic array(Paths)                of real;
    ! cost per bandwidth used for backup paths on arc ii,jj
    C_BackupCost:               dynamic array(Nodes,Nodes)          of real;
    ! availability of path kk alone
    D_PathAvailability:         dynamic array(Paths)                of real;
    ! 2d array of availability for paths kk and bb ( P(A)P(B|A) )
    D_CombinationAvailability:  dynamic array(Paths,Paths)          of real;
    ! array of availability req for services
    Y_AvailabilityReq:          dynamic array(Services)             of real;

    BigMBackup:                 dynamic array(Nodes,Nodes,Services) of real;

    Symmetric:                                                      boolean;
end-declarations


initialisations from Data
    R_Revenue;
    F_BandwidthCap;
    U_PathBandwidthUsage;
    C_PathCost;
    C_BackupCost;
    D_PathAvailability;
    D_CombinationAvailability;
    Y_AvailabilityReq;
    Symmetric;
end-initialisations

if(Symmetric) then
    forall(ii in Nodes, jj in Nodes) do
        if(exists(F_BandwidthCap(ii,jj)) and not exists(F_BandwidthCap(jj,ii))) then
            create(F_BandwidthCap(jj,ii));
            F_BandwidthCap(jj,ii):=F_BandwidthCap(ii,jj);
        end-if
    end-do
end-if

! for every arc in network
forall(ii in Nodes, jj in Nodes | exists(F_BandwidthCap(ii,jj))) do
```

```
    ! for every service
    forall(ss in Services) do
        ! set BigMBackup(ii,jj,ss) to highest bandwidth usage of any k for s in i,j
        create(BigMBackup(ii,jj,ss));
        BigMBackup(ii,jj,ss) := 0.0;
        forall(pp in Providers | exists(K_PathsServiceProvider(ss,pp))) do
            forall(kk in (K_PathsServiceProvider(ss,pp)*L_PathsUsingArc(ii,jj))) do
                if(BigMBackup(ii,jj,ss) < U_PathBandwidthUsage(ii,jj,kk)) then
                    BigMBackup(ii,jj,ss):= U_PathBandwidthUsage(ii,jj,kk);
                end-if
            end-do
        end-do
    end-do
end-do

!!!!!!!!!!!!!!!!!!!!!!!!!!!
! VARIABLES
!!!!!!!!!!!!!!!!!!!!!!!!!!!

declarations
    ! - x: binary, placement of service at provider
    x_Placement:        dynamic array(Services, Providers)      of mpvar;
    ! - y: binary, serving of a customer
    y_Serve:            dynamic array(Customers)                of mpvar;
    ! - u: binary, indicates which paths are used
    u_UsePrimaryPath:   dynamic array(Paths)                    of mpvar;
    ! - v: binary, indicates which backup paths are used
    v_UseBackupPath:    dynamic array(Paths)                    of mpvar;
    ! - o: binary, indicates if a combination of main and backup path is chosen
    o_UseCombination:   dynamic array(Paths,Paths)              of mpvar;
    ! - lambda: continous, amount of capacity reserved on a link for backup
    l_Lambda:           dynamic array(Nodes,Nodes)              of mpvar;
    ! f: binary, indicates if a service has a need to reserve backup capacity on a arc
    f_needsBackupOnArc: dynamic array(Nodes,Nodes,Services)     of mpvar;
    ! - q: continous, amount of backup capacity needed on an arc for a service
    q_backupPerService: dynamic array(Nodes, Nodes, Services)   of mpvar;
    ! - l: binary, indicates if two services have overlapping primary paths
    l_Overlap:          dynamic array(Services,Services)        of mpvar;
end-declarations

! - for all combinations of service and provider
forall (ss in Services, pp in Providers) do
    create (x_Placement(ss,pp));
    x_Placement(ss,pp) is_binary;
end-do

! - for all customers
forall(cc in Customers) do
    create(y_Serve(cc));
    y_Serve(cc) is_binary;
end-do

! - for all paths
forall(pp in Paths) do
    create(u_UsePrimaryPath(pp));
    u_UsePrimaryPath(pp) is_binary;
    create(v_UseBackupPath(pp));
    v_UseBackupPath(pp) is_binary;
end-do

! - for every possible combination of two paths as primary and backup
forall (pp in Paths, bb in Paths | exists(D_CombinationAvailability(pp,bb))) do
    create(o_UseCombination(pp,bb));
    o_UseCombination(pp,bb) is_binary;
end-do

! - for every arc
forall (ii in Nodes, jj in Nodes, ss in Services | exists(F_BandwidthCap(ii,jj))) do
    create(l_Lambda(ii,jj));
    create(q_backupPerService(ii,jj,ss));
    create(f_needsBackupOnArc(ii,jj,ss));
    f_needsBackupOnArc(ii,jj,ss) is_binary;
end-do

! - for every distinct pair of two services
forall (ss in Services, tt in Services | ss < tt) do
    create(l_Overlap(ss,tt));
    l_Overlap(ss,tt) is_binary;
end-do
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! CONSTRAINTS
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declarations
! Objective function
    Total_Profits:                                                      linctr;

