Optimized Routing and Scheduling of Inter-Compound Transports in Finished Vehicle Logistics

Von der Fakultät für Wirtschaftswissenschaften der
Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades eines Doktors der
Wirtschafts- und Sozialwissenschaften genehmigte Dissertation

vorgelegt von

Dipl.-Ing. Dipl.-Wirt.Ing.
Christian Stefan Franzen

Berichter: Univ.-Prof. Dr. rer. nat. habil. Marco Lübbecke
Berichterin: Univ.-Prof. Dr. rer. pol. Grit Walther

Tag der mündlichen Prüfung: 28. April 2022

Abstract

Distribution planning is a task of utmost importance for finished vehicle logistics that is mostly unexplored by research on mathematical optimization. This thesis proposes one of the first mathematical models for that planning task. Based on a comprehensive literature review, we derive an arc-based MIP formulation and examine complexity and tractability.

We show that the arc-based formulation is cumbersome when introducing additional requirements, concerning the route a vehicle might travel, and that branch-and-cut on that formulation can solve only small problem instances. To overcome the deficiencies of that approach, we develop a path-based formulation using the decomposition technique of Dantzig-Wolfe and solve it using branch-and-price-and-cut. Although that enables us to handle medium-sized and large instances, they cannot be solved to optimality. An important ingredient of each branch-and-price procedure is the algorithm for solving the pricing sub-problem, which is a shortest path problem with resource constraints (SPPRC) in our case. We evaluate multiple algorithms to solve that sub-problem efficiently and apply different acceleration techniques.

Since our exact solution approaches are unable to solve the largest instances appropriately, we try different heuristic techniques to speed up the solution process. The relaxed aggregation of commodities, heuristic pricing of columns and forcing integrality by flow rounding produce near-optimal solutions using significant less computing time. Furthermore, we adapt a heuristic algorithm combining simulated annealing with column generation, that has been proven by other researchers to be successful in solving instances of fixed-charge network design problems (FND).

In addition, we evaluate all approaches within an extensive computational study. The set of benchmark instances is randomly created using our instance generator, which is designed to model the specifics and characteristics of a finished vehicle outbound supply chain. Although we are unable to solve the largest of our

benchmark instances, our algorithms reliably find solutions for network design problems with up to 100 000 vehicles, which forms a solid basis for future research. We expect that the presented framework can be deployed to practice as part of a decision support system (DSS), at least if the size of the network and the amount of considered vehicles are limited.

Zusammenfassung

Die Distributionsplanung ist eine wichtige Aufgabe innerhalb der Logistik von Fertigfahrzeugen und wurde aus Sicht der mathematischen Optimierung bisher kaum wissenschaftlich untersucht. In dieser Arbeit wird eines der ersten mathematischen Modelle für diese Planungsaufgabe vorgeschlagen. Basierend auf einer umfassenden Literaturrecherche leiten wir eine Kanten-basierte MIP-Formulierung her und untersuchen Komplexität und praktische Handhabbarkeit.

Wir zeigen, dass man mit der Kanten-basierten Formulierung nur mühsam zusätzliche Anforderungen an die Route eines Fahrzeugs berücksichtigen kann und dass man durch Branch-and-Cut mit dieser Formulierung nur kleine Probleminstanzen löst. Um die Schwierigkeiten dieses Ansatzes zu überwinden, entwickeln wir unter Verwendung der Dantzig-Wolfe Dekomposition eine Pfad-basierte Formulierung und lösen diese mit Branch-and-Price-and-Cut. Das erlaubt uns zwar das Lösen mittelgroßer und großer Instanzen, es werden aber keine Optimallösungen gefunden. Ein wichtiger Bestandteil jedes Branch-and-Price Ansatzes ist der Algorithmus zur Lösung des Pricing-Subproblems. Dies ist in unserem Fall ein Kürzeste-Wege-Problem mit Ressourcenbeschränkungen (Shortest Path Problem with Resource Constraints, SPPRC). Wir evaluieren mehrere Algorithmen, um dieses Subproblem effizient zu lösen und wenden verschiedene Techniken zur Beschleunigung dieser Algorithmen an.

Da die exakten Lösungsverfahren nicht die größten Instanzen in unserem Benchmark lösen können, versuchen wir mit heuristischen Ansätzen den Lösungsprozess zu beschleunigen. Durch das relaxierte Zusammenfassen von Fahrzeugen, die heuristische Generierung von Spalten und das Erzwingen von Ganzzahligkeit mithilfe von Rundungs-Algorithmen, können nahezu optimale Lösungen mit deutlich weniger Rechenzeit gefunden werden. Darüber hinaus adaptieren wir eine Heuristik, die Simulated Annealing mit Spaltengenerierung kombiniert, von der durch andere Forscher gezeigt werden konnte, dass gute Lösungen für Netzwerkdesignprobleme (Fixed-Charge Network Design Problems, FND) gefunden werden.

Schließlich werden alle genannten Ansätze in einer umfangreichen Simulations-Studie numerisch evaluiert. Die verwendeten Benchmark-Instanzen werden randomisiert mit Hilfe eines Instanzgenerators erzeugt, der die üblichen Charakteristika und Prozesse der Fertigfahrzeug-Logistik berücksichtigt. Obwohl die vorgestellten Verfahren die größten unserer Benchmark-Instanzen nicht lösen können, finden unsere Algorithmen zuverlässig Lösungen für Netzwerkdesignprobleme mit bis zu 100 000 Fahrzeugen, was eine solide Basis für zukünftige Forschung darstellt. Wir erwarten, dass die vorgestellten Lösungsansätze als Teil eines Decision Support Systems (DSS) in der Praxis Anwendung finden, zumindest soweit die Größe des Netzwerks und die Anzahl der betrachteten Fahrzeuge dies zulässt.

Contents

Part I

Problem & Model

1 Introduction

1.1 Routing and Scheduling of Finished Vehicles

Every day there are thousands of finished vehicles leaving the assembly plants that are operated by automobile manufacturers around the globe. Some of these vehicles have already been sold to a customer while others will be stocked. For both scenarios, a complex chain of planning and operation activities is required to ensure that all vehicles will finally reach their destination. In the following, we will

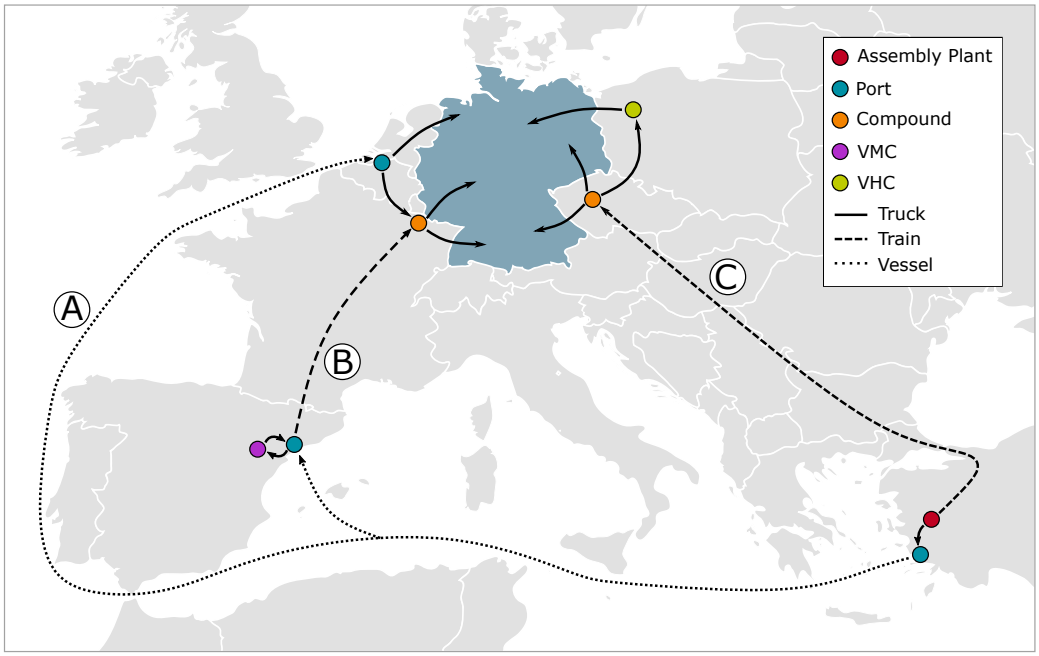


Figure 1.1: Example showing multiple route alternatives for delivering a certain vehicle from an assembly plant in Turkey to sales regions in Germany. There are different types of locations used on the routes, like assembly plants, ports, compounds, vehicle modification centers (VMCs) or vehicle holding centers (VHCs)

look exemplarily at one of these plants that produces vehicles for multiple sales regions in Germany and is located in Turkey. Figure 1.1 shows different routes a vehicle might take after being assembled in that facility. Lets assume the best choice (for whichever objective) would be to use route Ⓐ, going from plant to the port of departure by truck, sailing to the port of arrival in Belgium by roll-on roll-off (RoRo) vessel and delivering to the destination by truck again.

However, route Ⓐ is not suitable for a vehicle that has been damaged, e.g., while being loaded on the vessel, and thus needs repairs. The same holds for a vehicle that requires a modification, like a repainting. Both cases demand a vehicle modification center (VMC), which cannot be reached on that route. Instead, this vehicle will be detoured to route Ⓑ, land at a port in Spain and travel to the nearby VMC. After completing all services it will return to the port and use a train to reach the final distribution compound. Nevertheless, the question is whether detouring is always possible and desirable. It might be a reasonable approach for a single vehicle, but what is the best solution if several vehicles were damaged? Detouring all of them might lead to a downstream capacity shortage, e.g. within the VMC or for the following transports.

Similar conflicts may occur for unsold vehicles that are dedicated to the German market. Moving them to a close by vehicle holding center (VHC) is a reasonable approach to reduce delivery times in case the vehicles are ordered. Such a VHC can only be reached using route Ⓒ, traveling by train to an intermediate compound before being delivered to the VHC in northern Poland by truck. What happens, if a vehicle is sold during its way to that VHC? Obviously, delivering that unit from the intermediate compound to the market is possible, but what about other vehicles that might have been planned for delivery before? Transport capacities might not be sufficient to deliver all of them in time. Therefore the question occurs which of the vehicles needs to be delivered first? Would an initial transport of the vehicle to the VHC and a subsequent delivery from there be faster (or cheaper, depending on the primary logistics objective)?

It is daily business of a distribution planner to provide answers to all of these questions, which can be summarized to:

1. *Routing*: Which route should be used to travel from the vehicle's origin (assembly plant or VHC) to its destination (dealer, customer or VHC)?
2. *Scheduling*: When should each activity on the route start and end?

In this work we develop a solution framework based on mathematical optimization that is able to support a human planner in answering these questions regarding routing and scheduling simultaneously for all finished vehicles. The result of this planning will be a schedule per vehicle similar to Table 1.1. In addition, there are other activities besides transports that may consume a considerable amount of time and thus need to be considered.

Table 1.1: Exemplary schedule for a vehicle incorporating transports and other activities. Besides the type of activity and its start and end date, the involved locations and the logistic service provider (LSP) are listed.

Activity	Location	LSP	Start	End
Transport	Bolu, TR → Kocaeli, TR	Kalepar	22.01.	23.01.
Transport	Kocaeli, TR → Valencia, ES	NYK	23.01.	05.02.
Import	Valencia, ES	Grimaldi	05.02.	06.02.
Holding	Valencia, ES	Grimaldi	06.02.	07.02.
Transport	Valencia, ES → Madrid, ES	A1	07.02.	08.02.
Repair	Madrid, ES	Eurorepar	08.02.	11.02.
Transport	Madrid, ES → Valencia, ES	A1	11.02.	12.02.
Holding	Valencia, ES	Grimaldi	12.02.	14.02.
Transport	Valencia, ES → Diedenhofen, FR	Trasmedi	14.02.	16.02.
Holding	Diedenhofen, FR	Mosolf	16.02.	17.02.
Transport	Diedenhofen, FR → Dealer Frankfurt, DE	Bergé	17.02.	19.02.

1.2 Motivation

The automobile industry has been one of the most important industries in many countries around the world in the past century. Starting with the mass-manufacturing of Ford Model T in the early 1900s, it always has been a driver for macroeconomic growth, employment and technological advances. Thus, it does not come as a surprise that 6.8% of the European Union's gross domestic product (GDP) is generated by the automotive sector and 6.1% of the unions workforce is employed in there. Furthermore, the automobile industry is the largest private investor in R&D in Europe with an annual investment of almost €54 billion and 8,700 granted patents in 2017 (ACEA 2019).

In recent years, the whole industry (and especially the manufacturers) is exposed to continuously changing customer demands. In the course of proceeding urbanization, owning a vehicle or even a driving license is becoming less and less important for

individuals. Within cities there are numerous alternatives like public transport and car or bike sharing. Increasing environmental awareness, the tense parking situation and incidents like the Volkswagen emissions scandal stress vehicles more and more as a "problem", particularly in cities. As a result individual customers want to pay less for a new vehicle or do not want to buy one at all. Besides that, the high transparency induced by the digitalization allows customers to be well informed about market prices. To be profitable, these trends force automobile manufacturers to reduce costs and create strong competition in the new-vehicle market. In contrast, the manufacturers have to master further cost and complexity drivers: environmental certifications, safety standards, the marketing of a growing range of models, presenting as an attractive employer, increasing wages and the development of new technologies, e.g., alternative powertrains (Parment 2016).

The automobile industry has responded to this economical pressure by moving assembly plants to emerging markets (see Figure 1.3). Thus, not only the marginal costs of producing a single vehicle can be reduced, but also a potentially attractive market can be opened up. As a result, next to the traditional automobile countries like Germany, Japan or the United States more nations have evolved to important sales regions, e.g., India, South Korea, Mexico or Brazil. China is the most important market today with almost one-third of global vehicle sales (see Figure 1.4).

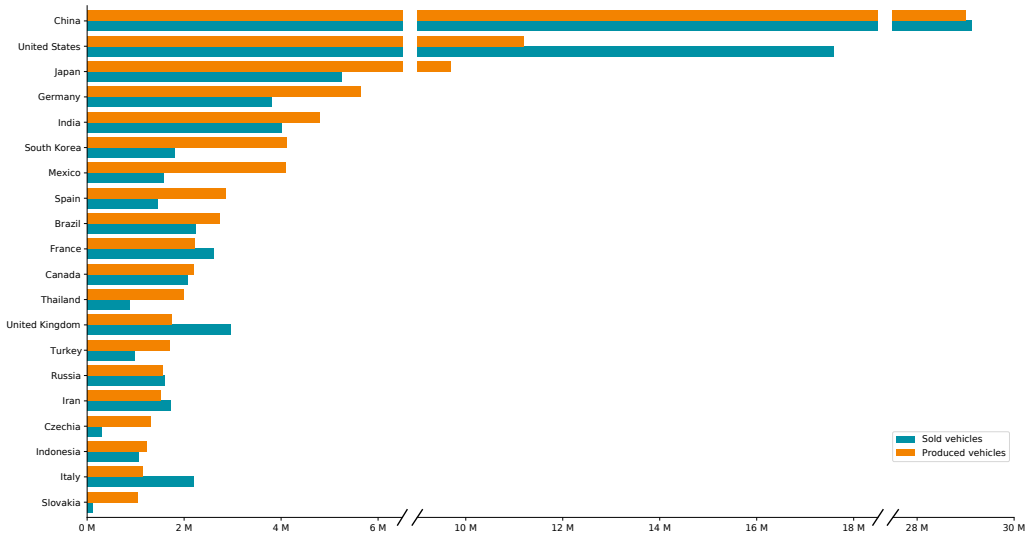


Figure 1.2: Chart showing the amount of sold and produced vehicles in 2017 for all countries with a production of more than one million vehicles. (Source: OICA 2019a; OICA 2019b)

Figure 1.3: Map showing assembly plants of the world's five largest motor vehicle manufacturers in 2016. (Source: Volkswagen Group 2019; MAN SE 2019; Toyota Motor Corp. 2018; Daihatsu Motor Co. 2019; Hino Motors 2019; Hyundai Group 2019; Kia Motors Corp. 2019; General Motors 2019; Ford Motor Company 2019)

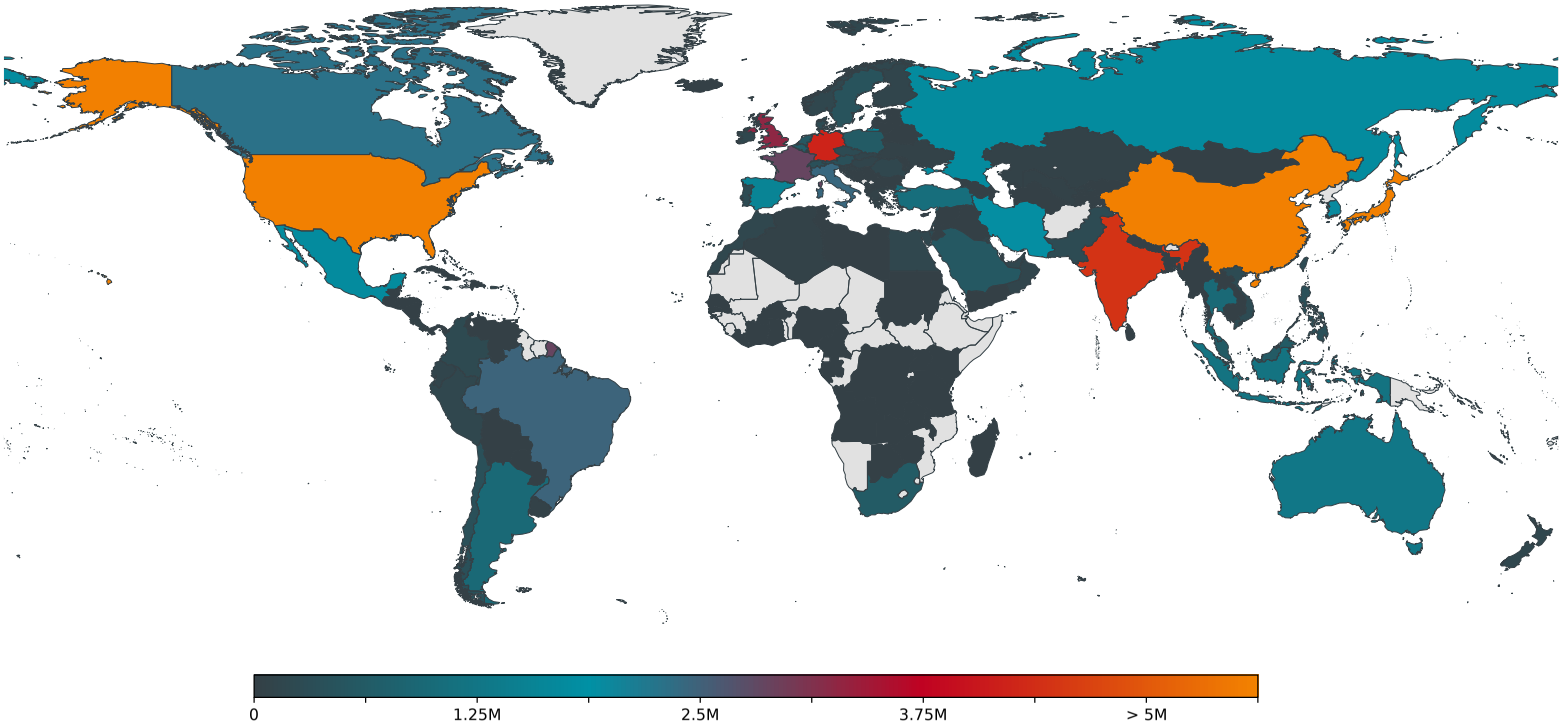


Figure 1.4: Map showing amount of sold vehicles including all brands and manufacturers in 2017 (in millions). For gray areas no statistics where available. (Source: OICA 2019b)

Although these countries have an impressive sales volume on their own, a significant amount of vehicles is exported to (and imported from) foreign countries (see Figure 1.2). This seems to be a reasonable approach when looking at the range of models provided by each automobile manufacturer. The strong competition on the new-vehicle market has led to a few large global players that offer a huge variety of models. The ten largest automobile manufactures produce almost 70% of global vehicle volumes (see Figure 1.5). For example, the Volkswagen group retailed 365 different models using 12 brand names in 2019 (including Ducati motorcycles, Volkswagen Group 2019). Obviously, not all production plants will assemble all of these models. From a manufacturing point of view it is beneficial to produce only very few models at the same assembly plant to accomplish economies of scale and to minimize setup costs. Hence, there will be only a few plants (or even a single one) that assembles a certain model (or platform that is used for different models). Consequently, these models must be exported to all sales regions. Of course, that imposes more responsibility on the outbound supply chain. The average distance from assembly plant to customers increases as well as the overall transport costs and times. Transshipping vehicles more often raises the probability of expensive vehicle damages and complicates predicting an estimated time of arrival both for the customer and for the intermediate transshipment locations. As a result, the involved carriers experience difficulties when planning their transports which increases lead times (and maybe costs) even further (Parment 2016; Haubrich 2017).

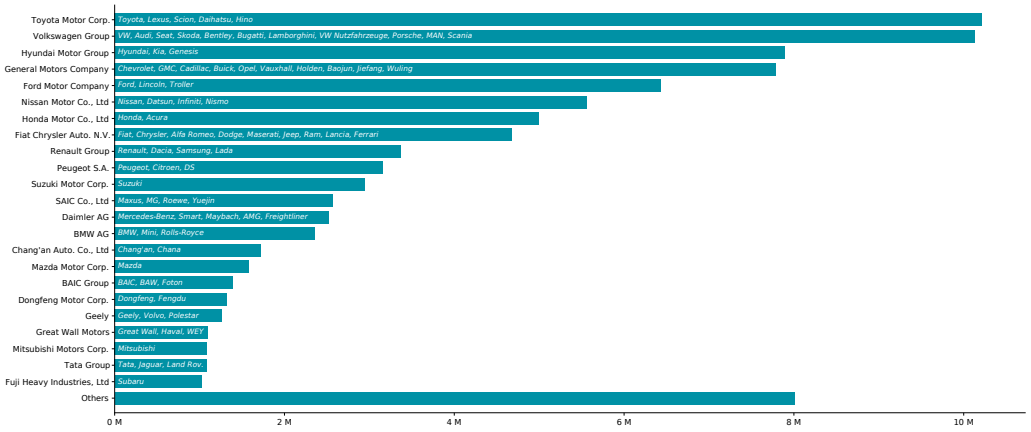


Figure 1.5: Ranking of the world's largest automobile manufactures in 2016 with a production of more than one million vehicles including their brands. (Source: OICA 2019a)

Since the progressive digitalization does not only affect cost sensitivity of the customers, another challenge arises with their expectations regarding delivery. Most individual customers are used to order goods at online shops like Amazon and receive a precise delivery date and possibilities to track the state and location of an order. Furthermore, customers are accustomed to obtain their order within hours or days. These requirements are not unique to the automobile industry but occur in several other industries, too (Wieberneit 2008b). However, such expectations conflict with the increasing delivery times from plant to customer and raises additional pressure on the outbound logistics processes. In some markets like the United States or the United Kingdom, customers demand the delivery of an ordered vehicle within 2 to 3 weeks (Holweg and Miemczyk 2002), German customers are willing to wait for 4 to 6 weeks (Schulz 2014). Besides individual customers also commercial customers, like car rental, leasing or sharing companies, require accuracy regarding delivery dates. Decommissioning old vehicles and replacing them with new ones is part of their daily business and must be done in a synchronized fashion.

One obvious strategy to meet short and accurate delivery times is the usage of local storage locations. Based on sales forecasts the automobile manufacturers push certain amounts of vehicles to the markets and store them at dealerships or VHCs (build-to-stock). In addition, this is beneficial for reaching economies of scale in the assembly plants. However, in many markets the majority of customers (e.g., 74% for the United States, see Holweg and Miemczyk 2002) wants to buy a customized vehicle that is equipped to the customers needs. Such customization can be carried out by moving vehicles on stock to VMCs or by ordering a customized vehicle from plant (build-to-order, see Chandra, Ghosh, and Srivastava 2016). The first option forces the outbound supply chain to take care for customization which increases its complexity, the second option exhibits the drawback of increased delivery times. Furthermore, from a production point of view it might be beneficial to perform some customizations using a VMC, even if the vehicle has been build-to-order (Haubrich 2017).

Managing all of these effects on an outbound supply chain is a highly complex task and requires sophisticated methods to compete against other manufacturers. We want to contribute a novel planning approach to help handling that complexity and operating a highly efficient outbound logistics network.

1.3 Decision Support Systems

Complex planning tasks like the ones outlined above require not only a set of sophisticated methods, but also a huge amount of accurate data. Although we will mainly focus on the methods in this work, we believe that they can only be applied meaningfully by embedding them in a decision support system (DSS) similar to the one presented by Haubrich (2017).

One important task of such a system is collecting all the data that is necessary to use the methods that will be explained in the remainder. Typically, that requires an integration to other enterprise applications of the automobile manufacturer. There are systems used by the sales department that can deliver all the details of an ordered vehicle, like the model, a promised delivery date or requested modifications. The production systems will be able to provide the expected date when the vehicle will leave the assembly plant. And there might be other applications that offer useful information, e.g., concerning quality issues which might lead to modifications of a vehicle.

Besides the data that is necessary for planning, it is important to be able to track the current state of a vehicle. What is its current location? Have modifications already been carried out? Has the vehicle been loaded onto the vessel? Typically, these information can be collected best by integrating with the systems used by the carriers and logistics service providers (LSPs), which operate the compounds and ports. Knowledge about the current state of a vehicle provides important information to evaluate deviations from the original plan. Due to the high complexity of the outbound supply chain it is very probable that such plan amendments will occur. There will always be incidents that require short-term changes. And the cause of these incidents might not even be rooted in distribution operations, like an ordering dealer that faces financial difficulties, which leads to vehicles being put “on hold”. Additionally, there might be effects that cannot be reflected appropriately during planning. Whatever is the reason for such deviations, in our experience finished vehicle logistics can be performed most efficiently if there exists a recurring loop of planning, tracking and adoption of the plan. This implies that the current plan is communicated regularly to all participants of the outbound supply chain, e.g., for requesting transport capacities from one of the carriers, including an option for feedback.