! Constraints
    ServeCustomer:              dynamic array(Services)                  of linctr;
    AllocatePrimaryPath:        dynamic array(Services,Providers)         of linctr;
    AllocateBackupPath:         dynamic array(Services,Providers)         of linctr;
    ArcCapacity:                dynamic array(Nodes,Nodes)                of linctr;
    AvailabilityRequirement:    dynamic array(Services)                   of linctr;
    SumBackupLimit:             dynamic array(Nodes,Nodes)                of linctr;
    SingleBackupLimit:          dynamic array(Nodes,Nodes,Services)        of linctr;
    NeedsBackupCapacity:        dynamic array(Nodes,Nodes,Services)        of linctr;
    NeededBackupCapacity:       dynamic array(Nodes,Nodes,Services)        of linctr;
    PathComboRequirement:       dynamic array(Paths,Paths)                of linctr;
    PrimaryOverlap:             dynamic array(Nodes,Nodes,Services,Services)   of linctr;
    BackupOverlap:              dynamic array(Nodes,Nodes,Services,Services)   of linctr;
end-declarations

! OBJECTIVE FUNCTION
! - total profits from serving customers
Total_Profits := sum (cc in Customers) (
            ! R_Revenue from serving customer (if served)
            R_Revenue(cc)*y_Serve(cc)
        )
        -
        ! for all paths
        sum(kk in Paths) (
            ! add path cost if used as primary
            C_PathCost(kk)*u_UsePrimaryPath(kk)
        )
        -
        ! for eac arc
        sum(ii in Nodes, jj in Nodes | exists(F_BandwidthCap(ii,jj))) (
            ! add cost of bandwidth reserved for backup paths
            C_BackupCost(ii, jj)*l_Lambda(ii, jj)
    );


! SERVE CUSTOMER CONSTRAINT
! Customers can only be served if all services for customer is provided
forall(cc in Customers) do
    forall(ss in S_ServicesForCustomer(cc)) do
        ServeCustomer(ss) :=
            sum (pp in Providers | exists(K_PathsServiceProvider(ss,pp))) (
                x_Placement(ss,pp)
            ) = y_Serve(cc);
    end-do
end-do

! for every service-provider pair
forall(ss in Services, pp in Providers | exists(K_PathsServiceProvider(ss,pp))) do
    ! ALLOCATE PRIMARY PATH CONSTRAINT
    ! If a service is placed at a provider, a primary path connecting to that provider location
    ! must be chosen
    AllocatePrimaryPath(ss,pp) :=
        sum(kk in K_PathsServiceProvider(ss,pp)) (
            u_UsePrimaryPath(kk)
        ) = x_Placement(ss,pp);

    ! ALLOCATE BACKUP PATH CONSTRAINT
    ! can only select a backup path to a provider if also selected primary path to the same provider
    AllocateBackupPath(ss,pp) :=
        sum(kk in K_PathsServiceProvider(ss,pp)) (
            v_UseBackupPath(kk)
        )
        <=
        sum(kk in K_PathsServiceProvider(ss,pp)) (
            u_UsePrimaryPath(kk)
        );
end-do

! ARC CAPACITY CONSTRAINT
! The total used bandwidth for main paths and reserved for backup paths must not exceed the
! arcs capacity
forall(ii in Nodes, jj in Nodes | exists(L_PathsUsingArc(ii,jj))) do
    ArcCapacity(ii,jj) :=
```

```
        ! for each path
        sum(kk in L_PathsUsingArc(ii,jj)) (
            ! add bw req for path if used as primary
            U_PathBandwidthUsage(ii,jj,kk)*u_UsePrimaryPath(kk)
        )
        ! add bandwidth reserved for backup paths on arc
        +
        l_Lambda(ii,jj)
        <= F_BandwidthCap(ii,jj);
end-do

! AVAILABILITY REQUIREMENT CONSTRAINT
! The selected main path, with possible backup path, must provide an availability equal to
! or higher than requirement
! - single main path: P(A)
! - with backup path: P(A) + P(B) - P(A)P(B|A)
forall(cc in Customers) do
    forall(ss in S_ServicesForCustomer(cc)) do
        AvailabilityRequirement(ss) :=
            ! Availability from main (and backup path): P(A) + P(B)
            sum(pp in Providers | exists(K_PathsServiceProvider(ss,pp))) (
                sum(kk in K_PathsServiceProvider(ss,pp)) (
                    D_PathAvailability(kk)
                    *
                    (
                        u_UsePrimaryPath(kk)
                        +
                        v_UseBackupPath(kk)
                    )
                )
            )
            -
            ! subtract P(A)P(B|A) if backup path is chosen
            sum(pp in Providers | exists(K_PathsServiceProvider(ss,pp))) (
                sum(kk in K_PathsServiceProvider(ss,pp), bb in K_PathsServiceProvider(ss,pp))
                    D_CombinationAvailability(kk,bb)*o_UseCombination(kk,bb)
            )
            >= Y_AvailabilityReq(ss)* y_Serve(cc);
    end-do
end-do

! BACKUP BANDWIDTH RESERVATION CONSTRAINTS
! - for every arc and service
forall (ii in Nodes, jj in Nodes, ss in Services | exists(F_BandwidthCap(ii,jj))) do

    ! NEEDS BACKUP CAPACITY ON ARC CONSTRAINT
    ! Sets the f variable to 1 if there is a need for backup capacity reservation
    !for the service on the arc
    NeedsBackupCapacity(ii,jj,ss) :=
        q_backupPerService(ii,jj,ss) - BigMBackup(ii,jj,ss)*f_needsBackupOnArc(ii,jj,ss) <= 0;

    ! AMOUNT BACKUP CAPACITY NEEDED ON ARC CONSTRAINT
    ! a service will require a backup reservation at an arc equal to its backup arc requirement
    ! minus the capacity used by the primary path at the same arc (as this capacity will be released
    ! if the primary path goes down and the backup is needed)
    NeededBackupCapacity(ii,jj,ss) :=
        ! bandwidth for backup minus bandwidth for primary on arc
        sum(pp in Providers | exists(K_PathsServiceProvider(ss,pp)))(
            sum(kk in (K_PathsServiceProvider(ss,pp)*L_PathsUsingArc(ii,jj)))(
                U_PathBandwidthUsage(ii,jj,kk)
                *
                (
                    v_UseBackupPath(kk)
                    -
                    u_UsePrimaryPath(kk)
                )
            )
        )
        -
        q_backupPerService(ii,jj,ss)
        <= 0;
end-do

! SUM SERVICE BACKUP BANDWIDTH CONSTRAINTS
! bandwidth reserved for backup paths on an arc is at least a fraction of the total bandwidth of all
! backup paths using that arc
forall( ii in Nodes, jj in Nodes | exists(L_PathsUsingArc(ii,jj))) do
    SumBackupLimit(ii,jj) :=
        MinBackupProportion*
        sum(cc in Customers) (
            sum(ss in S_ServicesForCustomer(cc)) (
```

```
                q_backupPerService(ii,jj,ss)
            )
        )
        <= l_Lambda(ii,jj);
end-do

! SINGLE SERVICE BACKUP BANDWIDTH CONSTRAINT
! bandwidth reserved for backup paths must be at least as high as the bandwidth required by the
! the most demaning single service
forall(ii in Nodes, jj in Nodes | exists(L_PathsUsingArc(ii,jj))) do
    forall(ss in Services) do
        SingleBackupLimit(ii,jj,ss) :=
            q_backupPerService(ii,jj,ss) <= l_Lambda(ii,jj);
    end-do
end-do

! PATH COMBINATION CONSTRAINT
! if path kk is used as main path and path bb as backup path, the corresponding combo variable
! o_UseCombination(kk,bb) must also be 1
! for any valid combination of two paths; two paths from same service-provider pair
forall(cc in Customers) do
    forall(ss in S_ServicesForCustomer(cc), pp in Providers| exists(K_PathsServiceProvider(ss,pp))) do
        forall(kk in K_PathsServiceProvider(ss,pp), bb in K_PathsServiceProvider(ss,pp)) do
            PathComboRequirement(kk,bb) :=
                u_UsePrimaryPath(kk) + v_UseBackupPath(bb) - o_UseCombination(kk,bb) <= 1;
        end-do
    end-do
end-do

! SERVICE PATH OVERLAP CONSTRAINTS
! To services have overlapping main paths if for any arc both paths are represented
! for every service combination
forall(ss in Services, tt in Services | ss < tt) do
    ! for every LINK (every (ii,jj) where ii < jj)
    forall(ii in Nodes, jj in Nodes | exists(L_PathsUsingArc(ii,jj))) do

        ! only for evey link (ii < jj) (failured happen at link level -> failing both arcs)
        if(ii < jj) then
            ! SERVICE PATH OVERLAP CONSTRAINTS
            ! Two services have overlapping primary paths if for any LINK both paths are represented
            PrimaryOverlap(ii, jj, ss, tt):=
                sum(pp in Providers | exists(K_PathsServiceProvider(ss,pp)))(
                    ! L_PathsUsingArc(ii,jj)=L_PathsUsingArc(jj,ii) -> need only paths using LINK ii,jj
                    sum(kk in (K_PathsServiceProvider(ss,pp)*L_PathsUsingArc(ii,jj))) (
                        u_UsePrimaryPath(kk)
                    )
                )
                +
                sum(pp in Providers | exists(K_PathsServiceProvider(tt,pp)))(
                    sum(kk in (K_PathsServiceProvider(tt,pp)*L_PathsUsingArc(ii,jj))) (
                        u_UsePrimaryPath(kk)
                    )
                )
                -l_Overlap(ss, tt)
                <= 1;
        end-if

        ! BACKUP PATH OVERLAP CONSTRAINT
        ! backup paths may not overlap at any ARC if their primary paths overlap on any LINK
        BackupOverlap(ii,jj,ss,tt):=
            f_needsBackupOnArc(ii,jj,ss)
            +
            f_needsBackupOnArc(ii,jj,tt)
            + l_Overlap(ss, tt)
            <= 2;
    end-do
end-do

writeln("\nModel building completed in ", timestamp - timetracker, " seconds");

writeln("\nSolving model...");
timetracker := timestamp;
maximize(XPRS_PRI, Total_Profits);

if (getprobstat=XPRS_OPT) then
    writeln("\nModel solved in ", timestamp - timetracker," seconds");
else
    writeln("\nModel was not solved after ", timestamp - timetracker," seconds");
end-if

writeln("\nTotal Profits: ", getobjval);
```

```
writeln(
    "\nTotal Backup Costs: ",
    sum(ii in Nodes)(sum(jj in Nodes)(C_BackupCost(ii, jj)*getsol(l_Lambda(ii, jj))))
);