Our recommendation for the usage of a DSS and its integration with all the other enterprise applications via EDI-connections could give the impression that we suggest to fully automate the entire outbound supply chain. Although that seems to be desirable from an efficiency point of view, in our experience that is just not possible. It would be presumptuous to believe that the real-world complexity of a supply chain can be mapped to a mathematical model or to an algorithm. We have witnessed how issues that nobody had thought of before could be resolved by a human planner just by doing the right phone call. We believe today's computer systems are unable to react similarly, even when introducing advanced technologies like artificial intelligence or machine learning. In contrast, we propose that the flexibility and knowledge of human planners should be used in combination with a computer system that supports their daily business. In the best case, planners will be involved most of their time in resolving edge-cases that cannot be handle automatically. Nevertheless, the planners should be in control of the system, which enhances acceptance and finally leads to a more efficient operation of the logistics network. For that, a human decision must always be preferred over a decision taken automatically in practice.

1.4 Outline

To develop satisfying answers to the questions we have asked so far, this thesis is divided into four parts. The first part deals with the considered problem domain and the corresponding mathematical models. We start with an introduction to distribution planning for finished vehicles in Chapter 2 and explain the major requirements that need to be taken into account for that planning task. Afterwards, in Chapter 3 we clarify some preliminary definitions that will be used throughout this work and will take care for a common understanding of notations. Chapter 4 summarizes the existing literature contributions in finished vehicle logistics and for mathematical optimization in freight transportation in general. Finally, we present one of the first mathematical models for distribution planning in an automotive outbound supply chain in Chapter 5. That includes an arc-based MIP formulation and some notes regarding complexity, feasibility and tractability.

The second part focuses on exact solution methods. In Chapter 6 we discuss the problem of finding an optimal route for a single vehicle, which turns out to be a shortest path problem with resource constraints (SPPRC). We evaluate multiple

algorithms to solve that problem and apply different techniques to accelerate the path search, primarily by using a novel approach that depends on the relations of a network to its time-expanded counterpart. In Chapter 7 we outline a branch-and-cut approach and present a detailed literature review of valid inequalities and their separation algorithms. Unfortunately, the arc-based formulation turns out to be cumbersome when introducing additional requirements concerning the route a vehicle might travel. Furthermore, performing branch-and-cut on that formulation is successful only for small and some medium size instances. To overcome the deficiencies of that approach, we develop a path-based formulation of our problem, using the decomposition technique of Dantzig and Wolfe (1960). We successfully solve all medium-sized and some large instances of the resulting MIP by a branch-and-price-and-cut algorithm, which will be introduced in Chapter 8.

Since our exact solution approaches were unable to solve the largest instances appropriately, in the third part of this thesis we introduce different heuristic techniques to speed up the solution process. We show under which conditions finished vehicles can be aggregated to a single commodity (without losing optimality) in Chapter 9 and how these conditions can be relaxed to obtain considerable smaller MIPs. Furthermore, in Chapter 10 we explain a heuristic path search algorithm that can be used to accelerate pricing within our branch-and-price-and-cut approach. Additionally, Chapter 11 reveals several flow rounding algorithms, that are able to create a near-optimal integral flow from a fractional one and should be preferred to branching. Finally, we close that part in Chapter 12 with the selection and adaption of a heuristic algorithm that has been originally build for the fixed-charge network design problem (FND) and combines simulated annealing and column generation.

To draw a final conclusion, we will provide an extensive computational study in part four. We start in Chapter 13 with an introduction of our benchmark instances, the associated instance generator and the computing environment. Furthermore, we give some hints regarding used methodologies, notations and performance metrics. In Chapter 14 we evaluate the performance of all path search algorithms and the corresponding acceleration techniques. While Chapters 15 to 17 study the performance of our branch-and-cut and branch-and-price-and-cut algorithms, in Chapter 18 we explain the computational experiments and their results for all heuristic approaches. Finally, we present our conclusion in Chapter 19 and close with an outlook to future research in Chapter 20.

2 Finished Vehicle Logistics

2.1 Distribution Planning

Research on automotive supply chains has spanned over several decades and remains the focus of many researchers and practitioners today (Holweg and Miemczyk 2002; Iijima and Sugawara 2005; Gunasekaran and Ngai 2005; Mattfeld 2006; Suthikarnnarunai 2008; Vecchiato 2012; Chandra, Ghosh, and Srivastava 2016). Typically, these supply chains are centered on the automobile production facilities and are split into three distinct segments (Haubrich 2017):

- **Inbound:** Logistic processes moving material required for production from suppliers to the assembly plants
- **In-plant:** All processes and material flows that occur during production within an assembly plant
- **Outbound:** Logistics for finished vehicles on their way from the assembly plant to the dealer or customer

Finished vehicle logistics cover all aspects of the outbound supply chain and will be the main focus of this thesis. There are transports that need to be carried out on road, rail, sea or river (in rare cases in the air, too). Vehicles must be (un-)loaded, stored or processed in a VMC, at intermediate transshipment locations. And there might be additional services like washing, refueling or pre-delivery inspections required. All of these activities call for careful planning to ensure sufficient resources and capacities whenever they are needed and minimize overall logistics costs.

In supply chain management, the planning process will be typically considered at different levels of aggregation and planning periods ranging from “aggregated long-term to detailed short-term planning” (Stadtler 2005). This seems to be a reasonable approach, taking into account that some decisions must be taken with longer lead times but with less details available, while other decisions can be

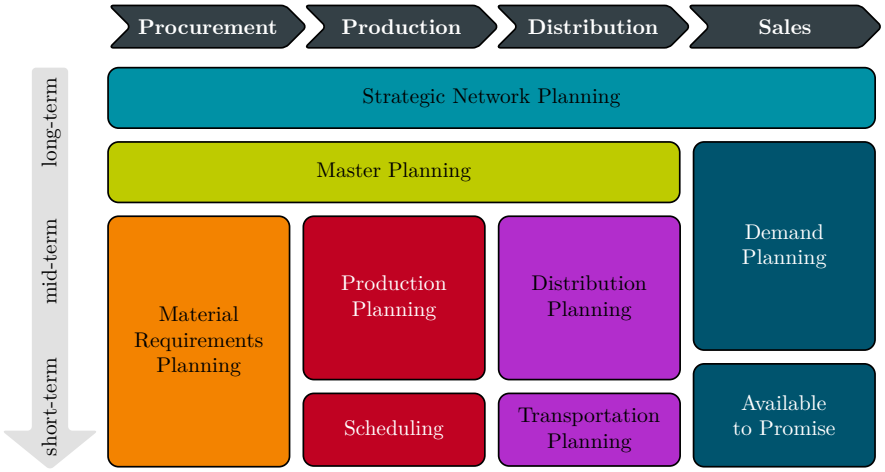


Figure 2.1: Visualization of the Supply Chain Planning Matrix according to Rohde, Meyr, and Wagner (2000).

delayed until more details are known. The classical model of the Supply Chain Planning Matrix (see Figure 2.1) considers different planning levels and a hierarchy between them (Rohde, Meyr, and Wagner 2000; Stadtler, Kilger, and Meyr 2015, p. 77). Focusing on the outbound supply chain only, the following decisions must be evaluated by each automobile manufacturer (compare also with Haubrich 2017).

Long-term strategic network planning On this level all decisions with a planning period of several years (up to 12 years) are regarded (Stadtler, Kilger, and Meyr 2015, p. 108). Typically, these involve significant investments in infrastructure and cannot be undone without substantial costs. Questions concerning the introduction of new models, capacity expansions of plants or (owned) distribution or storage centers need to be discussed here. Furthermore, the allocation of models to one or several plants and the selection of strategic partners (e.g., suppliers or LSPs) is an essential part of this level. The major challenge to face is the uncertainty of forecasts inherent to such a long planning period (Kauder and Meyr 2009).

Mid-term master planning Based on strategic decisions and a detailed demand forecast (normally with a granularity of weeks, see Meyr 2009) decisions regarding the implementation of a distribution network need to be carried out on this level. Typically the planning period is one year (Stadtler, Kilger, and Meyr 2015, p. 158). The major result of this planning will be contracts with LSPs agreeing on annual

capacities and rates for transports or other services (storage, loading, modifications, etc.). Furthermore, the usage of transshipment locations, VMCs and VHCs, that are normally operated by LSPs, will be fixed in conjunction with the routes a vehicle might take from each plant to a certain delivery region. The higher degree of details allows a budget calculation of overall distribution costs that will be the benchmark for all short-term operations within this planning period (Meyr 2009).

Short-term transportation planning In contrast to the preceding levels, transportation planning will not be done on forecasts of vehicle volumes but requires "real" vehicles. The routes of these vehicles have been fixed in a previous planning phase. Similarly, information about required services and their scheduled time and corresponding VMC are given. The automobile manufacturers have ordered transports or services from the LSPs using rates and conditions agreed upfront in master planning. So in most cases, transportation planning will be done by the LSPs. If trucks will be used for transports, an assignment from vehicles to trucks must be determined. This requires the planning of truck tours (vehicle routing, see Toth and Vigo (2014) for a comprehensive survey) as well as the consideration of proper loading of trucks (vehicle loading), which can be a complex task for an auto-carrier (Agbegha, Ballou, and Mathur 1998). The loading is as well challenging both for trains and vessels. Here constraints like the physics of the vessel on the water or low-hanging trees close to the rail track must be taken into account.

Distribution planning Transforming the volume-based routes from mid-term master planning to actual transport and service orders required by transportation planning is a task that needs to be performed by every automobile manufacturer in order to actually operate the outbound supply chain. According to Rohde, Meyr, and Wagner (2000) this task is called *distribution planning*. A straightforward method to carry out distribution planning, used by several automobile manufacturers today, is the creation of *Freight Movement Plans* (Powell and Sheffi 1989). Such a plan consists of rules similar to:

> Vehicles arriving at transshipment location A
> with final destination C
> must be forwarded to location B.

Formulating all distribution decisions in a similar way significantly simplifies the overall planning process, as it allows to separate decisions for different vehicles or different locations. It is possible to plan the next step of a vehicle's journey without considering other vehicles. It would be even imaginable to delegate decisions to all transshipment locations and allow local operators to carry out the distribution planning independently. Furthermore, automatically creating transport or services orders and transmitting them to the LSPs using EDI-connected computer software is simple and a best practice used by many automobile manufacturers.

Freight movement plans are a powerful tool that will work for the majority of vehicles. Nevertheless, in many cases their usage will imply drawbacks or difficulties. There might be sales regions where the route of a vehicle has not been uniquely determined in mid-term planning, so that there are alternative routes available. Formulating that as one of the rules explained before can be hard and most likely requires another decision criterion, like a split factor. Besides, a vehicle might have been damaged and needs to be detoured from its pre-defined route for repairs. Or a vehicle could already be delayed on its route and needs to be sped up. Typically, that can be achieved by using a different route or switching to another means of transport. All of these decisions must be taken while considering the actual global state of the distribution network. It is essential to know current and future utilization of each transport leg and free capacities in VMCs, since each decision might impact other vehicles.

Additionally, when using freight movement plans only, it is hard to efficiently consolidate transports. Such consolidations can be done in a twofold manner. Moving vehicles from different origins to a consolidation location and jointly shipping them to the next destination is called *spatial* consolidation. In contrast, vehicles are kept at inventory and accumulated with later arriving vehicles to achieve *temporal* consolidation. Of course both modes can occur in combination. Since tariffs of transport carriers usually exhibit a structure that per-unit-costs decreases with increasing amount of vehicles in a load (economies of scale), consolidation can heavily influence overall logistics costs. Consequently, distribution planners are asked to build vehicle loads that utilize the means of transport to capacity. However, in many cases an accurate forecast, which vehicles will arrive next at a location and the ability of detouring vehicles to exhaust capacities is missing. So usually the built vehicle loads will only contain vehicles that are already available at the origin location. Consolidation implies finding a balance between efficient

usage of economies of scale, short transportation routes and low inventory costs. It requires an overview of the entire distribution network. We follow the view of Matuschke (2013) that only the automobile manufacturer is able to gather all required information to efficiently consolidate vehicle transports.

Since distribution planning using freight movement plans shows the aforementioned weaknesses, in this work we will focus on presenting an alternative approached based on mathematical optimization and we will evaluate different algorithmic solution methods that can assists a distribution planner, when taking routing and scheduling decisions for a vehicle.

2.2 Problem Domain

Following the high-level introduction of finished vehicle logistics and distribution planning in the previous sections, we will continue now with a detailed explanation of the problem domain considered in this thesis. The central tasks of distribution planning for finished vehicles are

1. *Routing* each vehicle from its origin to its destination location and
2. *Scheduling* each activity on that route, while considering the schedule of other vehicles and available capacities.

Besides transports, the term "route" covers also additional services (e.g., within a VMC) that will be performed at intermediate locations. Therefore, in the following a route will be considered just as a sequence of services, where a service might change the location of a vehicle (transports). Of course, most of the time there will be many different possibilities how such a scheduled route could look like for a single vehicle. All of them must meet certain requirements in order to be feasible for logistics operations.

First and most obviously, all services are limited in their capacity. This does not apply to transports only but also to VMC and storage services within a compound. Usually, capacities must be considered as fixed, although in some cases they can be extended by spot-buying additional capacities (unplanned, short-term contracts) for sometimes considerably higher costs. Furthermore, capacities may vary over the planning horizon. That is especially true for means of transport that operate based on a predefined schedule. For example, there will not be a RoRo vessel

departing from a port every day. So outgoing transport capacities from that port are linked to the vessel schedules.

It is important to distinguish in this discussion between the physical capacity of a means of transport and the available capacity that has been agreed on in a contract during master planning (see Section 2.1). Only the latter can be considered here. Unfortunately, contracted capacities are most probably not given in a granularity sufficient for distribution planning, as it is inconceivable for a LSP to plan on such level of detail months ahead. This is especially the case, if the LSP is contracted by several automobile manufacturers. Instead, most likely unspecific patterns have been contracted like "overall capacity per week" or "capacity on four days per month". For distribution planning these patterns must be transformed to capacities of the required granularity, e.g., capacity per day or of a dedicated transport. When planning initially for a period of time, simple rules can be used to calculate fine-grained capacities, like distributing the weekly capacity evenly on each day of the week. However, ensuring that these capacities can be provided by the involved LSPs is essential for the reliability of the planning process. Thus, an iterative planning approach incorporating the feedback of the LSPs regarding planned capacities would highly improve prior estimations. Whatever approach is used by an automobile manufacturer, for the remainder of this work we ignore this issue and assume that all capacities are known in the required granularity.

As in the example of Figure 1.1, the possible routes for a single vehicle that can be chosen in distribution planning have been predetermined in master planning. Depending on the desired flexibility it might be beneficial to allow deviations from these routes. Why should a vehicle be forced to use route Ⓐ if route Ⓑ (without the VMC) is underutilized and detouring would speed up delivery? As finished vehicle logistics is a highly complex process and there are many things that can go wrong, ordinarily it is favorable to allow certain degrees of freedom to be able to cope with operational issues or use available capacities more efficiently. So instead of fixing the route a vehicle will take, in the following we allow using a predetermined part of the distribution network. However, there might be certain requirements that cannot be relaxed. If visiting a VMC or VHC is necessary, a vehicle cannot travel a route without one. Furthermore, there might be precedence relationships between services on the vehicles route, e.g., when using a vessel transport it is required to perform an inspection service in the port of arrival to verify that the vehicle has not been scratched. As long as a route from the

sub-network complies with these requirements it can be used to deliver a vehicle.

Besides the aforementioned limitations regarding space there are also constraints related to time. Performing any service on a route will consume a predefined period of time. Here again we need to distinguish between times that have been contracted in a service level agreement (SLA) and physical processing times. From the perspective of an automobile manufacturer actual processing times usually are negligible for distribution planning. There will be always a certain time-span between calling a transport carrier and the actual pick-up of vehicles. Transport times might be unpredictable due to traffic jams, train failure or troubled sea. Furthermore, the carrier might be allowed to do consolidations of transports on its own, so there could be an additional delay between pick-up and start of the actual delivery. When contracting SLAs with a carrier, all these dwell times combined with an estimated transport time are considered. The same applies to other services within a compound. Ultimately, contracted service levels will be balanced between completing a service as fast as possible and offering the LSPs flexibility to allow them efficient (short-term) transport planning. In the following, we treat these service levels as the maximum time-span between calling a LSP to perform a service and its actual completion.

For sold vehicles there will usually be an agreement with the ordering dealer or customer when the vehicle should be delivered. For most customers it is essential to be able to rely on these target delivery dates (TDDs). Besides minimizing overall logistics costs, delivering in time is a fundamental objective of distribution planning. Depending on the customer, there might be different interpretations of the TDD. In some cases, it might be acceptable to deliver as soon as possible, in other cases delivering before the TDD might be an issue. It is impossible to ensure timely delivery in all situations (e.g., a vehicle has a severe damage). However, such situations should be avoided whenever feasible. Please note that typically the TDD is agreed on with the ordering dealer or customer and hence only the delivery to the final location is of interest. Delays occurring at intermediate locations are negligible as far as they will not affect the date of the final delivery.

Since distribution planning is intended to connect mid-term master to short-term transport planning (see Section 2.1), a decent level of details is required. Thus, the planning horizon is limited to the near future, where such details are available. However, the LSPs ask for a reasonable forecast time to be able to plan their activities, so the planning horizon may not be too short. Finding a balance between

these two requirements, from our experience we expect that all vehicles leaving their assembly plant in the upcoming 3 to 4 weeks will be considered during distribution planning. Of course, all vehicles already traveling to their final destination must be taken into account as well.

As mentioned before, minimizing costs is one of the essential objectives when planning the distribution of finished vehicles. Obviously, there are different cost drivers within the outbound supply chain. On the one hand there are transport services that are offered by the carriers at different tariffs. Usually, their cost structure depends on the nature of the contract, the means of transport and the amount of vehicles to ship. On the other hand substantial logistics costs can arise within a compound. Figure 2.2 shows an example of a compound consisting of rail tracks (orange), vessel quays (blue), truck loading lanes (red), a VMC (green) and a VHC. There are multiple activities in a compound that imply costs like (un-)loading a means of transport, storing a vehicle for a period of time, preparing customs import and export or performing modifications and additional services in a VMC (e.g., repainting, damage repair, washing, refueling or battery recharging). Typically, these compound services are charged per vehicle. However, there might

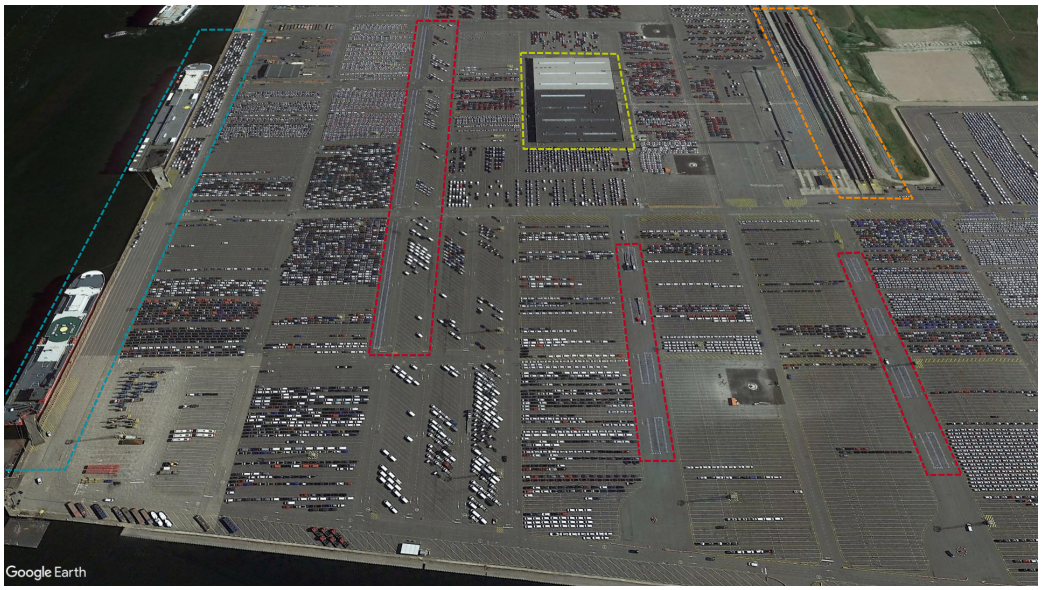


Figure 2.2: Example showing a compound with a rail track, a vessel quay and a VMC. Vehicles are stored in lanes or a fish-bone pattern and moved around for loading / unloading and services (Source: satellite image from Google Earth 2021).

be discounts when ordering for larger volumes. Summarizing the various types of costs for transport and compound services, we assume in the following that three different cost structures are sufficient to support the most common cases occurring in finished vehicle logistics:

1. **Overall costs increase linearly with each additional vehicle**. So there is a charge per vehicle defined in the contracts with the LSPs. Costs for handling vehicles within a compound normally will have this structure.

2. As a variation of the first bullet point, **the cost per vehicle may decrease with an increasing quantity of vehicles**. There may be continuous volume discounts or some kind of bulk prices.

3. Sometimes the amount of vehicles is irrelevant for calculating overall costs. Instead **there will be only fixed costs** for providing a service at a point in time. This holds true for ordering full truck loads, train wagons or for spot-buying an additional means of transport. However, there may be a discount, when ordering multiple loads at once, so the costs per means of transport may decrease.

For all these cases we assume that they comply with the economies of scale principle, so ordering services of a certain volume never will be more expensive than splitting the same order in multiple ones.

Finally, we want to highlight that there might be other cost structures that will not be considered within this thesis. There are cases where costs change when a service is demanded repeatedly within a period of time. For example, storage costs for a vehicle might increase if it is stored for more than a predefined amount of days. Or carriers might grant a bonus if a certain amount of transport services is offered from that carrier within the same month. In other rare cases only the maximum transport distance of the vehicles in a truck load determines the costs for all vehicles in that load. Since such complex tariffs can hardly be modeled in all facets and – to the best of our knowledge – are uncommon, we will not consider them in this thesis.

3 | Preliminary Definitions

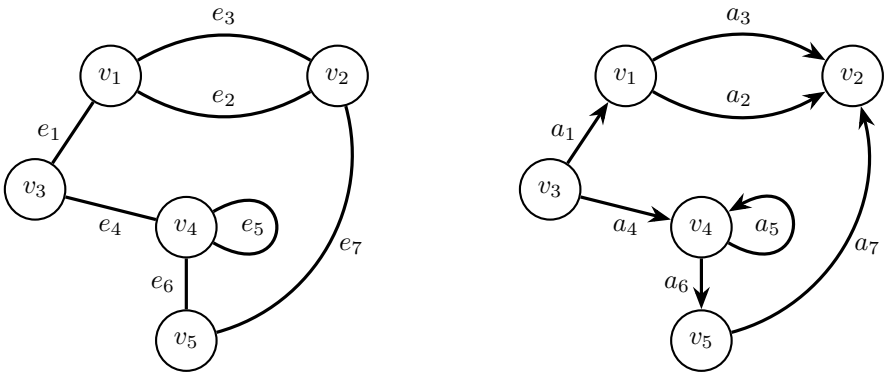
Before we start to present a comprehensive literature review and a mathematical model of the problem explained in Chapter 2, we first will give some preliminary definitions that will be used later on and should guarantee a common understanding of the used notation. More details can be found in the standard books of Zwillinger (2018), Diestel (2017), Bang-Jensen and Gutin (2008), Schrijver (2003), and Ahuja, Magnanti, and Orlin (1993).

3.1 Graphs

Modeling problems occurring in a supply chain is a challenging task, however networks provide a sufficient and widely used theory for that purpose. They are typically represented by graphs, which will be introduced in this section.

Undirected graphs An *undirected graph* $\mathcal{G} = (V, E)$ is a pair of finite sets V and E. The elements of set V are typically called *vertices*. Each element e of set $E \subseteq V \times V$ is called an *edge* and is defined as an unordered pair of vertices $e = \{ v, w \}$ which are called its *end points*.

An edge e with identical endpoints is called a *loop* or *self-loop*. Two edges e_1, e_2 are *parallel* edges if $e_1 = e_2$. An undirected graph that permits loops and parallel edges is called an *undirected pseudo-graph* otherwise it is called *simple*. In the following we will always consider an undirected graph to be a pseudo-graph, unless otherwise specified. A graph containing an edge for every pair of vertices is called *complete*. The *complement* $\bar{\mathcal{G}}$ of $\mathcal{G}$ is the graph on vertices V containing an edge between all pairs of vertices that have no edge within $\mathcal{G}$. An example on an undirected graph has been visualized in Figure 3.1a.



(a) Undirected pseudo-graph (b) Directed pseudo-graph

Figure 3.1: Visualization of (a) undirected and (b) directed pseudo-graphs defined by $\mathcal{G} = (V, E)$ and $\mathcal{D} = (V, A)$ with set of nodes or vertices $V = \{ v_1, v_2, v_3, v_4, v_5 \}$, set of edges $E = \{ e_1, e_2, e_3, e_4, e_5, e_6, e_7 \}$ and set of arcs $A = \{ a_1, a_2, a_3, a_4, a_5, a_6, a_7 \}$.

Directed graphs A *directed graph* or *digraph* $\mathcal{D} = (V, A)$ is a pair of finite sets V and A. Similar to the definition above the elements of set V are called *vertices* or *nodes*, while the elements of set A are called *edges* or *arcs*. To avoid confusion, we use the notion of nodes and arcs in the following to distinguish from vertices and edges in the undirected case. Each arc $a \in A$ is defined as an ordered pair of nodes $a = (v, w)$ with $v, w \in V$ which are called its start and end node. We use the notation $v = \mathit{tail}(a)$ and $w = \mathit{head}(a)$ to identify either of these nodes.