! for all served customers
forall(cc in Customers | getsol(y_Serve(cc)) > 0.001) do
    ! output served customer and generated profits for customer
    writeln(
        "\nCustomer ", cc, " (node ",cc,") is being served\n - R_Revenue: ",
        R_Revenue(cc)*getsol(y_Serve(cc))
    );
    ! for all services of customer
    forall(ss in S_ServicesForCustomer(cc)) do
        ! for the provider placement selected for service
        forall(pp in Providers | getsol(x_Placement(ss,pp)) > 0.001) do
            writeln(
                "  - Service ",ss," is placed at provider ",pp,
                "     - Availability req.: ", Y_AvailabilityReq(ss),
                "     - Exp. availability: ",
                (   ! calculate expected availability for mapping
                    sum(kk in K_PathsServiceProvider(ss,pp)) (
                        D_PathAvailability(kk)
                        *
                        (
                            getsol(u_UsePrimaryPath(kk))
                            +
                            getsol(v_UseBackupPath(kk))
                        )
                    )
                    -
                    ! subtract P(A)P(B|A) if backup path is chosen
                    sum(kk in K_PathsServiceProvider(ss,pp), bb in K_PathsServiceProvider(ss,pp))
                        D_CombinationAvailability(kk,bb)*getsol(o_UseCombination(kk,bb))
                )
            );
            forall(kk in K_PathsServiceProvider(ss,pp)) do
                if (getsol(u_UsePrimaryPath(kk)) > 0.001) then
                    writeln(
                        "     - Primary path: ", kk, " , cost: ",
                        getsol(u_UsePrimaryPath(kk))*C_PathCost(kk),
                        " (", getsol(u_UsePrimaryPath(kk))*100, " %)"
                    );
                end-if
            end-do
            forall(kk in K_PathsServiceProvider(ss,pp)) do
                if (getsol(v_UseBackupPath(kk)) > 0.001) then
                    writeln(
                        "     - Backup path: ", kk, " (",
                        getsol(v_UseBackupPath(kk))*100, " %)"
                    );
                end-if
            end-do
        end-do
    end-do
end-do

writeln("\nTotal backup usage");
writeln(
    strfmt("arc ",10),
    strfmt("reserved",10),
    strfmt("max req",10),
    strfmt("sum reqs*",10),
    strfmt("cost/bw",10),
    strfmt("paths",10)
);

declarations
    temp: real;
end-declarations

forall(ii in Nodes, jj in Nodes | exists(L_PathsUsingArc(ii,jj))) do
    ! Find the actual single maximal backup requirement on arc
    temp := 0.0;
    forall(ss in Services) do
        if(getsol(q_backupPerService(ii,jj,ss)) > temp) then
            temp := getsol(q_backupPerService(ii,jj,ss));
        end-if
    end-do
```

```
    ! Print information about the arc and backup reservations
    if(getsol(l_Lambda(ii,jj)) > 0.001) then
        write(
            strfmt("("+ ii+ ", "+ jj+ ")",10),
            strfmt(getsol(l_Lambda(ii,jj)), 10),
            strfmt(temp, 10),
            strfmt((sum(ss in Services)getsol(q_backupPerService(ii,jj,ss))), 10),
            strfmt(C_BackupCost(ii,jj), 10),
            "     "
        );
        forall(kk in L_PathsUsingArc(ii,jj) | getsol(v_UseBackupPath(kk)) > 0.001) do
            write(kk, ", ");
        end-do
        write("\n");
    end-if
end-do

end-model
```

Appendix F

C++ Code Samples

This appendix includes some samples of the C++ code implementing the Cloud Broker Application, showing some of the core functionality. The complete code is made available in the electronic delivery of this thesis.

F.1 Mapping Model with XpressBCL

The following code sample shows the implementation of the mapping model using the XpressBCL library.

```
/******  BuildModel *******
 * params:
 * - dataContent* data: pointer to data to build model from
 * - double beta_backupres: min fraction of total backup
 *                          requirement to reserve on arc
 *
 * Builds the MIP-model from the provided dataContent and
 * beta model parameter
 */
void CloudBrokerModel::BuildModel(dataContent * input_data,
    double beta_backupres, bool dedicated_only)
{
    /********* SETUP **********/
    this->data = input_data;
    this->dedicated = dedicated_only;
    if(this->dedicated) {
        this->beta = 1.0; // dedicated protection scheme -> beta = 1.0
    } else {
        this->beta = beta_backupres;
    }
    /* w variable iterator */
    list<XPRBvar>::iterator w_itr;
```

```
/********** CREATE VARIABLES **********/

cout << " - creating variables..\n";

/* Serve Customer variables
 * for: every customer
 */
y_serveCustomerVars.reserve(data->n_customers);
for(int cc = 0; cc < data->n_customers; cc++) {
    y_serveCustomerVars.push_back(
        master_problem.newVar(
                XPRBnewname(
                "y_serve_customer_%d", cc+1
            ),
            XPRB_BV, 0, 1
        )
    );
}
cout << "   - created " << y_serveCustomerVars.size()
     << " y-variables\n";

/* Use Mapping variables
 * for: every mapping
 * [shorthand for:
 *  for: every customer
 *   for: every service of customer
 *    for: every mapping of service
 * ] (global mapping list is ordered accordingly)
 */
for(int mm = 0; mm < data->n_mappings; mm++) {
    w_useMappingVars.push_back(
            master_problem.newVar(
                    XPRBnewname("m_use_mapping_%d", mm+1),
                    XPRB_BV, 0, 1
            )
    );
}
cout << "   - created " << w_useMappingVars.size()
     << " w-variables\n";

/* Services overlap variables
 * for: every pair of two services (s, t)
 *      - assume services are ordered,
 *        each combination of services where s < t
 */
int l_count = 0;
/* for every service s (except last service) */
if(!this->dedicated) {
    l_servicesOverlapVars.resize(data->n_services-1);
    for(int ss = 0; ss < data->n_services-1; ss++) {

        /* for every service t | t > s */
        l_servicesOverlapVars[ss].reserve(data->n_services-1-ss);
        for(int tt = ss+1; tt < data->n_services; tt++) {

            l_servicesOverlapVars[ss].push_back(
                    master_problem.newVar(
                            XPRBnewname(
                                "l_service_overlaps_%d_%d",
                                ss+1, tt+1
                            ),
```