An arc a with identical start and end node ($\mathit{tail}(a) = \mathit{head}(a)$) is called a *loop* or *self-loop*. Two arcs a_1, a_2 are *parallel* if $\mathit{tail}(a_1) = \mathit{tail}(a_2)$ and $\mathit{head}(a_1) = \mathit{head}(a_2)$ or *anti-parallel* if $\mathit{tail}(a_1) = \mathit{head}(a_2)$ and $\mathit{head}(a_1) = \mathit{tail}(a_2)$. A directed graph that permits loops and (anti-)parallel arcs is called a *directed pseudo-graph* otherwise it is called *simple*. In the following we will always consider a directed graph to be a pseudo-graph, unless specified differently. A digraph containing a pair of anti-parallel arcs for every pair of nodes is called *complete*. The *complement* $\bar{\mathcal{D}}$ of $\mathcal{D}$ is the directed graph on nodes V containing an arc between all pairs of nodes that have no arc within $\mathcal{D}$. An example of a directed graph has been visualized in Figure 3.1b.

Directed vs. undirected graphs On the one hand, constructing the *underlying undirected graph* $\mathcal{G}_\mathcal{D} = (V_\mathcal{G}, E)$ of a directed graph $\mathcal{D} = (V_\mathcal{D}, A)$ can be done

by choosing the vertex set $V_{\mathcal{G}} = V_{\mathcal{D}}$ and the edge set $E = \{ e_a \mid a \in A \}$ with $e_a = \{ tail(a), head(a) \}$.

On the other hand, the corresponding *bidirected digraph* for the undirected graph $\mathcal{G} = (V_{\mathcal{G}}, E)$ is the graph $\mathcal{D}_{\mathcal{G}} = (V_{\mathcal{D}}, A)$ where $V_{\mathcal{D}} = V_{\mathcal{G}}$ and $A = \{ a_e^{\rightarrow}, a_e^{\leftarrow} \mid e \in E \}$ with $tail(a_e^{\rightarrow}) = head(a_e^{\leftarrow})$, $tail(a_e^{\leftarrow}) = head(a_e^{\rightarrow})$ and $e = \{ tail(a_e^{\rightarrow}), head(a_e^{\rightarrow}) \}$. So each edge $e \in E$ will be replaced by two anti-parallel arcs $a_e^{\rightarrow}$ and $a_e^{\leftarrow}$.

Subgraphs and Cliques The graph $\mathcal{H} = (V_{\mathcal{H}}, E_{\mathcal{H}})$ is a *subgraph* of $\mathcal{G} = (V, E)$ if $V_{\mathcal{H}} \subseteq V$, $E_{\mathcal{H}} \subseteq E$ and each edge of $E_{\mathcal{H}}$ has both endpoints in $V_{\mathcal{H}}$ (so for all $e \in E_{\mathcal{H}}$ holds $e \subseteq V_{\mathcal{H}}$). If all edges from the set E with both endpoints in $V_{\mathcal{H}}$ are elements of set $E_{\mathcal{H}}$ we say graph $\mathcal{H}$ is *induced* by $V_{\mathcal{H}}$ and we call $\mathcal{H}$ an *induced subgraph* of $\mathcal{G}$. In analogy to the undirected case the terminology of a subgraph and an induced subgraph can be defined for directed graphs.

A complete induced subgraph is named a *clique*. A *maximal clique* is a clique that cannot be extended by additional vertices that are adjacent to vertices of the clique. The maximal clique of a graph with the most vertices is denoted the *maximum clique* and its number of vertices is called the *clique number* $\omega(\mathcal{G})$ of graph $\mathcal{G}$. Partitioning the vertices of a graph into cliques leads to a *clique cover*. A clique cover that consist of as few cliques as possible is a *minimum clique cover*. The amount of cliques in that minimum clique cover is defined as the *clique cover number* $\chi(\mathcal{G})$ of graph $\mathcal{G}$.

Cuts, Degrees and Neighborhoods Let $\mathcal{G} = (V, E)$ be an undirected graph. A vertex $v \in V$ and an edge $e \in E$ are *incident* if $v \in e$. Two vertices $v, w \in V$ are *adjacent* or *neighbors* if there exists an edge $e \in E$ incident to both vertices ($e = \{ v, w \}$). The set of all neighbors of v is called the *neighborhood* of v and denoted by $N_{\mathcal{G}}(v)$.

For $S \subset V$ the *cut* that is *induced* by S is the set

$$\delta_{\mathcal{G}}(S) \coloneqq \{ e \in E \mid e \cap S \neq \emptyset \wedge e \cap V \setminus S \neq \emptyset \}.$$

To simplify notation, for a single vertex $v \in V$ we will also write $\delta_{\mathcal{G}}(v)$ instead of $\delta_{\mathcal{G}}(\{ v \})$. The *degree* of vertex $v \in V$ is $|\delta_{\mathcal{G}}(v)|$.

Let $\mathcal{D} = (V, A)$ be a digraph. A node $v \in V$ and an arc $a \in A$ are *incident* if $v = tail(a)$ or $v = head(a)$. Two vertices $v, w \in V$ are *adjacent* or *neighbors* if there exists an arc $a \in A$ incident to both vertices $(a = (v, w) \vee a = (w, v))$. The set of all neighbors of v is called the *neighborhood* of v and denoted by $N_{\mathcal{D}}(v)$.

For a non-empty subset $S \subset V$ of vertices the *cut induced by* S is the set $\delta_{\mathcal{D}}(S) = \delta^+_{\mathcal{D}}(S) \cup \delta^-_{\mathcal{D}}(S)$ with

$$\delta^+_{\mathcal{D}}(S) \coloneqq \{\, a \in A \mid tail(a) \in S \wedge head(a) \in V \setminus S \,\}$$
$$\delta^-_{\mathcal{D}}(S) \coloneqq \{\, a \in A \mid tail(a) \in V \setminus S \wedge head(a) \in S \,\}.$$

The cut is *directed* if $\delta^+_{\mathcal{D}}(S) = \emptyset$ or $\delta^-_{\mathcal{D}}(S) = \emptyset$. Again, for a single vertex $v \in V$ we will also write $\delta_{\mathcal{D}}(v)$, $\delta^+_{\mathcal{D}}(v)$ or $\delta^-_{\mathcal{D}}(v)$ instead of $\delta_{\mathcal{D}}(\{\, v \,\})$, $\delta^+_{\mathcal{D}}(\{\, v \,\})$ or $\delta^-_{\mathcal{D}}(\{\, v \,\})$. The *degree* of vertex $v \in V$ is $|\delta_{\mathcal{D}}(v)|$, its *out-degree* is $|\delta^+_{\mathcal{D}}(v)|$ and its *in-degree* is $|\delta^-_{\mathcal{D}}(v)|$. If there is no ambiguity, we will omit the subscripts $\mathcal{G}$ and $\mathcal{D}$ of $N_{\mathcal{G}}$, $N_{\mathcal{D}}$, $\delta_{\mathcal{G}}$, $\delta_{\mathcal{D}}$, $\delta^+_{\mathcal{D}}$ and $\delta^-_{\mathcal{D}}$.

Paths and cycles A *path* of *length* k is a finite, non-empty sequence $P = (v_0, e_1, v_1, \ldots, e_k, v_k)$ of alternating vertices and edges of graph $\mathcal{G} = (V, E)$ with $v_0, v_i \in V, e_i \in E, e_i = \{\, v_{i-1}, v_i \,\}, 1 \leq i \leq k$. A path can also be defined for a directed graph $\mathcal{D} = (V, A)$ by $P = (v_0, a_1, v_1, \ldots, a_k, v_k)$ with $v_{i-1} = tail(a_i) \wedge v_i = head(a_i)$ or $v_{i-1} = head(a_i) \wedge v_i = tail(a_i)$ $(v_0, v_i \in V, a_i \in A, 1 \leq i \leq k)$. If $v_{i-1} = tail(a_i) \wedge v_i = head(a_i)$ holds for all $1 \leq i \leq k$ the path is called *directed*. For the directed and the undirected cases we will also use the notion v_0v_k-path. If there is no ambiguity, we might omit writing nodes and edges (or arcs) of a path and use only one of both.

We call a (directed) path *simple*, if it does not contain any edge or arc more than once. It is *elementary*, if all vertices of the path are unique. Please note that an elementary path is always simple. An elementary (directed) v_0v_k-path with length $k \geq 2$ and $v_0 = v_k$ is called a *(directed) cycle*. Cycles of length less than or equal to k are denoted *k-cycles*. A graph containing any cycle is called a *cyclic graph*, otherwise it is called *acyclic*.

Connectivity and components A graph $\mathcal{G} = (V, E)$ is *connected* if for all pairs of vertices $v_i, v_j \in V$ $(i \neq j)$ there exists a v_iv_j-path in $\mathcal{G}$. A digraph $\mathcal{D} = (V, A)$

is *strongly connected* if for all pairs of vertices $v_i, v_j \in V$ $(i \neq j)$ there exists a directed v_iv_j-path and a directed v_jv_i-path in $\mathcal{D}$. A maximal (strongly) connected subgraph is called a *(strongly) connected component* of a (di-)graph.

Trees An undirected simple graph $\mathcal{G} = (V, E)$ is a *forest* if it contains no cycles. A connected forest is a *tree*, thus all connected components of a forest are trees. The vertices of a tree with degree 1 are called *leaves*.

3.2 Network Flow

Most transportation problems cannot be modeled sufficiently by using graphs only. There is one essential feature missing: weights associated with arcs or nodes, which allow to model capacities, costs, demand and supply. Therefore, we will extend the notion of graphs to networks and discuss the concept of flows in these networks.

Network A *network* associates a digraph $\mathcal{D} = (V, A)$ with the following functions: a *capacity* function $u : A \to \mathbb{R}_{\geq 0}$, a *cost* function $c : A \to \mathbb{R}_{\geq 0}$, a *fixed-costs* function $\tilde{c} : A \to \mathbb{R}_{\geq 0}$ and a *balance* function $b : V \to \mathbb{R}$ with

$$\sum_{v \in V} b(v) = 0. \tag{3.1}$$

We will denote a network by $\mathcal{N} = (V, A, u, c, \tilde{c}, b)$. Furthermore, for a given arc a we name $c(a)$ the (per-unit) costs of that arc, $\tilde{c}(a)$ the arcs fixed-costs and $u(a)$ its capacity. Similar definitions apply to an undirected graph $\mathcal{G} = (V, E)$. Figure 3.2 visualizes an example network.

Flow A *flow* in a network $\mathcal{N} = (V, A, u, c, \tilde{c}, b)$ is a function $x : A \to \mathbb{R}_{\geq 0}$ on the arc set of $\mathcal{N}$. For convenience, we will use the notion x_a for the value $x(a)$. If $x_a \in \mathbb{Z}_{\geq 0}$ holds for all $a \in A$ the flow is called an *integer* or *integral flow*. For a given flow x in $\mathcal{N}$ the *excess* of node $v \in V$ is given by the function $\mathrm{ex}_x : V \to \mathbb{R}$ with

$$\mathrm{ex}_x(v) \coloneqq \sum_{a \in \delta^-(v)} x_a - \sum_{a \in \delta^+(v)} x_a. \tag{3.2}$$

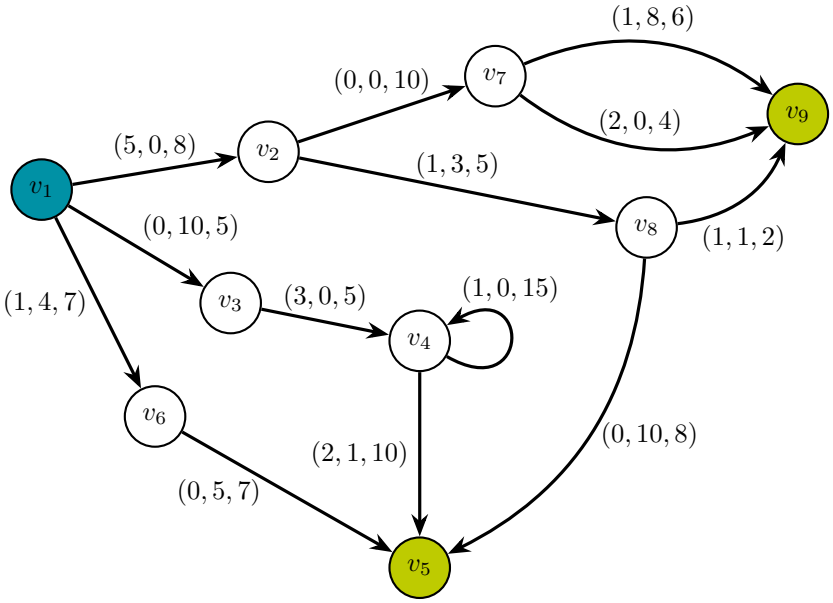


Figure 3.2: Visualization of a network combining a directed graph with weights. While arc weights are presented as tuples $(c, \tilde{c}, u)$, nodes are highlighted in blue ($b(v) < 0$) or green ($b(v) > 0$) to give an indication of their balance function value.

For given non-empty subsets $S, T \subset V$ of vertices the flow x is called an *S-T-flow* if

$$\mathrm{ex}_x(v) \begin{cases} < 0 & \forall v \in S \\ > 0 & \forall v \in T \\ = 0 & \forall v \in V \setminus (S \cup T). \end{cases} \tag{3.3}$$

The nodes of set S are called *sources*, the nodes of set T are called *sinks*. If $S = \{ s \}$ and $T = \{ t \}$ for some nodes $s, t \in V$, we call x an *s-t-flow*. A flow x in $\mathcal{N}$ is *feasible* if the following two conditions hold:

$$0 \leq x_a \leq u(a) \qquad \forall a \in A \tag{3.4}$$

$$\mathrm{ex}_x(v) = b(v) \qquad \forall v \in V. \tag{3.5}$$

Multi-Commodity Flow The previously presented definitions of network and flow are sufficient to model various real-life problems for transporting a single type of goods through a transportation infrastructure, like water in pipes, electricity in cables or data in the internet. However, if there are several different types of

goods that share the same infrastructure some extensions to the above definitions are required. To this end, let K be a finite, non-empty set of commodities. A *multi-commodity network* is a network $\mathcal{N} = (V, A, u, c, \tilde{c}, b)$ with a *cost* function $c : A \times K \to \mathbb{R}_{\geq 0}$ and a *balance* function $b : V \times K \to \mathbb{R}$ with

$$\sum_{v \in V} b(v, k) = 0 \qquad \forall k \in K. \tag{3.6}$$

Similarly, a *multi-commodity flow* in $\mathcal{N}$ is defined as function $x : A \times K \to \mathbb{R}_{\geq 0}$. We will use the notion x_a^k for the value $x(a, k)$. The *excess* of node $v \in V$ and commodity $k \in K$ is given by the function $\mathrm{ex}_x : V \times K \to \mathbb{R}$ with

$$\mathrm{ex}_x(v, k) \coloneqq \sum_{a \in \delta^-(v)} x_a^k - \sum_{a \in \delta^+(v)} x_a^k. \tag{3.7}$$

Finally, the feasibility conditions for multi-commodity flow x must be changed to

$$0 \leq \sum_{k \in K} x_a^k \leq u(a) \qquad \forall a \in A \tag{3.8}$$

$$\mathrm{ex}_x(v, k) = b(v, k) \qquad \forall v \in V, k \in K. \tag{3.9}$$

3.3 Linear and Integer Programming

Optimizing a linear objective function over a polyhedron is one of the most important sub-fields of mathematical optimization. In the following we will give a basic introduction to that field, as it is the basis for most of the solution methods presented in this thesis. Furthermore, we will clarify the notation that is used in the remainder of this work (for more details see Schrijver 1998; Schrijver 2003).

Convexity, Halfspaces and Extreme Points A set $C \subseteq \mathbb{R}^n$ is *convex* if $\lambda x + (1 - \lambda)y \in C$ for all $x, y \in C$ and any $\lambda \in [0, 1]$. The *convex hull* of a set $X \subseteq \mathbb{R}^n$ is the smallest convex set containing X and is defined by

$$\mathrm{conv}(X) \coloneqq \{ \sum_{i=1}^{k} \lambda_i x_i \mid x_i \in X, \lambda_i \in \mathbb{R}_{\geq 0}, \sum_{i=1}^{k} \lambda_i = 1 \}. \tag{3.10}$$

The sum $\sum_{i=1}^{k} \lambda_i x_i$ is called a *convex combination* of $x_1, \ldots, x_k$. An element $x \in X$ with $x \notin \mathrm{conv}(X \setminus \{x\})$ is an *extreme point* of X.

A set $H \subseteq \mathbb{R}^n$ is called an *affine halfspace* if for a given vector $c \in \mathbb{R}^n, c \neq 0$ and some $\delta \in \mathbb{R}$ the set is defined as

$$H \coloneqq \{\, x \in \mathbb{R}^n \mid c^T x \leq \delta \,\}. \tag{3.11}$$

If $c^T x = \delta$ holds the set is called an *affine hyperplane* instead.

Polyhedra and Polytops Given some matrix $\mathcal{A} \in \mathbb{R}^{m \times n}$ and some vector $b \in \mathbb{R}^m$ the convex set $P = \{\, x \in \mathbb{R}^n \mid \mathcal{A}x \leq b \,\}$ is called a *polyhedron*. It is the intersection of finitely many affine halfspaces. A bounded polyhedron is also called a *polytope* and can be defined as the convex hull of the finite set of the polyhedron's extreme points.

Faces and Facets Let $P = \{\, x \in \mathbb{R}^n \mid \mathcal{A}x \leq b \,\}$ be a polyhedron. Given a vector $c \in \mathbb{R}^n, c \neq 0$ and finite $\delta = \max\left\{ c^T x \,\middle|\, x \in P \right\}$ the affine hyperplane $H = \left\{ x \in \mathbb{R}^n \,\middle|\, c^T x = \delta \right\}$ is called a *supporting hyperplane* of P. A set $F \subseteq P$ is a *face* of P if $F = P$ or if F is the intersection of P with one of its supporting hyperplanes. Given an extreme point $x \in P$ the set $\{\, x \,\}$ always is a face of P. A *facet* of P is a maximal face $F \subseteq P$ with $F \neq P$ and $\dim(F) = n - 1$.

Linear and Integer Programming Common techniques to model and solve optimization problems are linear and integer programming. The first can be applied when minimizing or maximizing a linear objective function f over a polyhedron X. More formally, given a matrix $\mathcal{A} \in \mathbb{R}^{m \times n}$ and vectors $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$, a *linear program (LP)* can be described in one of the canonical forms

$$\begin{aligned} \min_x f(x) &= c^T x \\ \text{s.t. } \mathcal{A}x &\geq b \\ x &\in \mathbb{R}^n_{\geq 0} \end{aligned} \tag{3.12}$$

$$\begin{aligned} \max_x f(x) &= c^T x \\ \text{s.t. } \mathcal{A}x &\leq b \\ x &\in \mathbb{R}^n_{\geq 0} \end{aligned} \tag{3.13}$$

with polyhedron $X = \left\{ x \in \mathbb{R}^n_{\geq 0} \,\middle|\, \mathcal{A}x \leq b \right\}$. The elements $x \in X$ are called *feasible solutions* of the LP. A feasible solution attaining the maximum (or minimum) is called an *optimal solution*. There are two possibilities when an LP has no optimal solution at all. It can be *infeasible* ($X = \emptyset$) or *unbounded* (e.g., for all $\delta \in \mathbb{R}$ there

exists $x \in X$ with $c^T x > \delta$ in case of maximization). Otherwise it has (at least) one optimal solution $\overset{*}{x} \in X$.

Limiting optimization to the set of integer feasible solutions leads to the definition of an *integer program (IP)*.

$$\begin{aligned} \min_x f(x) &= c^T x && (3.14) \\ \text{s.t. } \mathcal{A}x &\geq b \\ x &\in \mathbb{Z}^n_{\geq 0} \end{aligned} \qquad \begin{aligned} \max_x f(x) &= c^T x && (3.15) \\ \text{s.t. } \mathcal{A}x &\leq b \\ x &\in \mathbb{Z}^n_{\geq 0} \end{aligned}$$

With X given as above we define the polyhedron $X_I = \text{conv}\Big(\Big\{ x \in \mathbb{Z}^n_{\geq 0} \;\Big|\; \mathcal{A}x \leq b \Big\}\Big)$ as the *integer hull* of X. Obviously $X_I \subseteq X$. If X is bounded, then X_I is also a polytope. Relaxing the integrality constraints of an IP leads to a linear program that is called the *LP-relaxation* of the IP. If only some components of vector $x \in X$ are integral, we call the resulting program a *mixed-integer program (MIP)*.

Simplex and Interior Point Method The first and best-known algorithm for solving linear programs is the *simplex method* designed by Dantzig (1951). Although a non-polynomial worst-case running time bound has been proven, it is quite efficient on most practical problems and widely used. The algorithm relies on the fact, that an optimal solution of a linear program exists at an extreme point of the corresponding polyhedron (as long as there exists an optimal solution at all). Starting with an arbitrary feasible solution, the algorithm moves from one extreme point to the other, improving the solution value in each iteration (or more exact, not worsening the solution value, as it might not improve for some iterations). It terminates as soon as the solution value cannot be improved further by moving to another extreme point.

An alternative to the simplex method is the *interior point method* presented by Karmarkar (1984). Instead of moving along the polyhedron's extreme points the algorithm moves across the feasible region. Its running time is polynomial and there exist efficient implementations that can compete with the simplex method.

Column Generation Many linear programs are too large to consider all of their variables explicitly within the simplex algorithm. However, looking at an optimal solution of such a program, the majority of variables will usually have a value of

zero and can be ignored. Thus, in theory only a small subset of all variables is required to find an optimal solution. When using *column generation* the original problem is split into a *(reduced) master problem* and a *sub problem*. The first contains only a subset of all variables and is intended to find the optimal solution for that subset. The latter is used to identify further variables that need to be added to the master problem. Adding variables is also called pricing. For that, the sub problem sometimes is called *pricing problem*. The algorithm terminates if no further variables can be found and the solution to the master problem will be a solution to the original problem. (Desrosiers and Lübbecke 2005)

Branch-and-Bound Simplex method and column generation can only be used to solve linear programs (except for some special cases). The most common algorithm for solving integer or mixed-integer programs is the *branch-and-bound* algorithm. The rough idea of this algorithm can be described as follows (for more details see Schrijver (2003, pp. 982–984) or Korte and Vygen (2012, pp. 553–556)):

Given is a MIP (or IP) and its LP-relaxation:

1. Solve the LP-relaxation to optimality.
2. If all integral variables of the MIP have integral values in the solution of the LP-relaxation then terminate. The optimal solution for the LP-relaxation is an optimal solution for the MIP.
3. Select any integral variable x that has a fractional value $\overset{*}{x}$ in the solution of the LP-relaxation and branch on this variable:
   a) Create a copy of the MIP and add a constraint forcing $x \leq \lfloor \overset{*}{x} \rfloor$
   b) Create a copy of the MIP and add a constraint forcing $x \geq \lceil \overset{*}{x} \rceil$
   c) Solve both sub-MIPs using branch-and-bound
4. Branching again and again will create a tree of problems. At some point this will lead to a feasible solution to the original MIP in one of the leaves of the tree. This solution may not be optimal. However, the objective value of this solution can be used as a bound for the optimal objective value.

5. After solving the LP-relaxation in any node of the tree, compare the objective value of this LP solution against the objective value of the best known feasible solution. If the LP solution is worse the entire sub-tree starting at this node will never lead to a better feasible solution and can be discarded. This procedure is called bounding.

Integrating column generation into branch-and-bound for solving the LP-relaxations is usually called *branch-and-price*.

Cutting Plane Method An alternative approach to branch-and-bound for solving integer programs has first been proposed by Dantzig, Fulkerson, and Johnson (1954) and Gomory (1958). Considering an arbitrary polyhedron X and its integer hull X_I, obviously solving an integer program on X could be reduced to solving an equivalent linear program on X_I since all extreme points of X_I are integral by definition. However, X_I is not known a priori. In the *cutting plane method* the polyhedron X is intersected with an affine halfspace to cut off certain parts of X and obtain the polyhedron X' with $X_I \subseteq X' \subseteq X$. If solving an equivalent linear program on X' yields an integer solution that will be the optimal solution to the integer program. Otherwise, the cutting-off procedure can be repeated to obtain X'' and so on.

The algorithm in its original form suffered from non-competitive running times and has little practical relevance today. However, the general idea of cutting planes is used very often in combination with branch-and-bound. Instead of just solving the LP-relaxation, at first several cutting planes are separated with the intention to close the gap between the solution value of the LP-relaxation and the optimal solution value. Even if the optimal solution is not found by solving the LP-relaxation, its solution value might help to discard additional sub-trees during bounding and thus reduce the overall size of the branch-and-bound tree. Although the separation of cutting planes and the solving of the enlarged LP-relaxation is more time-consuming, in many applications the overall branch-and-bound procedure could be sped up considerably. Combining the separation of cutting planes with branch-and-bound or branch-and-price is known as *branch-and-cut* and *branch-and-price-and-cut*.

4 Literature review

The supply chain of automobile manufacturers has attracted the attention of many researchers in the past decades. While they focused mainly on inbound logistics at the beginning, in recent years outbound logistics have gained more momentum. After our problem domain and the broader context of finished vehicle logistics have been outlined in previous chapters, in the following we will provide an overview of the preceding research related to that topics. Please note that we will try to give a rough idea of different modeling and solution approaches here. Nevertheless, there are certain topics that might require an in-depth survey of the existing literature. For readability reasons, we postpone such surveys to the point where that knowledge is required to follow our explanations.