```
                                XPRB_BV, 0, 1
                        )
                );
                ++l_count;
            }
        }
        cout << "   - created " << l_count << " l-variables\n";
    }

    /* Arc Backup Usage variable
     * for: every arc a
     */
    for(int aa = 0; aa < data->n_arcs; ++aa) {
        arc * a = &data->arcs[aa];
        lambda_arcBackupRes.push_back(
                master_problem.newVar(
                        XPRBnewname(
                            "lambda_backup_res_%d_%d",
                            a->startNode, a->endNode
                        ),
                        XPRB_PL, 0, a->bandwidth_cap
                )
        );
    }
    cout << "   - created " << lambda_arcBackupRes.size()
         << " lambda-variables\n";

    /******* CREATE OBJECTIVE FUNCTION ********/

    cout << " - creating objective function..";

    XPRBexpr z_obj_expression;

    /*  First term:
     *  - sum revenue from served customers
     */
    for(int cc = 0; cc < data->n_customers; cc++) {
        customer * c = &data->customers[cc];
        z_obj_expression += (
            Parameters::R_RevenueForCustomer(c)
            *y_serveCustomerVars[cc]
        );
    }

    /*  Second term:
     *  - sum cost from all used mappings primary paths
     */
    w_itr = w_useMappingVars.begin();
    for(int cc = 0; cc < data->n_customers; cc++) {
        customer * c = &data->customers[cc];
        for(unsigned int ss = 0; ss < c->services.size(); ss++)
        {
            service * s = c->services[ss];
            for (list<mapping*>::iterator m_itr = s->mappings.begin(),
                m_end = s->mappings.end(); m_itr != m_end; ++m_itr)
            {
                z_obj_expression -= (
                    Parameters::E_PrimaryPathCost((*m_itr)->primary)
                    *(*w_itr)
                );
                ++w_itr;
            }
```

```
    }
}

/* Third term:
 * - sum costs of total reserved capacity for backup paths on arcs
 */
for(int aa = 0; aa < data->n_arcs; ++aa) {
    arc * a = &data->arcs[aa];
    z_obj_expression -= (
        Parameters::E_PerBandwidthCostForArc(a)
        *lambda_arcBackupRes[aa]
    );
}

/* create final objective function */
z_objective = master_problem.newCtr("OBJ", z_obj_expression);
/* set objective function for problem */
master_problem.setObj(z_objective);

/******************** CREATE CONSTRAINTS ************************/

cout << "DONE!\n - creating constraints..\n";

/* SERVE CUSTOMER CONSTRAINTS
 * - If customer is to be served and generate revenue:
 *  - all services of that customer
 *    must be assigned a mapping
 * for: every customer
 *  for: every service of customer
 */
cout << "  - SERVE CUSTOMER CONSTRAINTS..";
w_itr = w_useMappingVars.begin();
serveCustomerCtr.reserve(data->n_services);
/* for every customer */
for(int cc = 0; cc < data->n_customers; ++cc) {
    customer * c = &data->customers[cc];

    /* for every service of customer */
    for(unsigned int ss = 0; ss < c->services.size(); ++ss) {
        service * s = c->services[ss];

        /* NEW CONSTRAINT: lhs expression */
        XPRBexpr map_service_expr;

        /* add serve customer var */
        map_service_expr += y_serveCustomerVars[cc];

        /* for every mapping of service*/
        for (list<mapping*>::iterator m_itr = s->mappings.begin(),
            m_end = s->mappings.end(); m_itr != m_end; ++m_itr)
        {

            /* subtract mapping selection var
            * - mapping selection var w is created in same order,
            *   can therefore loop over global w list
            */
            map_service_expr -= (*w_itr);
            ++w_itr;
        }

        /* create final constraint */
        serveCustomerCtr.push_back(
```

```
                master_problem.newCtr(
                        XPRBnewname(
                            "serve_customer_ctr_%d",
                            s->globalServiceIndex+1
                        ),
                        map_service_expr == 0.0
                )
            );
        }
    }
    cout << "DONE! (" << serveCustomerCtr.size() << " created) \n";

    /* ARC CAPACITY CONSTRAINT
     * - the sum of capacity used by chosen primary paths over link
     *   a and bandwidth reserved for backup must not exceed the
     *   capacity of the arc
     *
     * for: every arc a
     */
    cout <<     "  - ARC CAPACITY CONSTRAINTS..";

    /* for every arc */
    arcCapacityCtr.reserve(data->n_arcs);
    for(int aa = 0; aa < data->n_arcs; ++aa) {
        arc * a = &data->arcs[aa];

        /* NEW CONSTRAINT: lhs expression*/
        XPRBexpr arc_bw_usage;

        /* sum bandwidth use from all used mappings primary paths */
        w_itr = w_useMappingVars.begin();
        for(int cc = 0; cc < data->n_customers; ++cc)
        {
            customer * c = &data->customers[cc];
            for(unsigned int ss = 0; ss < c->services.size(); ++ss)
            {
                service * s = c->services[ss];
                for(list<mapping*>::iterator m_itr=s->mappings.begin(),
                    m_end = s->mappings.end(); m_itr != m_end; ++m_itr)
                {
                    mapping * m = *m_itr;

                    /* add bandwidth used by used mappings on arc */
                    arc_bw_usage += (
                Parameters::U_PrimaryBandwidthUsageOnArcForMapping(
                            a,m
                        )
                        *(*w_itr)
                    );

                    ++w_itr;
                }
            }
        }

        /* add bandwidth reserved for backup paths on arc */
        arc_bw_usage += lambda_arcBackupRes[aa];