4.1 Finished Vehicle Logistics

Miller, Wise, and Clair (1996) were one of the first that compared alternative approaches for outbound logistics of a North American automobile manufacturer. Their focus has been a discussion of network structures and suggestion of rail terminal locations, since rail is the dominant transport mode in North America. The same market has gained the attention of Ekşioğlu et al. (2010), although they were mainly interested in deciding on the location of assembly plants. Nieuwenhuis, Beresford, and Choi (2012) considered a similar problem, but examined the impact of plant locations on CO_2 emissions. They compared the transports from a manufacturer's domestic assembly plants to an alternative approach that built up plants in the destination market. Eskigun et al. (2005) optimized the location and size of distribution compounds, while incorporating transport and waiting times into the model. The impact of these times on the demand of the dealer has been examined by Lin (2014). Assuming all plant and compound locations as given, but deciding on the used means of transport between these locations has been studied by Çakır (2009), Jin et al. (2010), and Zeng, Hu, and Huang (2013). A

two-stage procedure that first clusters the dealers before assigning these clusters to distribution compounds has been proposed by Boujelben, Gicquel, and Minoux (2012). Finally, Hei et al. (2014) further assumed that there is a weekly schedule frequency for using train or vessel transports.

Besides publications concerning the strategic design of the transport network, there have been several discussions regarding the organization of outbound logistics. Typically, there is a distinction between a build-to-forecast and build-to-order strategy. As these publications usually are non-mathematical and only indirectly touch our problem domain, we refer to the excellent introductory papers of Gunasekaran and Ngai (2005), Suthikarnnarunai (2008), and Chandra, Ghosh, and Srivastava (2016) and will not review that area in more detail.

Since compounds constitute an important part of the outbound supply chain, it is no surprise that processes within such compounds had been the scope of research in the past. Mattfeld and Kopfer (2003) and Mattfeld (2006) gave a broad introduction to that topic and suggested a decision model for simultaneous workforce planning and inventory control. Tracing the position of vehicles within a compound using RFID has been examined by Kim et al. (2010). And an integrated planning approach for vessel berth allocation and the assignment of parking lots to vehicles has been proposed by Dorndorf and Kneis (2014). Although these publications help in understanding the processes carried out in a compound, they are again not directly related to our problem domain.

The same applies to literature contributions that focus on the delivery from the final distribution compound to the dealerships. Here, typically we see operational models that are based on the vehicle routing problem and support LSPs in building tours like in Cordeau et al. (2015) and Hu et al. (2015). Or they optimize load patterns for auto-carrier, which are packing problems similar to the ones proposed by Agbegha, Ballou, and Mathur (1998).

We found only a single paper that is directly concerned with distribution planning in finished vehicle logistics. Jin, Luo, and Ekşioğlu (2008) proposed an integrated approach of production sequencing and outbound logistics. Their focus has been answering the questions, when a vehicle should be assembled in the plant and when it will be delivered to the dealership. However, the authors did not incorporated transshipment locations and additional services into their model, which is an important requirement in our domain. Furthermore, an interesting survey has been

published by Haubrich (2017), who gives an overview of different optimization problems that occur within an automotive outbound supply chain and how these problems can be tackled within a decision support system.

Although, most of above literature contributions provided some insights into different aspects of finished vehicle logistics, none of them proposed a model that is able to handle all requirements presented in Chapter 2. Thus, we extended our literature review to modeling and solution approaches that have been suggested for other applications than automotive logistics.

4.2 General Freight Logistics

There are mainly three modeling approaches that dominate many mathematical publications on freight logistics. Typically, the problems occurring in that field are formulated as facility location, network design or vehicle routing problems. While the latter usually is applied in an operational context, the first two are mainly used when discussing strategic or tactical questions. We will focus on network design formulations in the following, as we do not see any use for the others in the context of our problem domain. Nevertheless, the interested reader is referred to the comprehensive surveys of Melo, Nickel, and Saldanha-da-Gama (2009) and Toth and Vigo (2014).

Since optimization in transport logistics has been subject of research for several decades now, there exists a multitude of survey papers summarizing the contributions in that area. While Cordeau, Toth, and Vigo (1998) studied approaches for modeling train transports, Christiansen et al. (2007) focused on maritime transportation. Additionally, there are publications concerned with models for freight transportation in general (Crainic and Laporte 1997; Crainic 2003; Crainic and Semet 2013; Bektaş 2017) or with so called *multi-modal* transportation, that sequentially uses different means of transport (Crainic and Kim 2007; SteadieSeifi et al. 2014). Although all of these surveys have a different thematic focus or summarize the contributions of different decades, they show a similar internal structure by classifying problems by their planning horizon into strategical, tactical or operational.

Network design and network flow models are typical representatives for strategical or tactical planning approaches. While for the latter a fixed network is given and

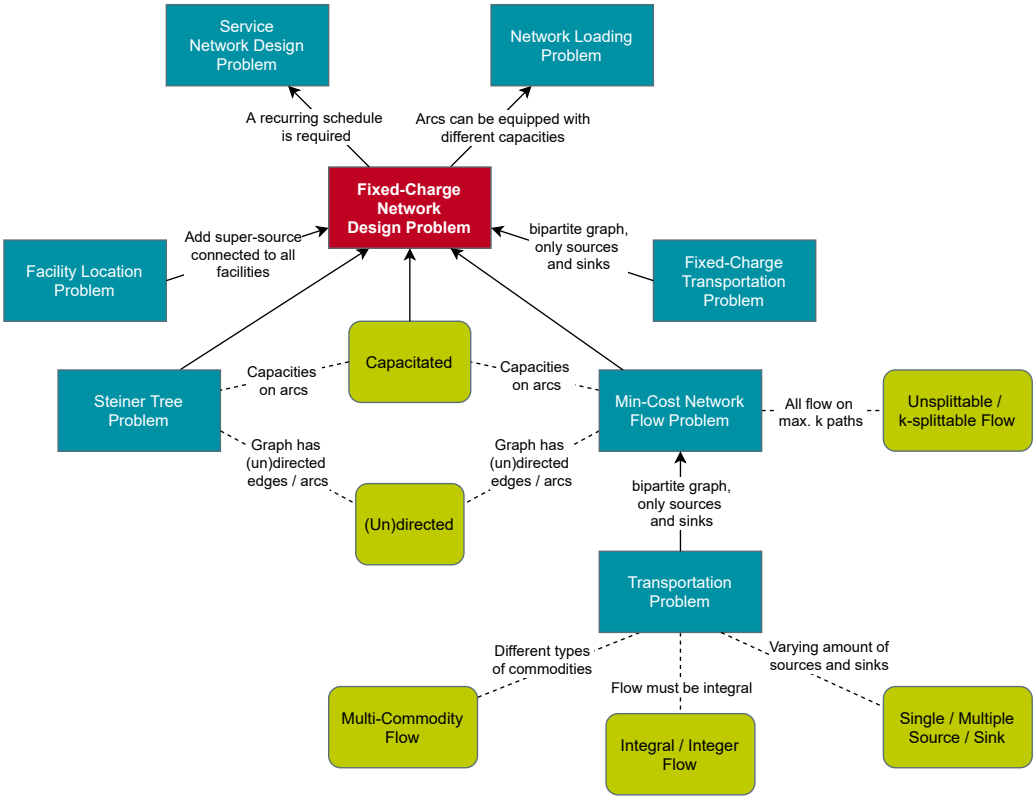


Figure 4.1: Overview of different problem models that are closely related to the Fixed-Charge Network Design Problem. While the blue boxes represent problem models, the green ones illustrate variations of these models, which are further explained by the connecting dashed lines. The arrows symbolize a "is generalized by" relation.

the flow of commodities through that network is subject of optimization, the first decides simultaneously, which parts of the network should be used. Figure 4.1 gives an overview of different problem models in that area and how they are connected to each other. The survey from Magnanti and Wong (1984) is a good introduction to multiple of these models. In our view the FND is of outstanding relevance for this literature review, since it is connected to a multitude of related problem classes and has been used in numerous applications. Hence, we decided to start our review there.

Fixed-Charge Network Design Geoffrion and Graves (1974) have been the first to discuss a problem that can be classified as a multi-commodity capacitated fixed-charge network design problem (MCFND). Although they did not incorporate arc

capacities in their model, they had to consider limitations on the minimum and maximum flow through intermediate nodes. They used Benders decomposition to solve their problem. The same solution method has been successful for other authors in the following decades (Costa 2005; Costa, Cordeau, and Gendron 2009; Gendron 2011; Costa et al. 2012).

Another classical approach to tackle FND is branch-and-cut. Padberg, van Roy, and Wolsey (1985) were among the first to propose valid inequalities and separation algorithms for that class of problems. The multi-commodity case has been examined by Gendron, Crainic, and Frangioni (1999) using different types of relaxations and there are many more that contributed to the set of valid inequalities that we know today (Atamtürk 2001; Chouman, Crainic, and Gendron 2003; Atamtürk and Günlük 2007; Chouman, Crainic, and Gendron 2009; Costa, Cordeau, and Gendron 2009; Gendron 2011; Letchford and Souli 2019). In contrast, the work of Rardin and Wolsey (1993) and Ortega and Wolsey (2003) especially focused on the uncapacitated FND. We will look into all that inequalities in more detail in Section 7.4.1. Recently, these branch-and-cut approaches have been further improved by using techniques like filtering and variable fixing (Chouman, Crainic, and Gendron 2018) or by choosing the right way of representing commodities (Chouman, Crainic, and Gendron 2016). For the uncapacitated case, a rather old method introduced by Lamar, Sheffi, and Powell (1987) can be used to further tightening the lower bound.

The first publication that proposed branch-and-price-and-cut to solve MCFND goes back to Gong (1996). In the past decade, further contributions followed (Frangioni and Gendron 2009; Hewitt, Nemhauser, and Savelsbergh 2012; Gendron and Larose 2014). The case of integral *unsplittable* flow has been worked on by Barnhart, Hane, and Vance (2000). Nevertheless, even more popular than branch-and-price-and-cut have been approaches based on Lagrangian relaxation for determining the bounds within a branch-and-bound scheme. Typically, these Lagrangian bounds are calculated using the sub-gradient (Holmberg and Yuan 2000; Crainic, Frangioni, and Gendron 2001; Sellmann, Kliewer, and Stein 2002; Bektaş, Chouman, and Crainic 2010) or bundle method (Crainic, Frangioni, and Gendron 2001; Kliewer and Timajev 2005). Gendron (2011) compared these methods with implementations of branch-and-cut and Benders decomposition and has been convinced of the superiority of the bundle method. Finally, there have been numerous heuristic approaches published in the literature, like the ones from

Powell (1986), Ekşioğlu, Pardalos, and Romeijn (2002), and Crainic, Gendron, and Hernu (2004). Since we will look into these heuristics in more detail, we postpone the in-depth review to Section 12.1.

Besides the standard formulations of FND and MCFND that have been subject of most above publications, there also exist more unusual variations. Incorporating robustness requirements like uncertain demand (Raack 2012) or activating an arc only once for the remainder of the planning horizon (Fragkos, Cordeau, and Jans 2017) are only two of them. We refer to Bektaş (2017) for a comprehensive survey and further applications.

Service Network Design Determining a schedule for services, which are offered within transportation networks, that typically have a focus on consolidation operations (e.g., in rail yards or sea ports), is the main application of service network design problems (SNDs). Usually, the used formulations in that area are slight variations of MCFND. Crainic (2000) gives a detailed introduction to the relation of these problem classes. The surveys of Wieberneit (2008a) and Wieberneit (2008b) show that most of the models and solution methods proposed for FND are applicable to SND, too.

Farvolden and Powell (1994) were one of the first to solve a SND by using a heuristic sub-gradient method on a time-expanded network. Such time-expansion has been explained in detail by Kim (1997) and Kim and Barnhart (1999), when they introduced branch-and-price-and-cut to that problem class. The same solution method has been used by Andersen et al. (2010), who examined a typical extension that explicitly handles assets (like means of transport) within the problem model. Andersen, Crainic, and Christiansen (2009a) compared different formulations for that extended model, that have been used by other authors, too (Andersen, Crainic, and Christiansen 2009b; Teypaz, Schrenk, and Cung 2010; Crainic et al. 2014). Besides branch-and-price-and-cut, other methods like tabu search (Pedersen, Crainic, and Madsen 2008) or math-heuristics (Vu, Crainic, and Toulouse 2013) have also been applied in that context.

In addition to the classical applications in freight rail transportation (Zhu, Crainic, and Gendreau 2014) or less-than-truckload freight operations (Jarrah, Johnson, and Neubert 2009), there exist a few exceptional variations of SND. Matuschke (2013) incorporated the selection of tariffs and complex cost structures into his

model and solves it with branch-and-cut and local search heuristics. Dall'Orto et al. (2006) proposed a stochastic version and Rothenbächer, Drexl, and Irnich (2016) integrated questions about facility locations while solving that problem with branch-and-price-and-cut.

Since the time-expanded networks that are typically used in the context of SND are large and entail many problem instances to be intractable, recently there have been a few contributions that try to avoid the time-expansion and instead focus on a continuous time model. Boland et al. (2017) have been among the first in that area and proposed a method that iteratively produces a time-expanded network, however only as much as is needed to find an optimal solution. Furthermore, this method does not rely on a discretization of the planning horizon, but is able to handle a continuous representation of time. Marshall et al. (2021) improved that method and showed that they are able to produce even smaller models that lead faster to high-quality solutions. It will be interesting how this novel modeling approaches will influence future publications related to SND.

Network Loading The primary characteristic of the network loading problem (NLP) that distinguishes it from the FND is, that arcs are not just activated (e.g., with a binary variable), but instead there are different types of transport capabilities installed on the arcs. Typically, per-unit costs are not of interest in such a setting and are ignored. Magnanti, Mirchandani, and Vachani (1991) introduced this problem class, proposed a branch-and-cut procedure and presented the first valid inequalities and separation algorithms. Magnanti, Mirchandani, and Vachani (1993) and Magnanti, Mirchandani, and Vachani (1995) expanded on their work and there are numerous authors that followed and suggested further cuts (Bienstock and Günlük 1996; Bienstock et al. 1998; Günlük 1999; Agarwal 2006). An approximation algorithm has been presented by Krishnan (1998).

Besides the usual formulation of NLP that expects the flow to be splittable, the unsplittable case has gained considerable attention. Both models haven been examined by Barahona (1996) and Atamtürk and Rajan (2002), who proposed a branch-and-cut solution approach. Furthermore, Chabrier (2003) applied a heuristic branch-and-price-and-cut method tackling a model with unsplittable flow. A comparison of heuristic, branch-and-cut and branch-and-price-and-cut approaches for that same problem class has been carried out by Chabrier et al.

(2004). Finally, Agarwal and Aneja (2017) proposed a branch-and-cut method for an application with undirected graphs.

Min-Cost Network Flow From the modeling approaches reviewed in this chapter, network flow problems are by far the most extensively studied, with over half a century of research. The first surveys on that topic have been published by Ford and Fulkerson (1962), Assad (1978), and Kennington (1978), more recent ones by Ahuja, Magnanti, and Orlin (1993) and Salimifard and Bigharaz (2020). Since it would go beyond the scope of this thesis to summarize all these contributions, we will concentrate on network flow models that minimize costs and deliver multi-commodity flows as a result. Since these types of models can be formulated as LP, typically they are solved efficiently using any of the well-known algorithms like simplex or interior-point method (Castro 2000). Nevertheless, these general purpose algorithms fail when the size of the corresponding LPs grows, which calls for specialized solution methods. Since we expect our problem instances to be fairly large, in the following we focus on these methods.

The most widely used technique to solve large network flow problems is column generation. Jones et al. (1993) have been one of the first to apply that method. They examined on the performance impact of different kinds of formulations that occur, when choosing the aggregation level of the commodities (single / multiple source / sink, see Section 5.7). They explained in detail how that aggregation influences the type of the pricing sub-problem that needs to be solved to be either a shortest path, shortest path tree or single commodity network flow problem. An alternative formulation has been suggested by Barnhart et al. (1994). Instead of generating a column for each path, they limited the generation to paths that deviate from a previously defined "key path". Thus, they were able to create considerable smaller LPs, which required less computational effort. Holmberg and Yuan (2003) stated further requirements on the paths of the commodities and hence solved a specialized pricing sub-problem. Besides, there exist further contributions on the usage of column generation (Farvolden, Powell, and Lustig 1993; Mamer and McBride 2000).

Additionally, there are other solution techniques for solving the multi-commodity network flow problem (MCF). Larsson and Yuan (2004) and Babonneau, du Merle, and Vial (2006) present algorithms based on Lagrangian relaxation. Approximation schemes have been proposed by Leighton et al. (1991) and Awerbuch and Leighton

(1994). And finally, Barnhart and Sheffi (1993) contributed a network-based primal-dual heuristic approach.

Besides that multitude of solution methods, there exist numerous modeling variants. Of widespread interest has been the case of integral or binary (unsplittable) flow. The first has been tackled by Alvelos (2005) and the latter by Barnhart, Hane, and Vance (1996), using branch-and-price. In contrast, Aggarwal, Oblak, and Vemuganti (1995) suggested a heuristic approach that first distributes the available arc capacities to the commodities and subsequently solves single commodity flow problems. Other variations are non-linear convex (Ouorou, Mahey, and Vial 2000) or piecewise linear objective functions (Balakrishnan and Graves 1985; Amiri and Pirkul 1997; Croxton, Gendron, and Magnanti 2003b; Croxton, Gendron, and Magnanti 2003a; Muriel and Munshi 2004; Chang 2008; Gendron and Gouveia 2016). The latter are typically modeled as FND and will be of utmost importance for our modeling approach of transport costs. Hoppe and Tardos (2000) looked at evacuation scenarios and hence were not interested in minimizing costs, but time to destination. Sharing the capacity between arcs has been examined by Wollmer (1971). And similarly to SND, there have been several contributions concerning continuous time models (Fleischer and Tardos 1998; Fleischer and Skutella 2007; Hall, Hippler, and Skutella 2007; Skutella 2009).

From an application perspective, freight transportation has been always one of the most important use cases of network flow models (Ford and Fulkerson 1962; Bookbinder and Sethi 1980). However, today there exists a wide range of other applications in literature. Haghani and Oh (1996) use network flows for disaster relief scenarios. The optimization of traffic networks has been conducted by Köhler, Möhring, and Skutella (2009). In communication they can be used to determine a highly efficient routing within optical (Peinhardt 2003; Ozdaglar and Bertsekas 2004) or multi-hop wireless networks (Kolar and Abu-Ghazaleh 2006). Since these publications typically do not provide alternative modeling or solution approaches, but map their problem to an existing model, we will not go into more detail here.

Integrated planning models Although we already mentioned several extensions or variations of the "standard" models above, we want to note that there exists a considerable amount of literature that tries to go one step further and combine several models into an integrated planning approach. For example, Cordeau, Pasin, and Solomon (2006) join location and capacity choices for plants and warehouses with supplier and transportation mode selection, product range assignment and product flows into a single model. The management of inventory, location of facilities and the determination of transportation policy has been worked on by Jayaraman (1998). And finally, Pan and Nagi (2013) propose a Lagrangian heuristic to solve a model spanning production, inventory, and transportation. Even though these models are powerful and might be interesting for finished vehicle logistics in general, we do not see a direct application within our context. Hence, we just mentioned these examples for the sake of completeness.

Evaluation This review of existing contributions to the area of freight logistics shows the diversity and strength of using network design or network flow models. Unfortunately, we did not find any model that is able to handle all of our requirements. Nevertheless, there have been several papers that shed some light on aspects of our problem domain, like the modeling of complex cost structures, incorporating the dimension of time or the consideration of additional requirements on a commodity's path. In regard to solution techniques, there has been an equal amount of contributions. The ones that focus on solving excessively large problem instances gained most of our attention. In the remainder of this work, we will base our decisions on what we have learned from above papers. If it supports understanding, we will summarize the main contributions of other researches in more detail right before we reuse their results.

5 Model: Flow in Time-Space Service Network

In this chapter a mathematical model of the previously described problem of delivering finished vehicles from their plant to the dealer or customer will be presented. First we discuss how the real-world logistics infrastructure can be transformed into a network based on a pseudo-graph (named a time-space service network, see Crainic and Laporte 1997) and that the finished vehicles can be represented by a (constrained) multi-commodity flow in that network. Second, we will use that model to formulate a mixed-integer program.

5.1 Service Network

Considering the requirements presented in Chapter 2 we will start to model the logistics infrastructure. Ignoring the dimension of time for now will considerably simplify this first step. Given is a directed pseudo-graph $\mathcal{D} = (V, A)$. The set V contains at least one node for each physical location of the logistics network. For each contract with a transport carrier there will be an arc within set A. If multiple carriers offer transports between two locations that will be considered by parallel arcs. Besides transports, there might be other services like storage or modifications. They will be modeled as arcs within set A, too. Typically, for the non-time-expanded case they are self-loops on the node where the service will be performed.

In some situations it is not sufficient to model a physical location as a single node but instead as a set $V_i \subseteq V$ of nodes. It might be necessary to model handling costs within a compound (see Figure 5.1) or it must be ensured that a service will be used only once before leaving a location (see Figure 5.2). Both cases can be modeled using "logical" nodes that have no counterpart within the physical logistics infrastructure. Furthermore, such cases can be combined to more complex

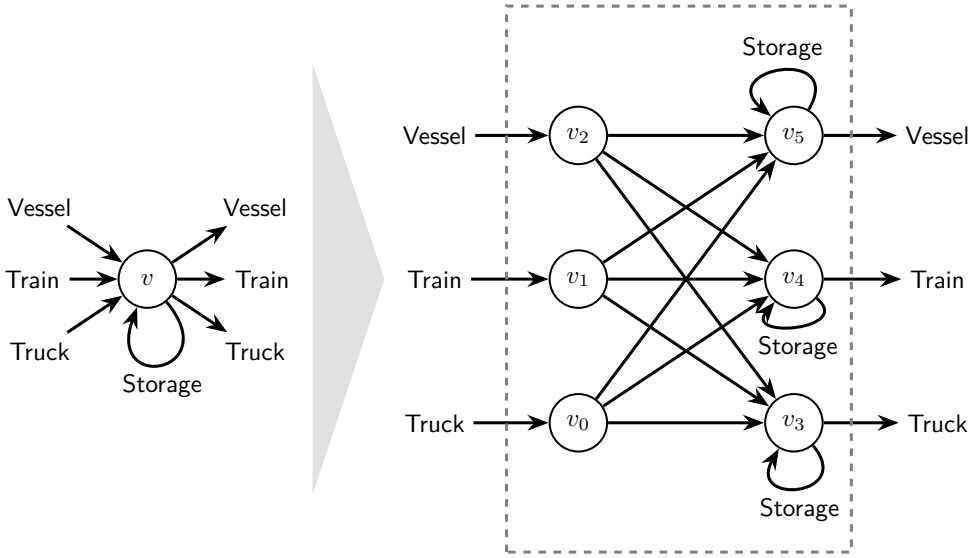


Figure 5.1: Explicitly modeling the compound internal handling can be achieved by using logical nodes.

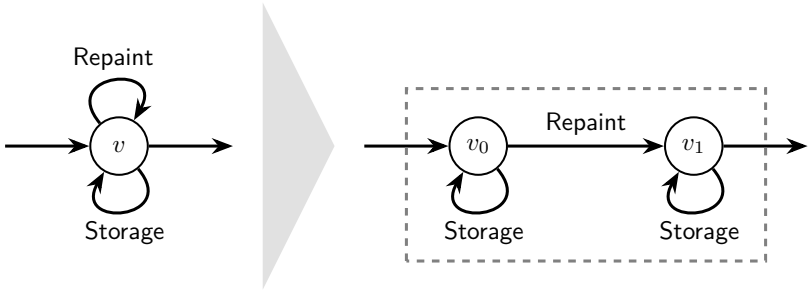
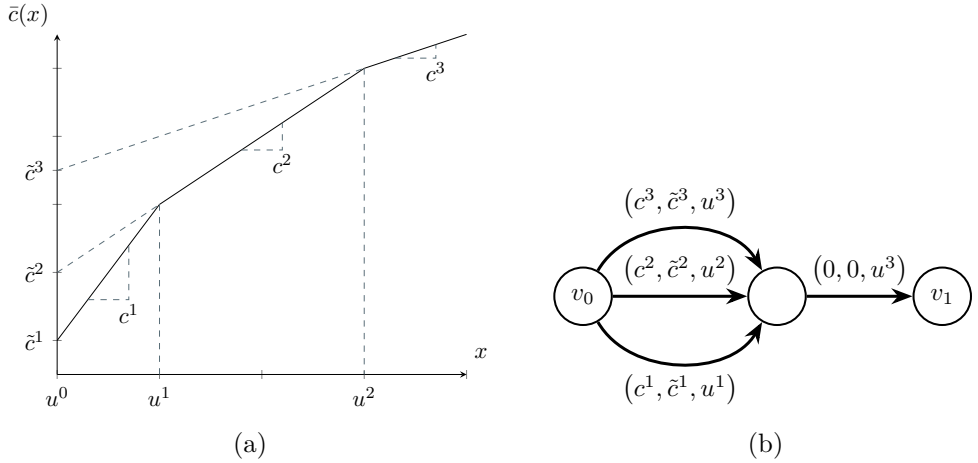


Figure 5.2: Ensuring that a self-loop is used only once by using logical nodes.

compound internal structures. See Guélat, Florian, and Crainic (1990) for a similar modeling approach. Obviously, such a modeling approach is only feasible if the capacities of the storage arcs can be neglected, otherwise their capacity would be multiplied. Alternatively, it would be easily possible to handle that duplication explicitly within our model and later on in the MIP, but we avoid these cases in the following to simplify notation.