        /* create final constraint: arc bw usage <= arc cap */
        arcCapacityCtr.push_back(
            master_problem.newCtr(
                XPRBnewname(
```

```
                    "arc_capacity_ctr_%d_%d",
                    a->startNode, a->endNode
                ),
                (arc_bw_usage
                <=
                Parameters::F_BandwidthCapacityForArc(a)
                )
            )
        );
    }
    cout << "DONE! (" << arcCapacityCtr.size() << " created) \n";

    /* SINGLE BACKUP RESERVATION CONSTRAINT
     * - backup capacity reserved on an arc must be large enough to
     *   support any single backup path using that arc
     *
     *   for: every arc
     *          for: every customer
     *                  for: every service of customer
     *
     *  When using dedicated protection scheme
     *  -> beta = 1.0 -> this constraint is made irrelevant by
     *     SUM BACKUP RESERVATION CONSTRAINT
     */
    if(!this->dedicated) {
        cout << "  - SINGLE BACKUP RESERVATION CONSTRAINTS..";

        /* for every arc                */
        backupSingleCtr.resize(data->n_arcs);
        int backupSingleCount = 0;
        for(int aa = 0; aa < data->n_arcs; ++aa) {
            arc * a = &data->arcs[aa];

            /* for every service */
            backupSingleCtr[aa].reserve(data->n_customers);
            w_itr = w_useMappingVars.begin();
            for(int cc = 0; cc < data->n_customers; ++cc) {
                customer * c = &data->customers[cc];
                for(unsigned int ss = 0; ss < c->services.size(); ++ss)
                {
                    service * s = c->services[ss];

                    /* NEW CONSTRAINT: lhs expression */
                    XPRBexpr service_backup_req_on_arc;

                    /* for every mapping of service */
                    for (list<mapping*>::iterator m_itr =
                        s->mappings.begin(), m_end = s->mappings.end();
                        m_itr != m_end; ++m_itr)
                    {
                        mapping * m = *m_itr;

                        /* add backup req on arc for used mapping */
                        service_backup_req_on_arc += (
            Parameters::Q_BackupBandwidthUsageOnArcForMapping(a, m)
                            *(*w_itr)
                        );

                        ++w_itr;
                    }

                    /* subtract arc backup usage variable */
```

```
                service_backup_req_on_arc -= lambda_arcBackupRes[aa];

                /* create final constraint */
                backupSingleCtr[aa].push_back(
                    master_problem.newCtr(
                        XPRBnewname(
                            "backup_single_ctr_%d_%d_%d",
                            a->startNode, a->endNode,
                            s->globalServiceIndex+1
                        ),
                        service_backup_req_on_arc <= 0.0
                    )
                );
                ++backupSingleCount;
            }
        }
    }
    cout << "DONE! (" << backupSingleCount << " created) \n";
}

/* SUM BACKUP RESERVATION CONSTRAINT
 * - a proportion of total backup requirement on an arc must
 *   be reserved
 * for: every arc
 */
cout << "  - SUM BACKUP RESERVATION CONSTRAINTS..";

/* for every arc */
backupSumCtr.reserve(data->n_arcs);
for(int aa = 0; aa < data->n_arcs; ++aa) {
    arc * a = &data->arcs[aa];

    /* NEW CONSTRAINT: lhs expression*/
    XPRBexpr total_backup_req_expr;

    /* for every mapping */
    w_itr = w_useMappingVars.begin();
    for(int cc = 0; cc < data->n_customers; ++cc)
    {
        customer * c = &data->customers[cc];
        for(unsigned int ss = 0; ss < c->services.size(); ++ss) {
            service * s = c->services[ss];
            for (list<mapping*>::iterator m_itr =
                s->mappings.begin(), m_end = s->mappings.end();
                m_itr != m_end; ++m_itr)
            {
                mapping * m = *m_itr;

                /* add backup bandwidth req on arc for used mappings
                *  (multiplied with beta factor)
                */
                total_backup_req_expr += (
                    beta
            *Parameters::Q_BackupBandwidthUsageOnArcForMapping(a, m)
                    *(*w_itr)
                );

                ++w_itr;
            }
        }
    }
```

```
        /* subtract backup reservation on arc variable*/
        total_backup_req_expr -= lambda_arcBackupRes[aa];

        /* create final constraint */
        backupSumCtr.push_back(
            this->master_problem.newCtr(
                XPRBnewname(
                    "backup_sum_ctr_%d_%d",
                    a->startNode, a->endNode
                ),
                total_backup_req_expr <= 0.0
            )
        );
    }
    cout << "DONE! (" << this->backupSumCtr.size() << " created)\n";

    /* PRIMARY OVERLAP CONSTRAINT
     * - if for any arc, the chosen mappings of two services uses that
     *   arc, the two services are said to overlap
     *
     *   for: every pair of two services services
     *
     * NOTE: not needed for dedicated protection scheme as backup
     *       collision will never happen
     */
    if(!this->dedicated) {
        cout << "  - PRIMARY OVERLAP CONSTRAINTS..";
        int primaryOverlapCtrCount = 0;

        /* for every LINK
        * ( for every arc where startNode < endNode )
        */

        // half as many links as arcs
        primaryOverlapCtr.resize(data->n_arcs/2);

        int link_index = 0;
        for(int aa = 0; aa < data->n_arcs; ++aa) {

            arc *a = &data->arcs[aa];

            // one arc per link: arcs where startNode < endNode
            if(a->startNode < a->endNode) {

                primaryOverlapCtr[link_index].resize(
                    data->n_services-1
                );