Besides using logical nodes to model compound internals, they are also useful when modeling complex tariff structures. Extending our graph $\mathcal{D} = (V, A)$ to a network $\mathcal{N} = (V, A, u, c, \tilde{c}, b)$ there will be a capacity function $u : A \to \mathbb{R}_{\geq 0}$, a costs function $c : A \to \mathbb{R}_{\geq 0}$ and a fixed-costs function $\tilde{c} : A \to \mathbb{R}_{\geq 0}$ defined on each arc $a \in A$ (we ignore the balance function b for now). All requirements of



(a) (b)

Figure 5.3: Example of a piecewise-linear concave cost function (a) and its corresponding sub-graph (b). The arcs are labeled with $(c(a), \tilde{c}(a), u(a))$.

Chapter 2 regarding service costs can be modeled using a piecewise-linear concave cost function $\bar{c}_a : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{\geq 0}$ (see Amiri and Pirkul 1997; Croxton, Gendron, and Magnanti 2003b; Muriel and Munshi 2004). Given is the overall vehicle volume $x_a \in \mathbb{Z}_{\geq 0}$ using arc a. The linear segments of the cost function are enumerated by index set S. Each segment $s \in S$ is determined by a lower bound $u^{s-1} \in \mathbb{Z}_{\geq 0}$ and an upper bound $u^s \in \mathbb{Z}_{\geq 0}$, costs $c^s \in \mathbb{R}_{\geq 0}$ and fixed-costs $\tilde{c}^s \in \mathbb{R}_{\geq 0}$. We expect $u^0 = 0$ and $u^{s-1} < u^s, c^{s-1} \geq c^s \geq 0, \tilde{c}^s = \tilde{c}^{s-1} + (c^{s-1} - c^s)u^{s-1}$ for all $s \in S$. Then the piecewise-linear concave cost function $\bar{c}_a$ is defined as:

$$\bar{c}_a(x) = \begin{cases} \tilde{c}^1 + c^1 x & 0 \leq x \leq u^1 \\ \tilde{c}^2 + c^2 x & u^1 < x \leq u^2 \\ \dots & \\ \tilde{c}^s + c^s x & u^{s-1} < x \end{cases} \tag{5.1}$$

Figure 5.3a shows an example of such a cost function with three segments. Using only the capabilities of network $\mathcal{N}$, this cost function can be modeled by replacing arc $a = (v_0, v_1)$ with the sub-graph shown in Figure 5.3b. The same approach can be used to model several other types of cost functions. See Matuschke (2013) for more examples.

5.2 Time Expansion

The definition of our service network above and the general definition of a network in Section 3.2 do not entail any notion of time. Thus, modeling requirements that are related to time require an approach that is commonly known as *time expansion* of a network into a *time-space network*. This approach has been introduced first by Ford and Fulkerson (1962) and is a widespread technique to handle time requirements in networks today (e.g., see Crainic and Laporte 1997; Crainic 2003; Jarrah, Johnson, and Neubert 2009; Andersen et al. 2010; Crainic et al. 2014).

Given is a discretization of the planning horizon into fixed-size time intervals. These intervals are enumerated by the index set $T = \{ 0, 1, 2, \ldots, T_{end} \}$. We expect the granularity of such an interval to be 1–2 days or a work shift, which turned out to be a good balance between planning accuracy and data availability. Although in the remainder of this thesis the size of the intervals is considered to be homogeneous, recent papers from Boland et al. (2017) and Marshall et al. (2021) suggest to use dynamically sized intervals that may lead to significant smaller time-space networks. However, we will not apply this approach in this thesis but may examine it in future research.

The time expansion of a network $\mathcal{N} = (V, A, u, c, \tilde{c}, b)$ into a time-space network $\mathcal{N}_T = (V_T, A_T, u_T, c_T, \tilde{c}_T, b_T)$ is a simple transformation that duplicates each node for each time-slot of the planning horizon and adjusts the arcs accordingly. Given a *traversal time* function $\tau : A \times T \to \mathbb{Z}_{\geq 0}$, the node set V_T and the arc set A_T can be defined as:

$$V_T = \{ v_{i,t} \mid v_i \in V, t \in T \} \tag{5.2}$$

$$A_T = \{ a_{i,t} = (v_{i,t}, v_{j,t+\tau(a_i,t)}) \mid a_i = (v_i, v_j) \in A, t \in T, t + \tau(a_i, t) \in T \} \tag{5.3}$$

For less complex use cases the capacity function $u_T : A_T \to \mathbb{R}_{\geq 0}$, cost function $c_T : A_T \to \mathbb{R}_{\geq 0}$ and fixed-costs function $\tilde{c_T} : A_T \to \mathbb{R}_{\geq 0}$ can just ignore the dimension of time and thus will be defined as:

$$u_T(a_{i,t}) = u(a_i) \qquad \forall a_i \in A, a_{i,t} \in A_T \tag{5.4}$$

$$c_T(a_{i,t}) = c(a_i) \qquad \forall a_i \in A, a_{i,t} \in A_T \tag{5.5}$$

$$\tilde{c_T}(a_{i,t}) = \tilde{c}(a_i) \qquad \forall a_i \in A, a_{i,t} \in A_T \tag{5.6}$$

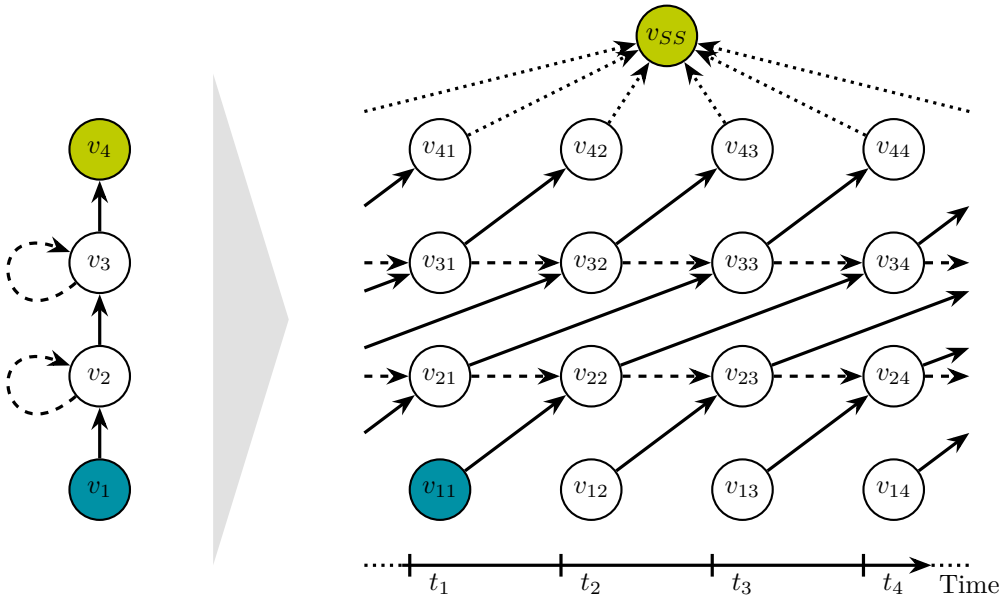


Figure 5.4: Example of time expansion of a network with four (physical) nodes. The colored nodes are source (v_1 / v_{11}), sink (v_4) and super-sink node (v_{SS}).

However, it might be beneficial to define these functions time-dependent to allow variations of capacity or costs over time. For example, a schedule of a vessel or a train can easily be modeled by such a time-dependent capacity function ($u_T(a_{i,t}) = 0$ whenever a service is not available). Hence, in the following we will not limit our time-space networks to the simple form shown above. Please note that a function value $\tau(a, t) = 0$ is allowed explicitly, so traversing an arc might consume no time. This is especially useful when modeling compound internal processes that mostly consume much less time than the predefined size of the planning horizon intervals. However, as we will show in Section 6.1 this definition of τ implies major algorithmic consequences.

The transformation of the balance function $b_T : V_T \to \mathbb{R}$ requires more effort. Just duplicating the balance at each point in time would miss the intention of the balance function. Instead it must be decided for each node $v_i \in V$ at which point in time a certain balance will be available with $\sum_{t \in T} b_T(v_{i,t}) = b(v_i)$. Furthermore, it must be ensured that all flows can reach their sinks from their sources which might not be the case if the volume is starting too late in the time-space network. Finally, the requirements presented in Chapter 2 contained a notion of “lateness” which

cannot be modeled so far. Looking at a time-space network, it is not obvious how a balance function can be defined to allow the flow to reach a sink at an arbitrary point in time. However, a common technique used in that case is the introduction of a super-sink node that is connected to all (time-expanded) sink nodes (compare Gendron and Larose 2014). The arcs from the sinks to the super-sink can further be used to defined lateness costs which will be shown later. Figure 5.4 shows an example of time expansion of a network.

In the following, a network is considered to be always a time-space network (subscript T will be omitted, if there is no ambiguity). However, that has no impact on the correctness of our models or algorithms, but just on the size of the problem instances (which will increase significantly). Keep in mind that all requirements related to time are hidden within the structure of the network and will not be considered explicitly.

5.3 Constrained Multi-Commodity Flow

All finished vehicles that are transported using the logistics infrastructure will be represented as multi-commodity flow within our model. In the physical infrastructure each vehicle will have a start location (e.g., the assembly plant) and a final destination (e.g., a dealership). The date when the vehicle will be ready for shipment from the start location is assumed to be given and in most cases can be estimated sufficiently from the production schedule. In contrast, estimating the date of delivery to the final destination is rather challenging. There might be a TDD given that has been promised to the dealer or the date might be irrelevant in case of a VHC delivery. Both cases can be modeled using the concept of a *lateness costs function* $\hat{c} : K \times T \to \mathbb{R}_{\geq 0}$, with K being the set of commodities and T being the set of discrete time intervals of the planning horizon. As mentioned before, all nodes of the time-space network that represent the final destination of a vehicle at some point in time will be connected to the super-sink node. The arcs that lead to that node will be annotated with lateness costs. These costs may vary over time and hence can be used to model the aforementioned cases. E.g., if a TDD is given the lateness costs will be low on the arc corresponding to that date and will be high otherwise. It would also be possible to have low costs until the TDD to allow earlier deliveries. All these options highly depend on the type of delivery and on

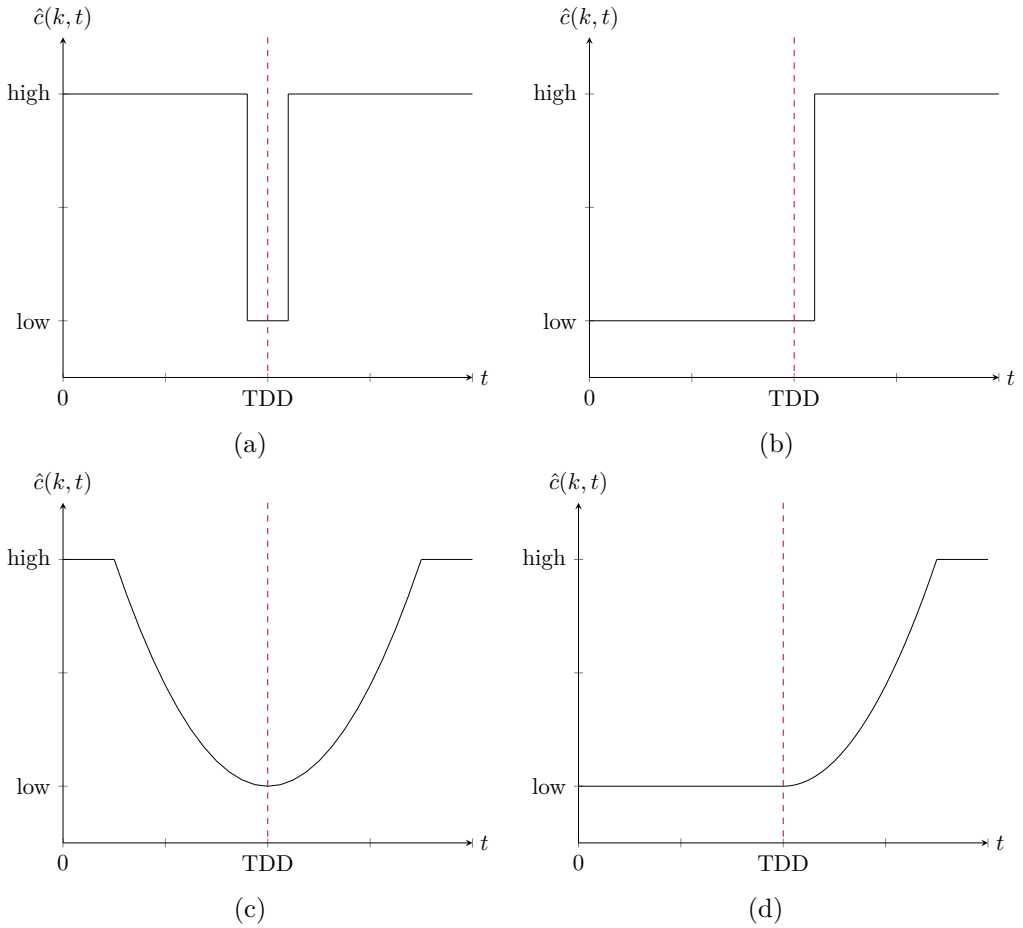


Figure 5.5: Examples of lateness costs functions. Function (a) forces high costs when not delivering close to TDD but (b) allows earlier delivery. Same applies to (c) and (d) but costs will increase slowly instead of instantly.

the terms agreed on with the receiver of the vehicle. Some examples of such a lateness costs function are shown in Figure 5.5.

Given a multi-commodity network $\mathcal{N} = (V, A, u, c, \tilde{c}, b)$, we define a set of commodities K. Each vehicle will be represented by such a commodity and each commodity represents a group of vehicles. As it might be possible to aggregate vehicles with similar logistics requirements (see Section 5.7), a commodity might have a set of origin nodes $V_o^k \subset V$ and a set of destination nodes $V_d^k \subset V$ (the latter typically contains only the super-sink node). The actual volumes of a commodity $k \in K$ that are either supplied at the origin nodes or demanded at the destination nodes

can be defined using the network's balance function $b : V \times K \to \mathbb{Z}$ with

$$b(v,k) \begin{cases} \leq -1 & \forall v \in V_o^k \\ \geq 1 & \forall v \in V_d^k \\ = 0 & \text{otherwise.} \end{cases} \tag{5.7}$$

Note that we slightly changed the common definition of the balance function b presented in Section 3.2 to be integral (a vehicle cannot be divided into smaller chunks). Consequently, we also changed the definition of a multi-commodity flow to be an integral function $x : A \times K \to \mathbb{Z}_{\geq 0}$.

Besides, there exist further requirements on the route a vehicle might take through the logistics infrastructure (see Section 2.2 for details). First, in master planning there might already be a route or a sub-network selected for a vehicle. Hence, for each commodity an arc set $A^k \subseteq A$ must be defined, limiting the arcs that can be used by that commodity. For convenience we define

$$V^k = \left\{ v \in V \;\middle|\; \exists a \in A^k : \mathit{head}(a) = v \vee \mathit{tail}(a) = v \right\} \tag{5.8}$$

as the subset of nodes that are connected by any arc $a \in A^k$. Note that defining $A^k = A$ is explicitly an option here. Second, it might be necessary to perform a certain service for a vehicle that requires to visit one of various VMCs. As there might be multiple such services required for a vehicle and each of these services might be performed at a different location, we model such an *arc-requirement* R as element of the set $\Psi^k = \mathfrak{P}(A^k) \setminus \emptyset$. The set $\Omega^k = \left\{ R_1, R_2, \ldots \;\middle|\; R_i \in \Psi^k \right\}$ defines all arc-requirements for commodity k. The following examples will explain how these requirements need to be interpreted:

$R_1 = \{ a_1 \}$	Arc a_1 must be used
$R_2 = \{ a_1, a_2, a_3 \}$	Use at least one of the arcs a_1, a_2 or a_3

Finally, besides defining arc-requirements for a commodity regardless of its route, there is also a need to force the usage of arcs depending on other arcs that have been used or will be used. For each commodity $k \in K$ there is *a predecessor arcs function* $\phi_P^k : A^k \to \mathfrak{P}(\Psi^k)$ and a *successor arcs function* $\phi_S^k : A^k \to \mathfrak{P}(\Psi^k)$. Before a commodity can flow over arc $a \in A^k$, it must be ensured that this commodity has already fulfilled all *predecessor arc-requirements* $\phi_P^k(a)$. In contrast, it must be assured that all *successor arc-requirements* $\phi_S^k(a)$ are accomplished before arriving

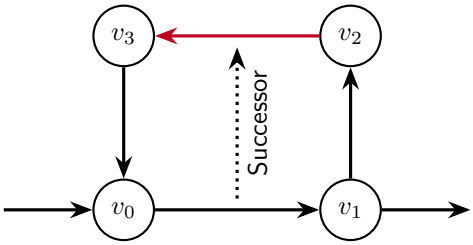


Figure 5.6: Example graph showing a successor relation between two arcs that cannot be resolved.

at the commodities sink. To shorten notation we define $\phi^k(a) = \phi^k_P(a) \cup \phi^k_S(a)$. To be feasible, all arcs must **not** have (1) a precedence to themselves or (2) a precedence that is defined within a cycle of the time-expanded network and cannot be resolved, similar to the example in Figure 5.6. As soon as arc (v_0, v_1) has been used by a path there is no chance to escape from the cycle that is required to fulfill the successor requirement on arc (v_2, v_3).

5.4 Objective

Defining an objective is essential when performing a task like distribution planning. There needs to be a measurement to distinguish a "good" plan from a "bad" one. Relating to the problem model described above, a plan (or a problem solution) is a set of paths $P^k = \left\{ p^k_1, p^k_2, \dots \right\}$ for each commodity $k \in K$ that needs to be derived from multi-commodity flow $x : A \times K \to \mathbb{Z}_{\geq 0}$. So finding this multi-commodity flow x (and ensuring that it can be decomposed into paths) is the first task that needs to be addressed. Of course, we do not want to find an arbitrary flow x but rather a good or even an optimal one. Such a quality measure should consider the requirements presented in Chapter 2, which means that there must be a balance between minimizing overall logistics costs and maximizing customer satisfaction. The first will be taken into account by service costs and fixed-costs and the latter will be represented by lateness costs. Thus, in the context of the model above both aims will be addressed by minimizing the function

$$f(x, y) = \sum_{k \in K} \sum_{a \in A^k} c(a, k) \cdot x(a, k) + \sum_{a \in A} \tilde{c}(a) \cdot y(a) \tag{5.9}$$

given function $y : A \to \{0, 1\}$ with the value $y(a) = 1$ if flow x is using the arc a and $y(a) = 0$ otherwise. Please recall that lateness costs are mapped to costs on the arcs leading to the super-sink and therefore are not handled explicitly in function f.

5.5 Mixed-Integer Program Definition

Since we reduced our problem to minimizing a linear objective function over a system of (integer) linear constraints, the obvious tool to solve that problem model is (mixed) integer programming. In the following, we are going to formulate a MIP using the model described above. Various modeling approaches can be found in the literature that lead to different MIPs handling a multitude of problems. The MCFND comes closest to the problem considered in this thesis (see Gong 1996; Gendron, Crainic, and Frangioni 1999). There are several MIP formulations available for that class of problems, the most important ones are the *arc-flow* and the *path-flow* formulation. We will show how the arc-based formulation can be extended to consider arc-requirements and ensure that the resulting flow will be integral. Following the usual naming scheme from the literature, our problem can be classified as path-constrained integer multi-commodity capacitated fixed-charge network design problem (PCIMCFND). The resulting MIP is defined as follows:

$$\min \sum_{k \in K} \sum_{a \in A^k} c_a^k x_a^k + \sum_{a \in A} \tilde{c}_a y_a \tag{5.10a}$$

$$\text{s.t.} \sum_{a \in \delta^+(v)} x_a^k - \sum_{a \in \delta^-(v)} x_a^k = b_v^k \qquad \forall k \in K, \forall v \in V^k \tag{5.10b}$$

$$\sum_{k \in K : a \in A^k} x_a^k \leq u_a \cdot y_a \qquad \forall a \in A \tag{5.10c}$$

$$x_a^k \leq \sum_{\tilde{a} \in R} x_{\tilde{a}}^k \qquad \forall k \in K, \forall a \in A^k, \forall R \in \phi^k(a) \tag{5.10d}$$

$$\sum_{a \in R} x_a^k \geq \sum_{v \in V_o^k} b_v^k \qquad \forall k \in K, \forall R \in \Omega^k \tag{5.10e}$$

$$x_a^k \in \mathbb{Z}_{\geq 0} \qquad \forall k \in K, \forall a \in A^k \tag{5.10f}$$

$$y_a \in \{0, 1\} \qquad \forall a \in A \tag{5.10g}$$

A solution to MIP (5.10) is defined by arc activation (binary variables y_a, definition (5.10g)) and flow volumes on arcs (integer variables x_a^k, definition (5.10f)). Equations (5.10b) ensure that the flow will be transported from its source to its

sink (flow conservation). The inequalities (5.10c) limit the flow to the arcs' capacity and guarantees that an arc can only hold flow if it has been activated before (using $y_a = 1$). Deactivated arcs cannot hold any flow ($y_a = 0 \implies x_a^k = 0$). Constraints (5.10d) take care that all arc-requirements that are related to precedence relations are fulfilled. In contrast, constraints (5.10e) handle arc-requirements that are defined independent of a route by summing up the flow on all arcs of an arc-requirement and enforcing that sum to be at least the commodities' volume. Please note that these last constraints are only valid for non-aggregated commodities (with volume $\sum_{v \in V_o^k} b_v^k = 1$), as they consider only flow on arcs but not on paths.

The example in Figure 5.7 illustrates that issue. Considering a commodity k with $b(v_o, k) = 2$ and $b(v_d, k) = -2$ and assuming an arc-requirement $\{ a_2, a_3 \} \in \Omega^k$, there exists a valid solution containing two paths for that commodity: $P_0 = (a_1, a_2, a_3, a_4)$ and $P_1 = (a_5, a_6)$. In that solution, constraints (5.10e) are fulfilled since path P_0 uses both arcs a_2 and a_3 which allows P_1 to use none of them, as the constraints force only the sum of the flow on that arcs to be sufficiently large. That is obviously not the intended result. The only solution we found to solve this issue is prohibiting the aggregation of commodities in that case. The explanations in Section 5.7 and the algorithm presented in Chapter 9 outline the decision-making process whether commodities can be aggregated in more detail.

Even worse, when the arc-requirement is located within a cycle of the time-expanded graph (all arcs must have $\tau(a, t) = 0$ within a cycle) the constraints (5.10d) and (5.10e) will not work at all. In that case, it would be a valid solution to

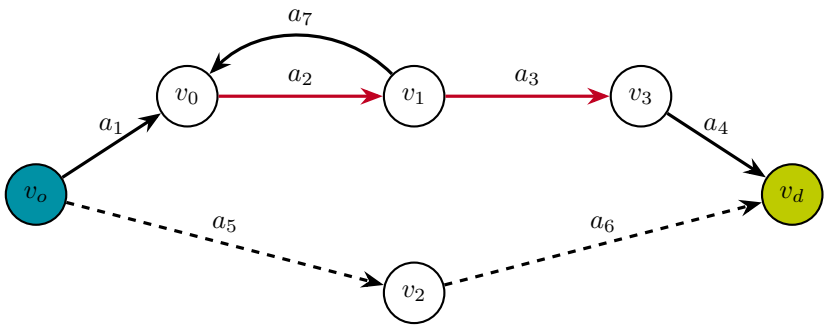


Figure 5.7: Example demonstrating in detail why the arc-requirement constraints of MIP (5.10) are only valid for non-aggregated commodities. A commodity k with $b(v_o, k) = 2$ should travel from v_o to v_d while using at least one of the arcs a_2 or a_3.

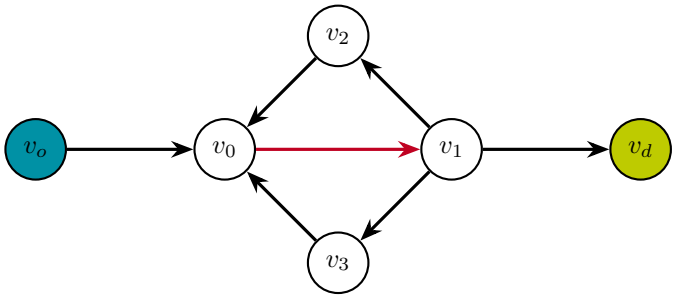


Figure 5.8: Example showing an arc that is contained in different cycles and thus can be used multiple times.

set $x^k_{a_2} = x^k_{a_7} = 2$ (because of the cycle, flow conservation holds) and send the entire commodity volume along P_1. Nevertheless, such situations can be avoided by adding additional constraints to MIP (5.10). There are efficient algorithms available to enumerate all cycles C in a pseudo-graph like the one presented by Hawick and James (2008). However, there might be an exponential amount of cycles, so enumerating all of them might take some time. Whenever an arc-requirement is located in such a cycle, it must be ensured that the flow on the arc is limited by the overall flow that entered the cycle from outside. If no flow has entered the cycle, the flow on the arc must be equal to zero. It will be even more complicated when the arc is contained in more than one cycle. Figure 5.8 shows an example of such a situation. There might be arc-requirements located in both cycles so a commodity might use arc (v_0, v_1) multiple times, once for entering the cycles and once for each round trip (maximum 3 times). However, if no flow is entering any of the cycles the flow within the cycles must be equal to zero. We can express this by adding constraint $x^k_{(v_0v_1)} \leq 3x^k_{(v_ov_0)}$ or $x^k_{(v_0v_1)} \leq 3x^k_{(v_1v_d)}$ to the MIP of our example.