                /* for every pair of two services (s, t) */
                for(int ss = 0; ss < data->n_services-1; ++ss) {
                    service *s = &data->services[ss];
                    primaryOverlapCtr[link_index][ss]
                        .reserve(data->n_services-1-ss);
                    for(int tt = ss+1; tt < data->n_services; ++tt) {
                        service *t = &data->services[tt];

                        /* NEW CONSTRAINT: lhs expression */
                        XPRBexpr primary_overlap_expr;

                        /* sum over mappings for service s using arc a */
                        for (list<mapping*>::iterator m_itr =
                            s->mappings.begin(),
```

```
                        m_end = s->mappings.end();
                        m_itr != m_end; ++m_itr)
                    {
                        mapping * m = *m_itr;
                        if(
                Parameters::U_PrimaryBandwidthUsageOnArcForMapping(a, m)
                        >= EPS
                        )
                        {
                        primary_overlap_expr +=
                            *mappingVarForMappingIndex(
                                m->globalMappingIndex
                            );
                        }
                    }

                    /* sum over mappings for service t using arc a */
                    for (list<mapping*>::iterator m_itr =
                        t->mappings.begin(), m_end =
                        t->mappings.end(); m_itr != m_end; ++m_itr)
                    {
                        mapping * m = *m_itr;
                        if(
                Parameters::U_PrimaryBandwidthUsageOnArcForMapping(a, m)
                        >= EPS
                        )
                        {
                        primary_overlap_expr +=
                            *mappingVarForMappingIndex(
                                m->globalMappingIndex
                            );
                        }
                    }

                    /* subtract primary overlap var forpair (s,t)*/
                    primary_overlap_expr -=
                        l_servicesOverlapVars[ss][tt-ss-1];

                    /* create final constraint */
                    primaryOverlapCtr[link_index][ss].push_back(
                        master_problem.newCtr(
                            XPRBnewname(
                                "primary_overlap_ctr_%d_%d_%d_%d",
                                a->startNode, a->endNode, ss, tt
                            ),
                            primary_overlap_expr <= 1.0
                        )
                    );
                    ++primaryOverlapCtrCount;
                }
            }

            ++link_index;
        }
    }
    cout << "DONE! (" << primaryOverlapCtrCount << " created)\n";
}

/* BACKUP OVERLAP CONSTRAINT
 * - if two services primary paths overlap, their backup paths
 *   can not have bandwidth requirements at the same arc
 *
```

```
 *   for: every pair of two services services
 *
 * NOTE: not needed for dedicated protection scheme as
 *       backup collision will never happen
 */
if(!this->dedicated) {
    cout << "  - BACKUP OVERLAP CONSTRAINTS..";
    int backupOverlapCtrCount = 0;
    /* for every arc a */
    backupOverlapCtr.resize(data->n_arcs);
    for(int aa = 0; aa < data->n_arcs; ++aa) {
        arc *a = &data->arcs[aa];
        backupOverlapCtr[aa].resize(data->n_services-1);

        /* for every pair of two services (s, t) */
        for(int ss = 0; ss < data->n_services-1; ++ss) {
            service *s = &data->services[ss];
            backupOverlapCtr[aa][ss]
                .reserve(data->n_services-1-ss);
            for(int tt = ss+1; tt < data->n_services; ++tt) {
                service *t = &data->services[tt];

                /* NEW CONSTRAINT: lhs expression */
                XPRBexpr backup_overlap_expr;

                /* sum over all mappings for service s using arc a */
                for (list<mapping*>::iterator m_itr =
                    s->mappings.begin(), m_end = s->mappings.end();
                    m_itr != m_end; ++m_itr)
                {
                    mapping * m = *m_itr;
                    if(
        Parameters::Q_BackupBandwidthUsageOnArcForMapping(a, m)
                    >= EPS
                    )
                    {
                        /* add mapping selection var for mapping */
                        backup_overlap_expr +=
                            *mappingVarForMappingIndex(
                                m->globalMappingIndex
                        );
                    }
                }

                /* sum over all mappings for service t using arc a */
                for (list<mapping*>::iterator m_itr =
                    t->mappings.begin(), m_end = t->mappings.end();
                    m_itr != m_end; ++m_itr)
                {
                    mapping * m = *m_itr;
                    if(
        Parameters::Q_BackupBandwidthUsageOnArcForMapping(a, m)
                    >= EPS
                    )
                    {
                        /* add mapping selection var for mapping */
                        backup_overlap_expr +=
                            *mappingVarForMappingIndex(
                                m->globalMappingIndex
                        );
                    }
                }
```

```
                    /* add primary overlap var for service pair (s, t) */
                    backup_overlap_expr +=
                        l_servicesOverlapVars[ss][tt-ss-1];

                    /* create final constraint */
                    this->backupOverlapCtr[aa][ss].push_back(
                        this->master_problem.newCtr(
                            XPRBnewname(
                                "backup_overlap_ctr_%d_%d",
                                a->startNode, a->endNode
                            ),
                            backup_overlap_expr <= 2.0
                        )
                    );
                    ++backupOverlapCtrCount;
                }
            }
        }
        cout << "DONE! (" << backupOverlapCtrCount  << " created)\n";
    }

    // set problem to maximise objective
    master_problem.setSense(XPRB_MAXIM);

    return;
}
```

F.2 Column Generation Main Loop

The following code sample shows the implementation of the column generation main loop, containing logic common to the three column generation solution methods proposed in this thesis.