More general, for each arc $a \in A^k$ that is contained in any cycle we can add inequality

$$x^k_a \leq (|\theta(a)| + 1) \sum_{\tilde{a} \in \delta^+(\theta(a))} x^k_{\tilde{a}} \qquad \forall a \in A^k : |\theta(a)| > 0 \tag{5.11}$$

with function $\theta(a) \subseteq C$ defining all cycles that contain arc a and $\delta^+(\theta(a))$ being the cut induced by all nodes that are part of these cycles. Please note that the super-sink node will never be contained in any cycle, so in a valid solution there always will be flow leaving the cycle. Obviously, within a large time-space network

there might be many of these arcs which imply many additional constraints and enumerating all cycles may be an expensive operation anyway. Thus, we are going to add constraints (5.11) only if required. In Section 7.2 we will explain how these constraints can be separated and added lazily to our MIP.

5.6 Complexity, Feasibility & Tractability

When choosing arc capacities u_a sufficiently large (e.g., for all $a \in A$ it is defined as $u_a = \sum_{k \in K} \sum_{v \in V_o^k} b_v^k$) and when there are no arc-requirements specified ($\phi^k(a) = \emptyset \; \forall k \in K, a \in A$ and $\Omega^k = \emptyset \; \forall k \in K$), the problem presented above is equal to the uncapacitated fixed-charge network design problem (UFND) which is proven to be NP-hard by Magnanti and Wong (1984). Thus the PCIMCFND is NP-hard, too. Balakrishnan, Magnanti, and Mirchandani (1997) showed that the problem becomes even harder to solve if arc capacities are considered. So it is unlikely that large instances of the PCIMCFND can be solved to optimality. However, it is not obvious at which size an instance is too large and if real-life instances are still tractable.

Considering the explanations of Chapter 2 and looking at production rates of the larger automobile manufacturers, there might be up to 750 000 assembled vehicles within a 4 weeks planning horizon, that need to be taken into account simultaneously. Lets assume that we are able to aggregate this amount of vehicles to a set of $|K| \approx 375\,000$ commodities, which implies a decent reduction factor of 0.5 in our experience. The largest real-life instances that we used during the work on this thesis, contain $|A| \approx 30\,000$ arcs with an average allowed sub-network size of $\left|A^k\right| \approx 40$ and $\left|V^k\right| \approx 20$ for each commodity in the non-time-expanded network. Even when ignoring arc-requirements and assuming that each arc and node will be available in the time-expanded network on 50 days only, that will lead to an instance of MIP (5.10) with 750 million variables and 375 million constraints. When looking at the size of the instances of the MIPLIB 2017 collection in Figure 5.9 we do not find a single instance of similar size. Hence, it is rational to expect, that the largest of our instances currently cannot be solved to optimality within a reasonable amount of time, even when using a highly-optimized problem specific algorithm. However, to apply our approaches in practice, an optimal solution might not be required, but a (proven) good one might be sufficient. Furthermore, real-life instances will not always be as large as in the computation explained

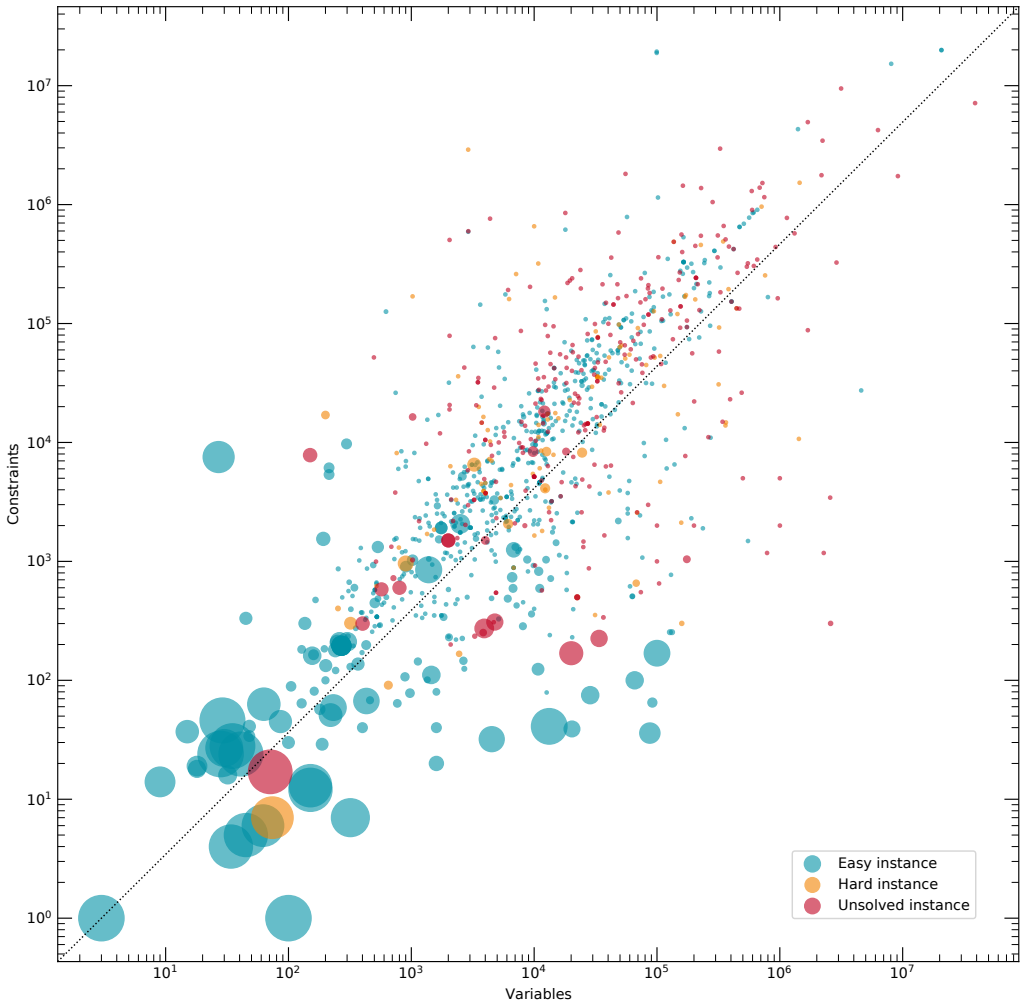


Figure 5.9: Overview of 1065 instances of MIPLIB 2017 collection comparing the amount of variables with the amount of constraints. The area of each bubble corresponds to the density of the instance. The density is defined as the relative amount of nonzero values in the coefficient matrix of the corresponding MIP. Densities smaller than 0.01 are drawn in uniform size. Instances classified as “easy” or “hard” could both be solved to optimality, but solving the first required at most 1 hour computational time of an out-of-the-box solver on standard desktop computing hardware while solving the latter demands for non-standard hardware and / or algorithms. (Source: Zuse Institute 2018)

above. The remainder of this thesis and especially the computational study will show which techniques are sufficient for which instance sizes and where advanced methodologies are unavoidable or even fail.

Besides calculating an optimal solution for an instance, from a business perspective it can be important to report a solution at all – almost regardless of its quality. Let's imagine that a logistics planner is forced to rearrange some transport capacities which leads to a situation where MIP (5.10) becomes infeasible (not all vehicles can be delivered). There will be no solution available to show to the planner and to assist in resolving that capacity shortage. Of course, there are techniques like calculating irreducible inconsistent subsets (IISs), but these are hard to use without profound knowledge of mathematical optimization and deep understanding of the MIP. Typically, both will not be available in practice. Thus, ensuring that MIP (5.10) always is feasible seems to be a reasonable approach. An obvious source of infeasibility are the capacity constraints (5.10c). By introducing an uncapacitated[1] arc $a_{od} = (v_o, v_d)$ for each commodity k and each combination of origin node $v_o \in V_o^k$ and destination node $v_d \in V_d^k$ there will always be a valid path that guarantees the feasibility of that constraints. However, such *shortcut arcs* should only be used in a solution if otherwise the instance would be infeasible. That can be achieved by choosing arc costs $c(k, a_{od}) = M^k$ with constant M^k being reasonable large. A similar technique has been used by Gendron and Larose (2014). Nevertheless, MIP (5.10) might become infeasible when commodity k uses arc a_{od}. There might be arc-requirements constraints (5.10e) that are not fulfilled. However, simply adding a_{od} to each such arc-requirements $R \in \Omega^k$ will resolve that issue.

5.7 Commodity Aggregation

It is well known in literature that commodities in most multi-commodity flow problems can be represented in different levels of aggregation (see Jones et al. 1993; Gendron, Crainic, and Frangioni 1999). Typically, this aggregation is determined by the commodities' source and sink nodes.

[1] Even though in our model each arc has a capacity, choosing $u(a) = \infty$ is equal to being uncapacitated

Single-Source / Single-Sink Commodity k represents all volumes for a certain pair of source and sink nodes. The cardinality of origin nodes $V_o^k = \{ v_o \}$ and destination nodes $V_d^k = \{ v_d \}$ is limited to $\left|V_o^k\right| = \left|V_d^k\right| = 1$. A solution contains a set of $v_o v_d$-paths for each commodity.

Single-Source / Multiple-Sinks In a higher level of aggregation, commodities will be represented only by their source nodes. Again, for the set of origin nodes it holds that $\left|V_o^k\right| = |\{ v_o \}| = 1$. However, a solution will not be composed of paths but of trees with root node v_o for each commodity.

Multiple-Sources / Single-Sinks This is quite similar to the aforementioned case and represents commodities by their sink nodes only. It holds that $\left|V_d^k\right| = |\{ v_d \}| = 1$. Again a solution will contain trees but with root node v_d.

Multiple-Sources / Multiple-Sinks Commodities with different origin and destination nodes are aggregated to a single commodity with $\left|V_o^k\right| > 1$ and $\left|V_d^k\right| > 1$. Obviously, that kind of commodity representation cannot be used in all cases but only if the original commodities are exchangeable in regard to their origin and destination node. Typically, that is not the case in finished vehicle logistics and thus we will not consider that level of aggregation any further.

Aggregating vehicles reduces the number of overall commodities that must be considered and thus might decrease the size of the corresponding MIP. In our case, the quantity of commodities is the dominant factor, while the size of the network plays a subordinate role, since the allowed sub-network of each commodity typically will be small. Looking at the amount of variables and constraints presented in Section 5.6, reducing the amount of commodities can have a reasonable impact on the tractability of a problem instance.

To the best of our knowledge, aggregating by the single-source-single-sink pattern entails no general disadvantages and thus should be considered in any case. Nevertheless, using the single-source-multiple-sink or multiple-source-single-sink approach would further decrease the size of the MIP. Unfortunately, the extensive computational study of Chouman, Crainic, and Gendron (2016) indicates that these kinds of aggregations might imply computational difficulties as it becomes harder to compute tight lower bounds in a branch-and-cut algorithm or convergence

is slower in a branch-and-price algorithm. Similar results have been reported by Jones et al. (1993) and Gendron, Crainic, and Frangioni (1999). These authors indicate that the choice for a commodity representation should always consider the used solution technique. We will return to this discussion in Chapter 15 and Chapter 16 when evaluating our computational results regarding this topic.

When using aggregation to reduce the amount of commodities, it must be guaranteed that the optimal solution of the MIP is not changing. In the case of MIP (5.10) this does not depend on source and sink nodes only. There are other criteria that must be considered. Given commodities $k_1, k_2 \in K$, aggregation of these commodities will not change the optimal solution of MIP (5.10) if certain conditions hold true:

1. Both commodities must be allowed to use the same arcs of the underlying network
$$A^{k_1} = A^{k_2} \tag{5.12}$$

2. Commodity-specific costs are equal for all arcs the commodities are allowed to use
$$c(a, k_1) = c(a, k_2) \qquad \forall a \in A^{k_i}, i = 1, 2 \tag{5.13}$$

3. None of the commodities has any arc-requirements (see Figure 5.7 and related explanations)
$$\phi^{k_1}(a) = \phi^{k_2}(a) = \emptyset \qquad \forall a \in A^{k_i}, i = 1, 2 \tag{5.14}$$
$$\Omega^{k_1} = \Omega^{k_2} = \emptyset \tag{5.15}$$

Only if these conditions are met and origin, destination or both nodes are equal (depending on the case), commodities k_1 and k_2 can safely be aggregated. Of course, the same procedure can be used to aggregate an arbitrary number of commodities, if the conditions apply pair-wise for each of them. For all solution methods described in the next chapters, we will use this aggregation technique to reduce the amount of commodities.

Part II

Exact Approaches

6 | Finding Single Commodity Paths

Before we begin the discussion of exact solution methods, it is worth taking a glance on a problem that will accompany us during the following chapters. Thus, we first present an algorithmic framework that will be used to find a valid shortest path for a single commodity, which is not a trivial task in our case. A mathematical model for that problem will be derived and we will shed some light on complexity results and solution methods.

Using one of the well known standard path search algorithms like the one from Dijkstra (1959) is not an option for our problem domain, since a feasible path for a commodity needs to respect all arc-requirements. Nevertheless, in the next section we will show that this problem can be modeled as a SPPRC and present multiple algorithms to solve that problem to optimality. Furthermore, we will explain that a path for a commodity without any arc-requirements can be found more efficiently using the A* algorithm (Hart, Nilsson, and Raphael 1968).

6.1 Shortest Path Problem with Resource Constraints

Summarizing the SPPRC first might be a good starting point for the explanation of the following path search algorithms. Given is a directed pseudo-graph $\mathcal{D} = (V, A)$ with non-empty sets of nodes V and arcs A. Resource constraints are formulated by means of *(minimal) resource consumption* and *resource intervals*. Let κ be the amount of resources. A *resource vector* is defined by $\boldsymbol{r} = (r^1, r^2, \ldots, r^\kappa)^\top \in \mathbb{R}^\kappa$ and its components are called *resource variables*. A resource vector $\boldsymbol{r}$ is *dominated* by resource vector $\boldsymbol{s}$ if inequality $s^i \leq r^i$ holds for all $i = 1 \ldots \kappa$. Domination will be denoted by $\boldsymbol{s} \leq \boldsymbol{r}$ and the interval $[\boldsymbol{s}, \boldsymbol{r}]$ is defined as the set $\{\, \boldsymbol{x} \in \mathbb{R}^\kappa \mid \boldsymbol{s} \leq \boldsymbol{x} \leq \boldsymbol{r} \,\}$. Such an interval is associated with each node $v \in V$. The resource consumption $\boldsymbol{r}_u$ at node u is determined by *resource extension functions* associated with each arc $a = (v, u) \in A$ for some node v and denoted by $\boldsymbol{\chi}_a = (\chi_a^i)_{i=1}^\kappa$. The function

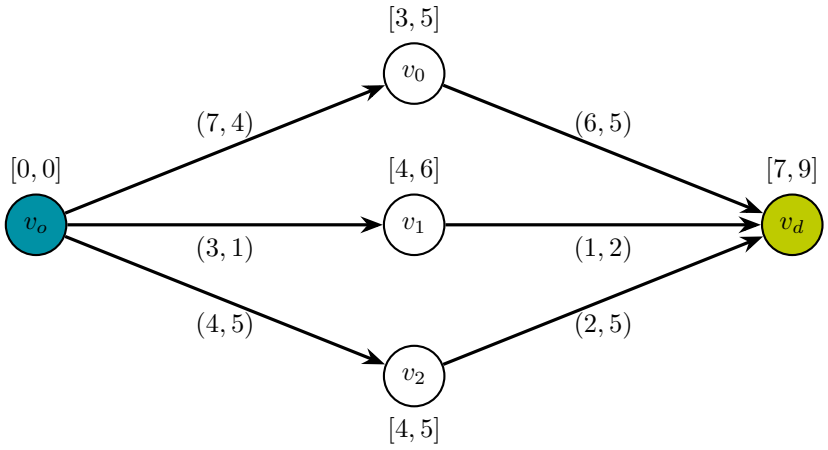


Figure 6.1: A small example for the shortest path problem with time windows

$\chi_a^i : \mathbb{R}^\kappa \to \mathbb{R}$ depends on the resource consumption $\boldsymbol{r}_v$ at node v and propagates that to the tail node u. Thus, along a path the resource consumption will be accumulated up to the destination node. For a more detailed explanation and a generalized definition of resource extension functions the reader is referred to Irnich and Desaulniers (2005).

A well known application for resource constraints are time windows associated with the nodes when a cost-minimal path is required that holds all these time windows. Figure 6.1 shows an example of such a problem instance. The arcs are labeled with costs and traversal times (c_a, τ_a), while the nodes have associated time windows (= resource intervals) when they can be visited. The resource extension function for an arc $a = (v, u)$ is defined as

$$\chi_a(r_v) = \max\{s_u, r_v + \tau_a\} \tag{6.1}$$

with r_v being the accumulated resource consumption at node v and s_u being the start of the time window at node u. As we have only a single resource (traversal time) in this example, we omit the index i on function χ_a and the vector notation. There are three paths connecting origin node v_o with destination node v_d: $P_0 = (v_o, v_0, v_d), P_1 = (v_o, v_1, v_d), P_2 = (v_o, v_2, v_d)$. When using P_0 the resource consumption at v_0 is equal to $r_{v_0} = 4$ which lies within the time window $[3, 5]$. The same applies to the accumulated consumption at node v_d with $r_{v_0} = 9$. Traveling along P_1 leads to an early arrival at node v_1 since the time window is defined

as $[4, 6]$ but the traversal time is $\tau_{(v_o v_1)} = 3$. In that case waiting at the node is allowed by definition which leads to a resource consumption of $r_{v_1} = 4$ and finally to $r_{v_d} = 7$. Path P_2 is invalid as arrival in time at node v_2 is possible but afterwards the arrival at v_d will be out of the time window $[7, 9]$ with a resource consumption of $r_{v_d} = 10$. So only P_0 and P_1 are feasible paths. When looking for the shortest path, P_1 will win due to its lower overall path costs. That small example already gives an idea how different requirements can be modeled within the mathematical framework described above.

Searching for a commodity path that is valid in the sense of our problem domain requires to take all arc-requirements into account. For a given commodity $k \in K$ with a non-empty set of arc-requirements $\Omega^k = \left\{ R_1, R_2, \dots \,\middle|\, R_i \in \Psi^k \right\}$ a resource will be added for each requirement $R_i \in \Omega^k$. Furthermore, the resource extension function for each $a \in R_i$ will be defined by

$$\chi_a^i(\boldsymbol{r}) = r^i + 1 \tag{6.2}$$

with a resource interval specified at the destination node v_d of $[1, \infty)$. That ensures that in a found path one arc of each arc-requirement is visited at least once.

Besides considering route independent arc-requirements Ω^k, there might be others related to the precedence relations which are described by the predecessor and successor arcs functions ϕ_P^k and ϕ_S^k. For a given arc $a \in A^k$ we will add a resource for each pair of arcs (a, a_P) with $a_P \in \phi_P^k(a)$. The same applies to pairs (a, a_S) with $a_S \in \phi_S^k(a)$. The resource extension functions for a, a_P and a_S are defined as

$$\chi_a^i(\boldsymbol{r}) = r^i - 1 \tag{6.3}$$

$$\chi_{a_P}^i(\boldsymbol{r}) = r^i + 1 \tag{6.4}$$

$$\chi_{a_S}^i(\boldsymbol{r}) = r^i + 1. \tag{6.5}$$

Finally, in both cases the resource interval at the destination node v_d has to be $[0, \infty)$ to guarantee that whenever arc a has been used on a path, arcs a_P and a_S have been used at least as frequently (but maybe more often). Besides that the precedence relations of the arcs must be fulfilled. So for the predecessor case there has to be the interval $[1, \infty)$ at the head node of a. Please note that the interval handling at the head of a is a bit more complicated for pseudo-graphs but can be done by introducing artificial intermediate nodes. For the sake of readability, we will ignore that detail in the following.

Besides the business requirements that have been modeled above, there are also some technical requirements that need to be considered within a solution method. The definition of a network in Section 3.2 does not allow negative values for the arc costs function c, however, when using the path search algorithm within a column generation framework (which will be done in Chapter 8) it cannot be guaranteed that $c(a) \geq 0$ for all arcs $a \in A$. In many cases, that is an unproblematic requirement when using time-expanded networks which are typically acyclic. Unfortunately, as shown in Section 5.2, traversal times of $\tau(a) = 0$ are required to model some of our business requirements. Thus, our time-space network might contain cycles and the existence of negative-cost cycles cannot be ruled out. Typically, dynamic programming algorithms for the SPPRC will get stuck in such a cycle and never terminate without explicitly handling that situation. Furthermore, even if termination is not an issue for the algorithm it is well known that using such cycling paths in a branch-and-price procedure might lead to weak lower bounds and large branch-and-bound trees (Irnich and Desaulniers 2005; Chabrier 2006). These issues are usually tackled by adding k-cycle elimination or by solving the elementary shortest path problem with resource constraints (ESPPRC) instead (Irnich and Desaulniers 2005). We will follow the latter approach, however, we will force only elementary paths if the cycles are not required to fulfill any arc-requirement or precedence relation.

6.2 Labeling Algorithm

There has been numerous solution methods proposed in the literature to solve SPPRC or ESPPRC instances to optimality. Approaches based on Lagrangian relaxation, constraint programming, graph modifications or path ranking are just a few of them. Nevertheless, using dynamic programming labeling algorithms is by far the most used solution technique and will be the one presented in the following. For a comprehensive survey on all that methods the reader is referred to Irnich and Desaulniers (2005) or Di Puglia Pugliese and Guerriero (2013).

The SPPRC is NP-hard, even when the arc costs are non-negative and when using a single resource only, e.g., costs and traversal time like in the example above (Garey and Johnson 1979, Problem ND30). However, there exists a pseudo-polynomial algorithm and a polynomial-time approximation scheme (PTAS) (Ziegelmann 2001). The requisite of an elementary path is significant regarding the complexity

Algorithm 6.1 SPPRC Labeling Algorithm

```
1: procedure SolveSPPRC(𝒟, v_o, v_d)
2:     l_0 ← CreateLabel(v_o)                          ▷ Create initial label
3:     ℒ ← ∅                                           ▷ Set of processed labels
4:     Q ← { l_0 }                                     ▷ Queue of pending labels
5:     while Q ≠ ∅ do
6:         l ← ChooseLabel(Q)
7:         Q ← Q \ { l }
8:         for j ← SuccessorsNodes(𝒟, l) do            ▷ For all succeeding neighbors
9:             if IsValidExtension(l, j) then
10:                l_j ← ExtendLabel(l, j)
11:                Q ← Q ∪ { l_j }                     ▷ Add new label to queue
12:            end if
13:        end for
14:        ℒ ← ℒ ∪ { l }
15:        if Dominance check is required then         ▷ Drop dominated labels
16:            Q ← Q ∩ DropDominated(Q ∪ ℒ)
17:            ℒ ← DropDominated(ℒ)
18:        end if
19:    end while
20:    return FindBestLabel(ℒ, v_d)
21: end procedure
```

of the problem. Dror (1994) proved that in this case the problem is NP-hard in the strong sense. It is no surprise that there are lots of heuristic approaches available, especially for the ESPPRC. Fu, Sun, and Rilett (2006) or Di Puglia Pugliese and Guerriero (2013) summarize the major contributions in that area. We will come back to these approaches in Chapter 10, in the following we are going to solve the problem to optimality.

The central idea of a labeling algorithm is propagating a *label* created at the start node v_o to neighboring nodes in the graph until the destination node v_d has been reached. A label will carry some information about the path, the costs and the resource consumption. Of course, a label will only be propagated to another node if its resource consumption matches the node's resource interval. To avoid exponential growth[1] of the amount of labels, each propagated label will be compared to other labels of the same node and it will be verified if the label is dominated by another label or dominates one on its own. Only non-dominated labels will be stored

[1] Exponential growth regarding the size of the graph and not the information stored within a label

for further propagation. The algorithm terminates when all labels have been propagated. The shortest path holding all resource constraints can be derived from the Pareto-optimal[2] labels at the destination node v_d. Algorithm 6.1 shows the textbook implementation of such a labeling algorithm.

Obviously that generic algorithm offers different loosely defined entry points to allow a variety of implementations. How is the next label chosen from the queue Q? How is a valid extension of a label to a successor node defined? How is decided whether a dominance check is required and how is that check carried out? In the following we will take a look on three variants of the algorithm that are used to solve the SPPRC or ESPPRC.

6.3 Adapted Algorithm of Feillet

Feillet et al. (2004) were the first to adapt the classical algorithm for the SPPRC proposed by Desrochers (1986) to solve the ESPPRC. They reused the idea described by Beasley and Christofides (1989) and introduced a binary resource for each node $v \in V$ of a given graph $\mathcal{D} = (V, A)$. Furthermore, they added a resource that counts the amount of visited nodes, being aware that a label cannot dominate another label if it visits more nodes. Having a dedicated resource for that purpose is computationally more efficient than comparing all binary node resources. Using that extended label definition, they were able to avoid each label that visits a node twice, which would be a non-elementary path. However, that label definition will lead to the generation of many more labels that cannot be dominated easily. We modified their algorithm to be able to cope with pseudo-graphs and introduced state-space augmentation to compensate the performance degradation that occurred because we could not apply their concept of unreachable nodes to our problem model.

In their implementation Feillet et al. (2004) decided to not use a queue of labels like it has been done in Algorithm 6.1 but to use a queue of nodes instead. The algorithm will move from node to node and propagate its labels at once to all successors. Obviously, the order of the node traversal is an essential factor here

[2]Since the characteristics of Pareto-optimality have minor relevance for the remainder of this thesis, we do not introduce that term formally here. The interested reader is referred to Irnich and Desaulniers (2005).

to ensure that as many labels as possible are dominated early and not being propagated any further. Additionally, the authors introduced the concept of *unreachable nodes*, a form of resource-based bounding. Consider a label that cannot be extended from node v_i to node v_j due to resource limitations. If the triangle inequality holds for each resource, that same label cannot be extended to v_j if any other node $v_k \in V \setminus \{ v_i, v_j \}$ is visited first. Thus, all labels propagated from v_i can consume the binary node resource for node v_j even if that node has never been visited. That might improve the amount of labels that can be disregarded due to domination. If the triangle inequality does not hold, the resource consumption of the direct edge from v_i to v_j must be replaced by the resource consumption of the shortest path.

Even though the algorithm of Feillet et al. (2004) solves the ESPPRC, it is not directly applicable to our problem without modifications, since the prerequisites for calculating unreachable nodes do not hold in all cases. Consider there is a predecessor relation between arc (v_k, v_j) and the only arc out of v_j. In that case the direct propagation of a label from v_i to v_j would violate the resource limits of that predecessor resource but using v_k in between would be perfectly fine. So branding v_j as an unreachable node for all labels out of v_i would be wrong. We discovered that without the notion of unreachable nodes the fast computation of the shortest path will be a serious challenge, however, we were able to handle this problem using state-space augmentation (see below).

Algorithm 6.2 shows our adopted implementation of the algorithm of Feillet et al. (2004). Compared to the original paper, we have introduced a few unobtrusive modifications. At first, a node is removed from the queue of pending nodes right at the beginning to ensure that it can be added to the queue by a loop arc again. Similarly, instead of iterating over all succeeding nodes the algorithm iterates over all outgoing arcs of the chosen node. Both modifications are necessary to be able to correctly handle pseudo-graphs. Furthermore, there is a IsValidExtension method used which was originally a cycle detection step checking the binary resource for node u only. We will explain that method in more detail later.

The algorithm starts by initializing the queue of pending nodes with start node v_o and the set of labels with a corresponding label l_0. Such a label l consists of a resource vector $\boldsymbol{r}_l$, its resident node v_l, its predecessor label l_l and the last arc a_l the label has been propagated with. The last two are required to be able to reconstruct the path by which a label has been traveled through the graph.

Algorithm 6.2 Adapted Algorithm of Feillet

```
1: procedure SolveESPPRC(𝒟, v_o, v_d)
2:     Q ← { v_o }                                  ▷ Queue of pending nodes
3:     l_0 ← CreateLabel(v_o)                       ▷ Create initial label
4:     ℒ ← { l_0 }                                  ▷ Set of undominated labels
5:     while Q ≠ ∅ do
6:         v ← ChooseNode(Q)
7:         Q ← Q \ { v }
8:         ℒ_v ← GetLabelsAtNode(ℒ, v)
9:         for (v, u) ← OutArcs(𝒟, v) do            ▷ For all outgoing arcs
10:            ℒ_vu ← ∅
11:            for l_v ∈ ℒ_v do
12:                if IsValidExtension(l_v, u) then
13:                    l_u ← ExtendLabel(l_v, u)
14:                    ℒ_vu ← ℒ_vu ∪ { l_u }
15:                end if
16:            end for
17:            ℒ_u ← DropDominated(ℒ_u ∪ ℒ_vu)      ▷ Drop dominated labels
18:            if ℒ_u has changed then
19:                Q ← Q ∪ { u }                    ▷ Add successor node to queue
20:            end if
21:        end for
22:    end while
23:    return FindBestLabel(ℒ, v_d)
24: end procedure
```

The resource vector assembles resources of different kinds:

- Resource r_c will be used to accumulate the path's cost
- For each arc-requirement $R_i \in \Omega^k$ there will be a resource r^i_Ω tracking if a requirement already has been fulfilled
- Precedence relations between arcs will be ensured by dedicated resources $r^i_{\phi_P}$ and $r^i_{\phi_S}$ for each pair (a, a_P) of predecessor and each pair (a, a_S) of successor arcs with $a_P \in \phi^k_P(a)$ and $a_S \in \phi^k_S(a)$
- Each node $v \in V^k$ will be represented by a resource r^i_v to be able to prevent cycles within paths if they are not required. In our case resource r^i_v will be non-binary, as in some cases multiple occurrences of a node might be allowed. Similar to Feillet et al. (2004), we introduced resource r_V that accumulates the amount of visited nodes to speed up computations.

Please note that the definition of labels and the used graph are specific to a single commodity $k \in K$. The graph is limited to the set of allowed nodes and arcs $\mathcal{D}^k = (V^k, A^k)$, the cost function $c(a, k)$ depends on the commodity and all resources are derived from commodity-related set Ω^k and functions ϕ_P^k and ϕ_S^k.

After initialization, the algorithm continues by picking (and removing) a node v from the queue and propagating all of its labels $\mathcal{L}_v$ to successor nodes using the outgoing arcs $\delta^+(v)$ of that node. For each successor node u it is validated, if an extension is possible using the aforementioned method IsValidExtension. All new labels are collected within set $\mathcal{L}_{vu}$ and merged with the labels $\mathcal{L}_u$ of the successor node u while simultaneously removing all dominated labels. Label l_1 is dominated by label l_2 if the corresponding resource vectors dominate each other $\boldsymbol{r}_{l_2} \leq \boldsymbol{r}_{l_1}$. The dominance relation of these labels will be denoted by $l_2 \leq l_1$. Since the concept of dominance is essential for limiting the amount of created labels, the introduction of resources r_v^i for each $V \in V^k$ has a major impact on the algorithm's performance. Having more resources increases the probability that labels are not dominating each other. During the implementation of the algorithm, we observed that the overall time required to find the shortest path increased with a factor between 10 to 1000 after introducing the node resources. Thus, even small benchmark instances became intractable. However, there are techniques available in literature to limit that amount of resources (so called *state-space*). Kohl (1995) and Boland, Dethridge, and Dumitrescu (2006) introduced the concept of *state-space augmentation*. The main idea is to solve the ESPPRC without any node resources (so actually solving the SPPRC) and iteratively adding resources, if a path containing cycles has been found. In the worst case the algorithm is forced to run $\left|V^k\right|$ times and will end with the original state-space formulated above. However, we expect this to be rather unlikely and decided to extend the algorithm of Feillet et al. (2004) accordingly. We followed the suggestion of Boland, Dethridge, and Dumitrescu (2006) adding resources in a conservative manner only for the node that has been visited most frequently in the last iteration. However, if a node has been visited more than once to fulfill any arc-requirement or precedence relation, we expect that cycle to be required for a valid path and will ignore that during the augmentation step. For more details the reader is referred to the work of Boland, Dethridge, and Dumitrescu (2006). Please note that termination of the algorithm can no longer be guaranteed with an incomplete state-space. If cycling is not avoided by dedicated binary node resources there is no possibility for the algorithm to escape from a cycle. Neither do we enforce strictly positive

resource consumption on each arc nor is the resource consumption bounded on each node. Nevertheless, this shortcoming can be circumvented by introducing an artificial resource r_L that consumes a single unit on each arc. Hence, that resource will measure the total length of the path. It will be limited to the resource interval $r_L \in [0, M]$ with M being sufficiently large. Choosing a discrete value for that parameter is not trivial and requires some insights into the structure of the instances that need to be solved. Section 14.5 gives some details on how we have chosen M within our computational study.

The method IsValidExtension is essential for eliminating labels that will not lead to a valid path. Whenever propagating a label to another node this method checks if that step is compliant with all resource intervals. So, for intermediate nodes all predecessor relations are validated, additionally at the destination node v_d the arc-requirements and successor relations. Furthermore, that method will perform a cycle detection and avoidance step. As cycles are not prohibited in general in our case, it is required to categorize them into *avoidable* and *unavoidable cycles*. We expect a cycle to be unavoidable if it fulfills any arc-requirement or precedence relation. More formally, consider label l_1 at the beginning of the cycle and label l_2 at the end (both having the same resident node v), a cycle is called avoidable if the following conditions hold:

$$r^i_{l_1,\Omega} = r^i_{l_2,\Omega} \qquad \forall R_i \in \Omega^k \tag{6.6}$$

$$r^i_{l_1,\phi_P} = r^i_{l_2,\phi_P} \qquad \forall a_i \in \phi^k_P(a), \forall a \in A^k \tag{6.7}$$

$$r^i_{l_1,\phi_S} = r^i_{l_2,\phi_S} \qquad \forall a_i \in \phi^k_S(a), \forall a \in A^k \tag{6.8}$$

Hence, an avoidable cycle will not change the consumption of any resource related to arc-requirements or precedence relations. Similar to the original algorithm of Feillet et al. (2004), all labels completing an avoidable cycle will be disregarded. In contrast, all labels closing an unavoidable cycle will be propagated as usual. That rule will not ensure the termination of the path search. If there exists a negative-cost cycle that changes the resource consumption, the algorithm might get stuck in that unavoidable cycle. However, the artificial resource r_L introduced above, will guarantee termination. In that case, we accept that the resulting path contains a cycle, which might have an impact on the lower bound of our branch-and-price method. We expect that impact to be negligible, since the relative amount of these cycling paths will be typically low, compared to the overall amount of paths. Cycle detection will only be performed for nodes contained in the augmented state-space.

6.4 Performance Improvements

In order to gain some benefits from the concept of unreachable nodes, we implemented similar techniques to the ones that have been introduced by Lozano and Medaglia (2013) and Lozano, Duque, and Medaglia (2015) and extended the validation of resource intervals to potential future nodes. Such extension might reduce the amount of undominated labels and thus improve the overall performance. Consider a (partial) directed path $P = (v_d, \dots, v)$ ending at node v and its corresponding label l. Whenever propagating that label to a successor node u using arc a_{vu}, thereby extending path P to $\tilde{P} = (v_d, \dots, v, a_{vu}, u)$, it is possible to verify weather all arc-requirements and precedence relations still can be fulfilled. For each arc-requirement $R \in \Omega^k$ there must be at least one arc $a \in R$ that can be reached from node u (there exists a directed path from u to $tail(a)$) or which is already included in path $\tilde{P}$. Similarly, all successor arcs $a_S \in \phi_P^k(a)$ must be reachable (or being already included in path $\tilde{P}$) for all arcs $a \in \tilde{P}$ that have been used in path $\tilde{P}$ before. In all situations where an arc-requirement or a successor relation cannot be fulfilled any longer, path $\tilde{P}$ will not lead to a resource-feasible path and the propagation of label l along arc a_{vu} can be discarded.

In contrast, predecessor relations cannot be used to avoid path extensions. However, considering a precedence between arcs (a, a_P) with $a_P \in \phi_P^k(a), a \in A^k$ it is possible to increase the corresponding resource $r^i_{\phi_P}$ if arc a cannot be reached from node u. Although the propagation of label l to node u cannot be avoided in this case, changing the resource consumption at least increases the probability that the propagated label will be dominated by any other label.

Obviously, these reachability checks require some sort of preprocessing, otherwise propagating labels from node to node would be computationally too expensive. Calculating the connectivity of all pairs of nodes upfront can be done easily using a simple depth-first search algorithm. However, in the worst case doing such preprocessing for each commodity $k \in K$ might become costly for large sets of nodes V^k even for a polynomial-time algorithm. Actually, preprocessing the graph for small sets V^k can be an issue just because $|K|$ typically will be large in real-life instances of our problem. We were able to mitigate that scaling issues by determining the connectivity of all node pairs heuristically. For that, we performed the depth-first search on a non-time-expanded support graph, which is significantly smaller than the original time-expanded graph and can be constructed easily.

Given is a non-time-expanded graph $\mathcal{D} = (V, A)$ and its time-expanded counterpart $\mathcal{D}_T = (V_T, A_T)$ with time index set T. We construct a supply graph $\mathcal{D}_S$ that contains an arc $a_i \in A$ if $u_T(a_{i,t}) > 0$ for any $t \in T$. Consequently, each arc of the support graph has positive capacity at any point in time in $\mathcal{D}_T$. Since this support graph has (at most) equal size of the non-time-expanded graph ($|V| \geq |V_S|, |A| \geq |A_S|$) performing an all-pairs connectivity calculation can be done quickly even for real-life instances. The definition of the support graph ensures that connectivity of the time-expanded graph will never be "over-estimated". If nodes v and u are not connected within $\mathcal{D}_S$, their time-expanded counterparts v_t and u_t are not connected for all $t \in T$ either. However, the opposite inference does not hold, so the nodes v_t and u_t might not be connected for some $t \in T$, although nodes v and u of the supply graph are connected. That property of the supply graph ensures that no label will be discarded during a reachability check that should not have been discarded. By contrast, some labels will be propagated that would not be propagated if connectivity had been calculated exactly.

We assume that the limitation of a commodity to the sub-graph induced by A^k has only marginal impact on the connectivity of the nodes in V^k. Thus, the reachability results acquired on $\mathcal{D}_S$ will be used for each commodity $k \in K$. We expect this will be sufficient to speed up the overall shortest path search. However, we will examine this assumption in detail in the computational study in Chapter 14.

Another common technique to speed up computation of the shortest path is pruning by bounds (Fu, Sun, and Rilett 2006; Di Puglia Pugliese and Guerriero 2012; Lozano and Medaglia 2013; Sedeño-Noda and Alonso-Rodríguez 2015). That technique relies on an upper bound $\bar{r}_c$ of the costs of the shortest path. Each label l with cost consumption r_c for which $r_c \geq \bar{r}_c$ can be discarded immediately. Please note that this technique is only applicable if not all Pareto-optimal paths should be found, but finding any path is sufficient, and if there are no negative-cost arcs in the network (or at least not connected to resident node v_l of label l). We have implemented two different approaches for determining an upper bound within our algorithms. First, a valid bound can be derived easily from the costs of an already found valid path. Second, when using the path search in the pricing step of a branch-and-price algorithm, we are only interested in paths with negative reduced costs. This can be transferred trivially into an upper bound of the path search algorithm. Our algorithms always choose the tightest bound from these two approaches.

6.5 Adapted Boost Algorithm

During preliminary testing of our implementations we have investigated, that the strategy of Feillet's algorithm to propagate all labels of a node at once is a severe limitation for some performance improvements that will be explained in the remainder of this chapter. Hence, instead of managing a queue of nodes we wanted to manage a queue of labels. A similar approach has been taken by Desrochers and Soumis (1988) for the shortest path problem with time windows (SPPTW) which is a special case of the SPPRC. An algorithm for the SPPRC with a label queue is publicly available as part of the Boost Graph Library (Drexl 2020). We adopted this algorithm to support all the cycle-detection, node reachability and branch pruning techniques described above. Algorithm 6.3 shows the resulting modified algorithm.

Algorithm 6.3 Adapted Boost Algorithm

```
1:  procedure SolveESPPRC(𝒟, v_o, v_d)
2:      l_0 ← CreateLabel(v_o)                          ▷ Create initial label
3:      ℒ ← ∅                                           ▷ Set of undominated labels
4:      Q ← { l_0 }                                     ▷ Queue of pending labels
5:      while Q ≠ ∅ do
6:          l ← ChooseLabel(Q)
7:          Q ← Q \ { l }
8:          if IsNotDominated(l) then
9:              ℒ_{v_l} ← GetLabelsAtNode(ℒ, v_l)
10:             MarkDominatedLabels(ℒ_{v_l})
11:             ℒ ← DropDominated(ℒ \ Q)
12:         end if
13:         if IsNotDominated(l) then
14:             for (v_l, u) ← OutArcs(𝒟, v_l) do       ▷ For all outgoing arcs
15:                 if IsValidExtension(l, u) then
16:                     l_u ← ExtendLabel(l, u)
17:                     Q ← Q ∪ { l_u }                 ▷ Add new label to queue
18:                     ℒ ← ℒ ∪ { l_u }
19:                 end if
20:             end for
21:         end if
22:     end while
23:     return FindBestLabel(ℒ, v_d)
24: end procedure
```

In contrast to the approach of Feillet et al. (2004) and Algorithm 6.2, this algorithm starts with the initialization of the queue of unprocessed labels with a label created for the origin node v_o. The labels from queue Q are removed piece by piece and processed until the queue is empty. When processing a label l, the algorithm first verifies that the label has not been marked as dominated in an earlier iteration. Afterwards, all labels at the same resident node v_l are checked pair-wise for being dominated and will be marked accordingly. All processed and dominated labels can be deleted immediately. If the current label l is not dominated by any other label, the algorithm continues to propagate the label to all successor nodes of v_l. Again the method IsValidExtension is used to filter labels that will not lead to a valid path from v_o to v_d. If propagation is admissible, the new label will be appended to the queue.

The major distinction between this algorithm and the one proposed by Feillet et al. (2004) is the delayed dominance check. The algorithm allows putting labels in the queue that might be dominated by other labels in the queue. The dominance check is delayed until a label at a node is going to be processed. This might reduce the total amount of dominance checks, e.g., if certain nodes are not processed at all due to branch pruning. However, in average there will be more labels to consider during a pair-wise dominance check if the queue contains dominated and undominated labels. Furthermore, removing the dominated labels after the dominance check will require some CPU cycles, too. We will see in Chapter 14, that this modified algorithm is able to compete with the one presented before in Section 6.3.

6.6 Delayed Dominance Algorithm

Inspired by the adapted Boost algorithm presented above, we were wondering if more CPU cycles can be saved by delaying the dominance checks even further. Looking at an iteration of the Boost algorithm, it is conspicuous that a pair-wise dominance check is performed for all labels of a node, although this is not required. It would be sufficient to validate if the currently processed label is dominated by any other label to be able to skip further propagation. This is the central idea of Algorithm 6.4 which we are going to propose as the *Delayed Dominance* algorithm in the following.

Algorithm 6.4 Delayed Dominance Algorithm

```
1:  procedure SolveESPPRC(𝒟, v_o, v_d)
2:      l_0 ← CreateLabel(v_o)                          ▷ Create initial label
3:      ℒ ← ∅                                           ▷ Set of processed labels
4:      Q ← { l_0 }                                     ▷ Queue of pending labels
5:      while Q ≠ ∅ do
6:          l ← ChooseLabel(Q)
7:          Q ← Q \ { l }
8:          ℒ_{v_l} ← GetLabelsAtNode(ℒ, v_l)
9:          if IsNotDominated(ℒ_{v_l}, l) then
10:             ℒ ← ℒ ∪ { l }
11:             for (v_l, u) ← OutArcs(𝒟, v_l) do       ▷ For all outgoing arcs
12:                 if IsValidExtension(l, u) then
13:                     l_u ← ExtendLabel(l, u)
14:                     Q ← Q ∪ { l_u }                 ▷ Add new label to queue
15:                 end if
16:             end for
17:         end if
18:     end while
19:     return FindBestLabel(ℒ, v_d)
20: end procedure
```

Similar to the Boost algorithm our proposed procedure starts by initializing the queue of pending labels with a label for the origin node v_o. In each iteration one label l from that queue is selected and compared against all the other labels at the same resident node v_l which have been processed before. If the label is not dominated, it is propagated to all successor nodes of v_l and the new labels are added to the queue, after filtering them with method IsValidExtension. The algorithm terminates as soon as queue Q is empty. Please note that a label will never be removed from the set of processed labels $\mathcal{L}$. Hence, when the algorithm terminates, the set might contain dominated and undominated labels. If the algorithm should return all Pareto-optimal shortest paths, this needs to be ensured by a final pair-wise dominance check on all labels at the destination node v_d. Since we are only interested in any shortest path, we will terminate that dominance check with the first undominated label.

Although that procedure has the potential to avoid a significant amount of dominance checks, each dominance check that is performed might require more and more CPU cycles, since the set $\mathcal{L}$ is constantly growing. We expect that the success of this method primarily depends on the order in which the labels are taken from

the queue. If the most promising labels are propagated first, that might lead quickly to good feasible solutions, which can accelerate the overall solution process by excessive branch pruning. We will investigate this in more detail in Chapter 14.

6.7 Label Setting vs. Label Correcting

Up to now we have not provided any details regarding the order in which nodes or labels are taken from their queues, although that might have a major impact on the overall performance. Typically, shortest path algorithms are classified into *label-setting* or *label-correcting* algorithms (Ahuja, Magnanti, and Orlin 1993). The first have the beneficial property that each node needs to be visited at most once. Besides saving iterations by avoiding multiple node visits, that property further guarantees that the algorithm can terminate with an optimal solution as soon as any label has been propagated to the destination node v_d. Additionally, in the context of the SPPRC this property ensures that a label which has been processed at a node can never be dominated by another label afterwards. Thus, the amount of dominance checks might be reduced. Nevertheless, the label-setting property is directly related to the order of the nodes or labels in the queue. Hence, the algorithm must take great care in maintaining the proper order of the queue, which might be computationally expensive. Although label-setting algorithms typically have a superior worst-case complexity bound compared to label-correcting algorithms, that does not guarantee a superior average-case performance on real-world instances (Dial et al. 1979; Bertsekas 1993).

For graphs with particular characteristics efficient ordering schemes have been proposed in the literature that guarantee the above property, like the topological ordering for acyclic graphs (Cormen et al. 2013, pp. 655–658). When solving the SPPRC or ESPPRC, typically a lexicographical ordering of the labels' resource vectors is used (Desrochers and Soumis 1988; Powell and Chen 1997; Boland, Dethridge, and Dumitrescu 2006). However, this is only admissible if the graph ensures strictly positive resource consumption on all arcs. As none of these graph characteristics is applicable in our case, we decided to implement a computational fast and simple ordering of labels or nodes instead of spending more computing time in maintaining a sophisticated order. Nodes of the time-expanded graph will be sorted first by time and second by an arbitrary (but deterministic) node ID. The same rule is applied when sorting labels using the information of their resident

node. This ordering scheme is equivalent to the topological ordering if all traversal times are strictly positive. We will discuss an alternative sorting strategy at the end of this section.

6.8 Efficiently Storing Labels and Nodes

In all algorithms mentioned above we used a queue Q and a set of labels $\mathcal{L}$ without specifying the details of those data structures. Obviously, within the implementation of the algorithms just using a set data structure like it has been done in the pseudo-code would not perform very well. All queues should be able to efficiently maintain an order of the nodes or labels. Furthermore, it must be possible to efficiently select all labels of a certain node from $\mathcal{L}$ and add labels to or remove labels from that selection. Several sophisticated data structures have been proposed in the literature to be able to quickly select only labels that are required for a dominance check. The *generalized bucket* proposed by Desrochers and Soumis (1988) or the *bucket graph* recently suggested by Sadykov, Uchoa, and Pessoa (2017) are just a few of them. Lozano and Medaglia (2013) presented an approach that stores only a thoroughly selected subset of the labels. Nevertheless, these data structures add significantly more complexity to an algorithm that we were not willing to take at the first attempt. We leave it to further research, if such data structures yield benefit for the algorithms proposed above.

In our implementations the set $\mathcal{L}$ is equivalent to a double-linked list of labels for each node of the time-expanded graph. Hence, iterating all labels of a node and adding and removing labels can be performed in constant time. For the Boost and the Delayed Dominance algorithm the queue Q has been implemented as an array-based binary heap. This ensures that the first element of the queue can be picked efficiently while simultaneously maintaining an order of all elements. As our implementation of Feillet's algorithm required that nodes cannot be added to the queue twice, we decided to use a red-black tree-based queue instead. This data structure guarantees a more efficient lookup of an element compared to a binary heap, although it entails some memory overhead compared to an array-based heap.

6.9 Accelerate Path Search with A* Algorithm

In the most general case of our problem solving the ESPPRC for each commodity is required to ensure that the shortest paths adhere to all arc-requirements and precedence relations. However, in a real-world instance, we expect the majority of vehicles to have no arc-requirements or precedence limitations at all. Searching for the shortest path using the previously described techniques would be rather inefficient in these cases. Instead, one of the classical shortest path algorithms like the one from Dijkstra (1959) can be used. Fortunately, we can use further knowledge on the connection between the non-expanded and the time-expanded graph to speed up the path search.

The A* algorithm is a (guided) path search algorithm that first has been proposed by Hart, Nilsson, and Raphael (1968). It is a generalization of Dijkstra's algorithm (Thomas, Calogiuri, and Hewitt 2019). For each node $v \in V$ of a given digraph $\mathcal{D} = (V, A)$ and for each destination node $v_d \in V$ a cost estimate function $\hat{r}_c : V \times V \to \mathbb{R}_{\geq 0}$ is required that assesses the costs from node v to the destination node. While Dijkstra's algorithm continues in each iteration with the (partial) path P that has the lowest path costs $r_c(P)$, in A* search the estimated costs $r_c(P) + \hat{r}_c(v_P, v_d)$ of a complete path are used instead (with v_P being the end node of path P). There are several papers stating the success of this approach. Yanagisawa (2006) uses A* search within a Lagrangian relaxation based approach to solve multi-commodity flow problems. Recently, Thomas, Calogiuri, and Hewitt (2019) proposed a bi-directional A* algorithm to tackle the SPPRC. Nevertheless, the success of the method mainly depends on the quality of the cost estimates. If for a given optimal path $\overset{*}{P}_{vv_d} = (v, \ldots, v_d)$ from node v to node v_d it is valid that

$$\hat{r}_c(v, v_d) \leq r_c(\overset{*}{P}_{vv_d}) \qquad \forall v, v_d \in V \tag{6.9}$$

the cost estimate function $\hat{r}_c$ is called *admissible*. Thus, it never overestimates the actual shortest path costs. Furthermore, if the inequality

$$\hat{r}_c(v, v_d) \leq r_c(\overset{*}{P}_{vu}) + \hat{r}_c(u, v_d) \qquad \forall v, u \in V \tag{6.10}$$

holds with $\overset{*}{P}_{vu}$ being the shortest path from v to u, the cost estimate function is termed *consistent*. Please note that consistency implies admissibility (Hart, Nilsson, and Raphael 1968). It is well known that the A* algorithm guarantees termination

with the shortest path if the cost estimate function is admissible. Moreover, the set of explored nodes will be minimal for a consistent cost estimate function. Thus, there cannot be any other algorithm expanding fewer nodes while using the same estimates (Dechter and Pearl 1985).