```
void CloudBrokerOptimiser::RunColumnGeneration(
    int cg_alg, int cg_maxiters, int cg_maxcount,
    const char *opt_alg, int max_cg_time)
{
    AbstractColumnGenerator * cg;
    bool time_restricted = max_cg_time > -1;

    cout << "- Column Generation By: ";
    switch(cg_alg) {
        case CG_BRUTEFORCE:
            cout << "Brute Force\n";
            cg = new BruteForceColumnGenerator(
                &this->model, this->data
            );
            break;
        case CG_HEURISTIC_A:
            cout << "Heuristic A\n";
            cg = new HeuristicAColumnGenerator(
```

```
                &this->model, this->data
            );
            break;
        case CG_HEURISTIC_B:
            cout << "Heuristic B\n";
            cg = new HeuristicBColumnGenerator(
                &this->model, this->data
            );
            break;
        default:
            cerr<<"undefined cg algorithm choice: "<<cg_alg<<"\n";
            return;
    }
    cout << "-- max iterations: " << cg_maxiters << "\n";
    cout << "-- max mappings: " << cg_maxcount << "\n";
    cout << "-- max cg time: " << max_cg_time << "\n";

    timer run_cg_total_start;
    double lp_time_total = 0.0;
    double cg_time_total = 0.0;
    double run_cg_total_time = 0.0;

    double lp_opt = 0.0;

    this->model.SetColumnGenerationConfiguration(true);

    int itercount = 0;
    while((cg_maxiters <= 0 || itercount < cg_maxiters) &&
          (cg_maxcount <= 0 || data->n_mappings < cg_maxcount))
    {

        ++itercount;
        bool foundColumn = false;

        // solve LP-relaxation
        timer lp_start;
        cout << "\nIteration " << itercount+1
             << ":\n-- running lp-relaxation..\n";
        this->model.RunModel(false, 0, opt_alg);
        lp_time_total += lp_start.elapsed();

        lp_opt = this->model.GetObjectiveValue();
        this->model.SaveBasis();
        dual_vals duals = this->model.GetDualVals();

        cout << "-- Generating Columns..\n";
        timer cg_start;

        // generate columns for every (service, provider) pair
        for(int cc = 0; cc < this->data->n_customers; ++cc)
        {
            customer *c = &this->data->customers[cc];
            for(unsigned int ss = 0;
                ss < c->services.size(); ++ss)
            {
                service *s = c->services[ss];

                if(cg->GenerateColumnsForService(c, s, &duals))
                {
                    foundColumn = true;
                }
            }
```

```
        }

        cg_time_total += cg_start.elapsed();

        if(!foundColumn)
        {
            cout << "\n==> No more columns found"
                 << "column generation stopped\n";
            /* no new column was found -> end column generation */
            break;
        }

        if(time_restricted)
        {
            double current_total_time =
                run_cg_total_start.elapsed();
            if(current_total_time > max_cg_time)
            {
                cout << "\n!! Terminating Column Generation"
                     << " due to time restriction\n";
                break;
            }
        }

        this->model.LoadSavedBasis();
    }

    this->model.SetColumnGenerationConfiguration(false);

    run_cg_total_time = run_cg_total_start.elapsed();

    cout << "\n########## COLUMN GENERATION COMPLETED ##########\n";
    cout << "- Total time:          " << run_cg_total_time << "\n";
    cout << "- Total LP solve time: " << lp_time_total << "\n";
    cout << "- Total CG time:       " << cg_time_total << "\n";
    cout << "- Overhead time:       "
         << run_cg_total_time - cg_time_total - lp_time_total << "\n";
    cout << "- # iterations:        " << itercount << "\n";
    cout << "- # of mappings total: " << this->data->n_mappings << "\n";
    cout << "- LP-optimum value:    " << lp_opt << "\n\n";

    delete cg;
}
```

F.3 Column Generation Brute Force Search

The following code sample shows implementation of core logic of the Column Generation by Brute Force Search method.

```
bool BruteForceColumnGenerator::GenerateColumnsForService(
    customer *c, service *s, dual_vals *duals
)
{
```

```
mapping bestFound;
double best_eval = 0.0;

for(unsigned int pp = 0;
    pp < s->possible_placements.size(); ++pp
)
{
    placement *p = &s->possible_placements[pp];
    for(unsigned int kk = 0; kk < p->paths.size(); ++kk)
    {
        returnPath *k = p->paths[kk];
        // check if feasible mapping alone
        if(k->exp_availability >= s->availability_req) {
            // path offers sufficient availability
            // -> dont add backup path
            mapping m;
            m.primary = k;
            m.backup = NULL;
            double eval = this->_evalMapping(
                &m, s, duals
            );
            if(eval > best_eval) {
                best_eval = eval;
                bestFound = m;
            }
        }
        // OR try combining with other path to
        // placement as backup
        else {
            // not sufficient availability from path
            // -> look for possible backup paths
            for (unsigned int bb = 0;
                bb < p->paths.size(); ++bb
            )
            {
                returnPath * b = p->paths[bb];
                // calculate combo
                // availability [ P(A)*P(B|A) ]
                double k_and_b =
                    entities::prob_paths_a_and_b(
                        k, b
                    );
                if(k->exp_availability
                   + b->exp_availability
                   - k_and_b
                   >= s->availability_req)
                {
                    // combination of a and b is feasible
                    // -> add routing
                    mapping m;
                    m.primary = k;
                    m.backup = b;
                    double eval = this->_evalMapping(&m, s, duals);
                    if(eval > best_eval) {
                        best_eval = eval;
                        bestFound = m;
                    }
                }
            }
        }
    }
}
```

```
    if(best_eval >= EPS) {
        cout << "--> NEW MAPPING (BF): " << best_eval << "\n";
        this->model->AddMappingToModel(&bestFound, s);
        return true;
    }
    return false;
}
```