Exploiting the connection between a non-time-expanded graph $\mathcal{D} = (V, A)$ and its time-expanded counterpart $\mathcal{D}_T = (V_T, A_T)$ with time index set T, a consistent cost estimate function can easily be deduced. To this end, we will reuse the idea of support graph $\mathcal{D}_S = (V_S, A_S)$ introduced in Section 6.4. That graph will contain an arc $a_i \in A$ if $u_T(a_{i,t}) > 0$ for any $t \in T$, hence each arc of the support graph has positive capacity at any point in time in $\mathcal{D}_T$. Furthermore, we define the cost function $c_S : A_S \to \mathbb{R}_{\geq 0}$ to be

$$c_S(a_i) = \min_{t \in T} \{\, c_T(a_{i,t}) \,\} \qquad \forall a_i \in A_S, a_{i,t} \in A_T \tag{6.11}$$

Calculating the single-sink shortest path tree for each destination node $v_d \in V_S$ will lead canonically to a consistent cost estimate function for paths in D_T. In contrast to the reachability calculations presented in Section 6.4, we are going to construct a separate support graph for each commodity $k \in K$ instead of using a single one for all commodities. In preliminary testing, we realized that arc costs of the commodities are too diverse to deliver decent estimates otherwise.

Please note that A* search can only be used for commodities without any arc-requirements or precedence relations. Furthermore, there are only non-negative arc costs allowed within the network. Nevertheless, the general idea of using a lower bound on the remaining path costs might also be beneficial within our algorithms for the ESPPRC. On the one hand, that might improve the upper bound pruning when using complete path cost estimates instead of the current path costs. On the other hand, it can be used as an ordering scheme for labels and nodes. Labels can be sorted by their expected costs when being propagated further to the destination node v_d. For nodes, the minimal expected costs of all labels at that nodes can be used. Furthermore, whenever cost estimates are calculated for a network, performing the reachability calculation mentioned above is no more necessary as it will be a by-product of the shortest path search. We will examine the benefits of this approach in Chapter 14.

While we are calculating the shortest paths on the support graph D_S for cost estimation within the A* algorithm, that estimates can be improved in the course

of our algorithms for the ESPPRC. We will show this in the following. Consider a label l and a set of arcs $A_l = \{ a_1, a_2, \ldots, a_h \}$ that needs to be used before reaching destination node v_d due to arc-requirements or successor relations. Assume set $\mathcal{H}_{A_l}$ to contain all permutations of that arcs set

$$\mathcal{H}_{A_l} = \{ (a_1, a_2, \ldots, a_h), (a_2, a_1, \ldots, a_h), (a_h, a_1, \ldots, a_2), \ldots \} \tag{6.12}$$

with $|\mathcal{H}_{A_l}| = h!$. The remaining path cost estimate for label l at resident node v_l can be calculated to

$$\hat{r}_c(v_l, v_d) = \min_{H \in \mathcal{H}_A} \{ \hat{r}_c(v_l, a_{H_1}) + \hat{r}_c(a_{H_1}, a_{H_2}) + \ldots + \hat{r}_c(a_{H_h}, v_d) \} \tag{6.13}$$

where $H = (a_{H_1}, a_{H_2}, \ldots, a_{H_h})$ and $\hat{r}_c(a_1, a_2)$ being an abbreviating notation for $\hat{r}_c(tail(a_1), tail(a_2))$. Thus, we determine the path costs when visiting all required arcs in all possible permutations and choose the sequence with the lowest costs as estimate. Obviously, that will be a lower bound on the true path cost. Actually, finding all unique permutations is not as trivial as in the example above, when combining arc-requirements and successor arcs. For each arc-requirement it is sufficient to choose one of several arcs and not all of them. Furthermore, it must be ensured that there are no duplicate arcs within a permutation. Otherwise, the estimate might not be a valid lower bound. We will omit the details of that permutation-building algorithm here, since they are not required for understanding the general idea. It might be worth noting that a similar approach has been proposed by Lozano, Duque, and Medaglia (2015). However, they calculated the cost estimates for all nodes in a preprocessing step and used them during path search.

7 | Branch-and-Cut

7.1 Solving with Out-of-the-Box Solver

After having presented a mathematical model for our problem in the previous chapters, in the following we are going to look into different solution techniques for instances of MIP (5.10). The simplest approach one can think of is loading such an instance to one of the well known out-of-the-box solvers like CPLEX, Gurobi, SCIP or FICO Xpress and let them decide how to solve the given instance. Typically, these solvers use a combination of pre-solving, branch-and-cut and heuristics to find proven optimal solutions. That simple approach will work for most small and a few medium size instances (see Chapter 15 for details), however in particular for large instances it is required to use more advanced solution techniques to address the issues that may occur.

For example, a given instance is too large and either cannot be loaded into computer memory or solving takes too much time. The latter may be caused by an excessively growing branch-and-bound tree or repeatedly solving the LP-relaxation of the MIP is consuming too much computing time. Additionally, all solvers are equipped with a set of heuristics for finding "good" feasible solutions. These heuristics might struggle to find a feasible solution at all or only with poor solution quality. Finally, even if the optimal solution has been found by the solver, it is not trivial to prove its optimality. All solvers will try to generate cuts to close the gap between the value of the best known feasible solution and the solution value of the LP-relaxation. However, these general purpose cuts might not be that effective and need to be extended with problem specific cuts.

In the remainder of this chapter we will discuss different approaches to tackle those challenges. Their impact on the overall solution process will be examined in a computational study in Chapter 15.

7.2 Generating Lazy Constraints

Before trying to speed up the solution process, first we need to take care, that solutions obtained by one of the solvers are valid for our problem domain. We explained in Section 5.5 that a valid solution might require the generation of lazy constraints of form

$$x_a^k \leq (|\theta(a)| + 1) \sum_{\tilde{a} \in \delta^+(\theta(a))} x_{\tilde{a}}^k \qquad \forall a \in A^k : |\theta(a)| > 0 \tag{7.1}$$

with function $\theta(a)$ defining all cycles that contain arc a and $\delta^+(\theta(a))$ being the cut induced by all nodes that are part of these cycles. In the following, we are going to explain the algorithm used for the generation of these inequalities.

Whenever the used out-of-the-box solver has found a (local) optimal integer solution for MIP (5.10), this solution can be inspected by the solver's callback mechanism[1]. For commodities without any arc-requirements or precedence relations, an optimal solution will never contain any unavoidable cycles, so none of the lazy constraints is required in that case. For the others, all utilized paths and cycles are determined using the flow decomposition algorithm described in Section 7.3. Afterwards, each of the cycles is classified to being avoidable or unavoidable (following the definition from Section 6.3). Please note that due to the lack of negative costs, all avoidable cycles must have zero-costs and thus can be ignored, in contrast to the elaborations in Chapter 6. Nevertheless, any of the unavoidable cycles might not be connected to a path from the commodity's origin to its destination. That is the case when the generation of the lazy constraint is necessary.

For each of these cycles the arc a is determined that forces the cycle to be unavoidable. This might be the arc contained in an arc-requirement or it might be a predecessor or successor arc. If there are multiple arcs within the same cycle, we choose one of them randomly. Afterwards, all cycles $\theta(a)$ that contain arc a are determined using the algorithm described in Hawick and James (2008). The search space of this algorithm is limited to the commodity's arc set A^k which should ensure a quick termination. Finally, the resulting lazy constraint is added to the root node of the branch-and-bound tree which implies that the current (local) optimal solution becomes invalid and forces the solver to continue. If no lazy constraint has been found, the MIP solution will be considered as valid.

[1] We used Gurobi's `GRBCallback`, but all mentioned solvers offer a similar functionality.

7.3 Transform Arc-based Flow to Paths and Cycles

Now that we are able to gather a feasible solution for an instance of MIP (5.10), the corresponding multi-commodity flow $x : A \times K \to \mathbb{Z}_{\geq 0}$ must be transformed into a solution to our problem, which will be a set of paths $P^k = \left\{ p_1^k, p_2^k, \dots \right\}$ for each commodity $k \in K$ (see Section 5.4). Ahuja, Magnanti, and Orlin (1993, Chapter 3.5) proved that such arc-based flow always can be decomposed into flow on paths and cycles using a flow decomposition algorithm. The textbook version of that algorithm is described in Algorithm 7.1.

The algorithm starts by iterating for each commodity k over all its source nodes $v_o \in V$ with $b(v_o, k) < 0$ and selecting an outgoing arc $a_{v_o v_1}$ from such node with positive flow $x(a_{v_o v_1}, k) > 0$. That arc is used to begin a new path $p = (a_{v_o v_1})$. Afterwards, another arc $a_{v_1 v_2}$ with positive flow out of node v_1 is appended to that path $p = (a_{v_o v_1}, a_{v_1 v_2})$, and so forth, until a sink node $v_d \in V$ is reached or node $v_i \in V$ is visited twice within path p. In the first case, a path from an origin to a destination of commodity k has been found and will be added to the set of paths $p \in P^k$. In the second case, the cycle containing node v_i must be extracted from path p, by removing all arcs that are not part of that cycle, and will be added to the set of cycles $p \in C^k$. Irrespective of whether an origin-destination path or a cycle has been identified, the minimum flow $d = \min_{a \in p} \{ x(a, k) \}$ on the arcs of that path or cycle must be consumed to ensure that it cannot be used in a subsequent iteration of the algorithm ($x(a, k) = x(a, k) - d$). When all flow originating in any source node has been consumed, the algorithm continues by randomly selecting any arc a with positive flow and performs the overall procedure again to detect a cycle p that contains arc a. Again, the found cycle is added to set C^k and the corresponding arc flow is depleted. The algorithm terminates as soon as there is no remaining flow to consume, $x(a, k) = 0$ for all $k \in K, a \in A^k$.

For an application of multi-commodity flow with typical non-negative costs, the cycles obtained by that algorithm can be just ignored. We will do the same for all commodities without arc-requirements or precedence relations. However, for the other commodities such cycles may be required and must be considered within a solution. Hence, before dropping the cycles, we verify that none of them must be merged with a path to make that path feasible in regard to arc-requirements or precedence constraints (unavoidable cycles). Since these commodities have a

single path from origin to destination, such composition of path and cycles will be always possible and unambiguous.

Algorithm 7.1 Flow Decomposition

procedure DecomposeFlow($x, \mathcal{N} = (V, A, u, c, \tilde{c}, b)$)
  for $k \in K$ **do**
    $P^k \leftarrow \emptyset$, $C^k \leftarrow \emptyset$ ▷ Set of origin-destination paths / cycles
    $V_o \leftarrow \{ v \in V \mid b(v,k) < 0 \}$ ▷ Set of origin nodes

    for $v_o \in V_o$ **do**
      $p \leftarrow$ FindPathOrCycle($x, \mathcal{N}, v_o$) ▷ Might return path or cycle
      if IsCycle(p) **then**
        $C^k \leftarrow C^k \cup \{ p \}$
      else
        $P^k \leftarrow P^k \cup \{ p \}$
      end if
    end for

    for $a \in \left\{ a \in A^k \,\middle|\, x(a,k) > 0 \right\}$ **do** ▷ There might be further cycles
      $p \leftarrow$ FindPathOrCycle($x, \mathcal{N}, tail(a)$)
      $C^k \leftarrow C^k \cup \{ p \}$
    end for
  end for
  return $\left\{ (P^k, C^k) \,\middle|\, k \in K \right\}$
end procedure

procedure FindPathOrCycle($x, \mathcal{N} = (V, A, u, c, \tilde{c}, b), v_o$)
  $V_d \leftarrow \{ v \in V \mid b(v,k) > 0 \}$ ▷ Set of destination nodes
  $p \leftarrow \emptyset$, $v \leftarrow v_o$
  while $v \notin V_d, v \notin p$ **do**
    $a \leftarrow$ SelectRandomArc($\{ a \in \delta^+(v) \mid x(a,k) > 0 \}$)
    $p \leftarrow (p, a)$, $v \leftarrow head(a)$
  end while
  if $v \notin V_d$ **then** ▷ Did we end in a cycle?
    $p \leftarrow$ ExtractCycle(p)
  end if
  $d \leftarrow \min_{a \in p} \{ x(a,k) \}$ ▷ Flow volume on path or cycle
  for $a \in p$ **do**
    $x(a,k) \leftarrow x(a,k) - d$
  end for
  return p
end procedure

7.4 Generating Cuts

It is a well known result in literature that the LP-relaxation of MCFND provides a weak lower bound in the context of a branch-and-bound procedure (Gendron and Crainic 1994; Gendron, Crainic, and Frangioni 1999). We expect the same to be true for MIP (5.10), which is supported by the MIP gap of the root node calculated for some of our benchmark instances (see Figure 7.1). There we see an average gap of 10 % with a maximum of 133 %. Typically, such weakness is compensated by generating *valid inequalities* (or *cuts* / *cutting planes*) that are redundant for the corresponding MIP (in contrast to above lazy constraints), but strengthen its LP-relaxation. There have been numerous valid inequalities suggested in the literature for solving commodity flow or network design problems. Hence, we will continue with a comprehensive review of these existing results. A similar survey has been presented by Marchand et al. (2002).

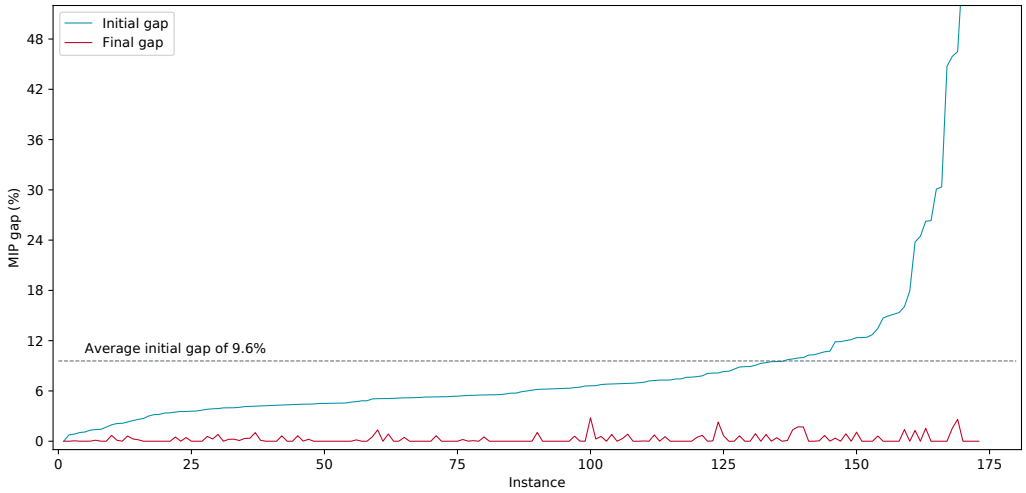


Figure 7.1: Chart showing the initial MIP gap after solving the root node compared to the final MIP gap acquired after solving for 2 hours using Gurobi on benchmark instances C1D5, C5D5 and C10D10. The MIP gap represents the relative difference from the best known solution to the best lower bound calculated from the LP-relaxation.

7.4.1 Literature Review of Valid Inequalities

Strong linking / forcing inequalities The most widely used valid inequalities for network design problems, first introduced by Magnanti, Mirchandani, and Vachani (1993), can be obtained by disaggregating the capacity constraints to

$$x_a^k \leq d^k y_a \qquad \text{with} \quad d^k = \sum_{v \in V : b_v^k > 0} b_v^k \qquad \forall k \in K, a \in A^k. \tag{7.2}$$

The efficiency of these cuts has been reported by several authors and seems to be essential for the success of a branch-and-cut method (Katayama, Chen, and Kubo 2009; Andersen et al. 2010; Hewitt, Nemhauser, and Savelsbergh 2012; Gendron and Larose 2014). Unfortunately, this inequality is valid only for non-cyclic paths and thus cannot be used for commodities with arc-requirements or precedence relations.

Residual capacity inequalities Given is a single arc $a \in A$ and an arbitrary subset of commodities $Q \subseteq K$. For $x_a^k \in [0,1]$ and $y_a \in \mathbb{Z}_{\geq 0}$ the following inequalities hold:

$$\sum_{k \in Q} \frac{d^k}{u_a}(1 - x_a^k) \geq \beta_a^Q(\gamma_a^Q - y_a) \tag{7.3}$$

$$\text{with} \qquad d(Q) = \sum_{k \in Q} d^k \qquad\qquad \alpha_a^Q = \frac{d(Q)}{u_a}$$

$$\beta_a^Q = \alpha_a^Q - \left\lfloor \alpha_a^Q \right\rfloor \qquad\qquad \gamma_a^Q = \left\lceil \alpha_a^Q \right\rceil$$

These inequalities generalize the strong linking constraints for the case $y_a > 1$, as in that case the strong linking constraints are no longer forcing constraints (Frangioni and Gendron 2009). Obviously, these inequalities suffer from the same weakness regarding cycling paths.

Cover inequalities Considering the special case that $x_a^k \in \{0, 1\}$, Barnhart, Hane, and Vance (2000) show that the cover inequalities used for 0–1 knapsack problems can be used for integer multi-commodity flow problems, too. For a given arc $a \in A$ the subset $Q \subseteq K$ of commodities is called a cover, if the commodity demands exceed the arc's capacity: $\sum_{k \in Q} d^k > u_a$. The cover Q is minimal, if for each $l \in Q$

holds that $\sum_{k \in Q} d^k - d^l \leq u_a$. Such minimal covers give rise to valid inequalities of the form

$$\sum_{k \in Q} x_a^k \leq |Q| - 1. \tag{7.4}$$

Knapsack inequalities Let $S \subset V$ be a non-empty subset of the nodes V, $\bar{S} = V \backslash S$ its complement and $\delta^+(S) \subseteq A$ its induced cut, connecting a node in S to a node in $\bar{S}$. With $d(S, \bar{S})$ being the amount of commodity flow originating in S and having its destination in $\bar{S}$, the inequality

$$\sum_{a \in \delta^+(S)} u_a y_a \geq d(S, \bar{S}) \qquad \forall S \subset V, S \neq \emptyset \tag{7.5}$$

holds (Magnanti, Mirchandani, and Vachani 1993; Barahona 1996) given function $d(S, \bar{S})$ to be defined as

$$d(S, \bar{S}) = \sum_{k \in K} \sum_{v \in \bar{S}} b_v^k. \tag{7.6}$$

Please note that above inequalities may only be facet-defining, if both graphs induced by sets S and $\bar{S}$ are connected (Gong 1996; Bienstock and Günlük 1996; Luo and Kianfar 2018). There exist variants of these inequalities, if different kinds of y_a variables exist like in the models of Bienstock and Günlük (1996), Günlük (1999), and Chabrier (2003). Furthermore, there are generalizations to p-partitions of the nodes set (Magnanti, Mirchandani, and Vachani 1993; Barahona 1996; Bienstock et al. 1998; Agarwal 2006). However, they all utilize the same idea: The activated arcs within the induced cut must have sufficient capacity to ensure the transport of all commodity demand.

Metric inequalities Costa, Cordeau, and Gendron (2009) show that a special variant of Benders inequalities, the so-called metric inequalities, dominates above knapsack inequalities, in case a commodity uses multiple arcs from cut $\delta^+(S)$. Given the amount of used arcs $\alpha_{v,S}^k \in \mathbb{Z}_{\geq 0}$ for each commodity $k \in K$ and each destination node $v \in V_d^k$, the calculation of $d(S, \bar{S})$ must be changed to

$$d(S, \bar{S})_{\text{metric}} = \sum_{k \in K} \sum_{v \in \bar{S}} \alpha_{v,S}^k b_v^k. \tag{7.7}$$

Bienstock et al. (1998) and Chouman, Crainic, and Gendron (2016) present a path-based scheme for determining $\alpha_{v,S}^k$ in a separation algorithm. Please note that Definition (7.7) is valid only if commodities are aggregated as single-source-single-sink or single-source-multiple-sinks. However, it can be adopted easily to support the multiple-source-single-sink aggregation.

Flow cut-set inequalities Flow cut-set inequalities are an extension of the above knapsack inequalities that incorporate flow variables. Let $Q \subseteq K$ be a set of commodities. Furthermore, $\{C_1, C_2\}$ is a partition of the cut $\delta^+(S)$. Bienstock et al. (1998) prove that the inequality

$$\sum_{k \in Q} \sum_{a \in C_1} x_a^k + \sum_{a \in C_2} u_a y_a \geq d_Q(S, \bar{S}) \tag{7.8}$$

holds with $d_Q(S, \bar{S})$ being a lower bound on the demand of commodities Q that must be transported from S to $\bar{S}$. In a computational study Raack et al. (2011) showed, that separating flow cut-set inequalities will have only a marginal impact on the lower bound in a branch-and-cut method compared to separating the simpler knapsack inequalities. Please note that there exist more complex approaches extending the knapsack inequalities with flow variables, like the ones presented by Atamtürk (2002) or the network cut-set inequalities used in Chouman, Crainic, and Gendron (2003).

Dicut inequalities For uncapacitated network design problems Chabrier (2003) showed that the knapsack inequalities can be simplified to the dicut inequalities

$$\sum_{a \in \delta^+(S)} y_a \geq 1 \qquad \forall S \subset V, S \neq \emptyset \tag{7.9}$$

if there is any positive demand flowing out of S ($d(S, \bar{S}) > 0$). Similar simplification, which has been presented first by Ortega and Wolsey (2003), is valid for flow cut-set inequalities

$$\sum_{k \in K} \sum_{a \in C_1} x_a^k + d(S, \bar{S}) \sum_{a \in C_2} y_a \geq d(S, \bar{S}). \tag{7.10}$$

n-step general cut-set inequalities Luo and Kianfar (2018) show that most cut-based inequalities presented above can be generalized to the n-step general

cut-set inequalities. Nevertheless, the authors point out that separating these general inequalities may be computationally more expensive than separating simpler inequalities, which have the larger impact on the lower bound anyway. For more details the interested reader is referred to the original paper.

Minimum cardinality inequalities Assuming (w.l.o.g.) that the capacities of the arcs in the induced cut $\delta^+(S)$ are given in non-increasing order: $u_{a_{(i)}} \geq u_{a_{(i+1)}}, i = 1, \ldots, \left|\delta^+(S)\right| - 1$. Chouman, Crainic, and Gendron (2009) show that the minimum cardinalty inequalities

$$\sum_{a \in \delta^+(S)} y_a \geq \max \left\{ h \,\middle|\, \sum_{i=1,\ldots,h} u_{a_{(i)}} < d(S, \bar{S}) \right\} + 1 \tag{7.11}$$

are valid for the MCFND. These inequalities enforce that the amount of open arcs ($y_a = 1$) is sufficiently sized to transport all commodity demand. They can be interpreted as a capacity-aware variant of the dicut inequalities.

Cut-set cover inequalities A set $C \subseteq \delta^+(S)$ is called a cover if the capacity of the arcs in $\bar{C} = \delta^+(S) \setminus C$ is not sufficient to cover the commodity demand:

$$\sum_{a \in \bar{C}} u_a < d(S, \bar{S}) \tag{7.12}$$

Moreover, the cover C is considered minimal if opening any arc $a_0 \in C$ is sufficient to cover the demand:

$$\sum_{a \in \bar{C}} u_a + u_{a_0} \geq d(S, \bar{S}) \qquad \forall a_0 \in C \tag{7.13}$$

For each cover C Chouman, Crainic, and Gendron (2009) showed that the cut-set cover inequality is valid for the MCFND:

$$\sum_{a \in C} y_a \geq 1 \tag{7.14}$$

If C is a minimal cover, this cut may be facet-defining. A generalization to a p-partition of the nodes can be found in Agarwal and Aneja (2017).

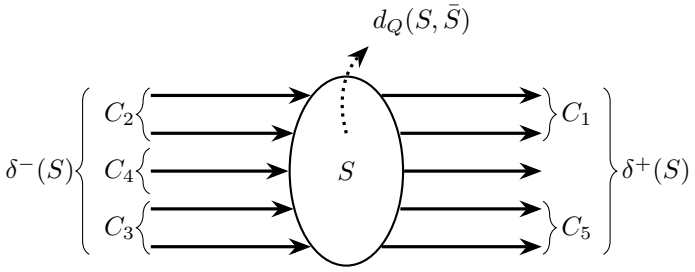


Figure 7.2: Visualization of arc sets used for the definition of flow cover and flow pack inequalities.

Flow cover inequalities For a subset $Q \subseteq K$ of commodities and an arbitrary arc $a \in A$ we define $d_a^Q = \min\Big\{ u_a, \sum_{k \in Q} d^k \Big\}$ to be an upper bound on the flow of commodities Q on arc a. A flow cover (C_1, C_2) is defined by sets $C_1 \subseteq \delta^+(S)$ and $C_2 \subseteq \delta^-(S)$ such that holds:

$$\beta = \sum_{a \in C_1} d_a^Q - \sum_{a \in C_2} d_a^Q - d_Q(S, \bar{S}) > 0 \tag{7.15}$$

Hence, the capacity of the arcs C_1 out of S must be larger than the flow originating in S or entering S using arcs C_2. Atamtürk (2001) and Chouman, Crainic, and Gendron (2009) prove that the flow cover inequality

$$\begin{aligned} \sum_{k \in Q} \sum_{a \in C_1} x_a^k + \sum_{a \in C_1} r_a^Q (1 - y_a) \leq d_Q(S, \bar{S}) + \sum_{a \in C_2} d_a^Q \\ + \sum_{a \in C_3} \min\Big\{ d_a^Q, \beta \Big\} y_a + \sum_{k \in Q} \sum_{a \in C_4} x_a^k \end{aligned} \tag{7.16}$$

is valid for the MCFND with $r_a^Q = \max\Big\{ 0, d_a^Q - \beta \Big\}$, $C_3 \subseteq \delta^-(S) \setminus C_2$ and $C_4 \subseteq \delta^-(S) \setminus (C_2 \cup C_3)$. Figure 7.2 visualizes the different arc sets used in the previous definitions.

Flow pack inequalities A flow pack (C_1, C_2) with $C_1 \subseteq \delta^+(S)$ and $C_2 \subseteq \delta^-(S)$ can be defined similar to a flow cover but requires that

$$\beta = \sum_{a \in C_1} d_a^Q - \sum_{a \in C_2} d_a^Q - d_Q(S, \bar{S}) < 0. \tag{7.17}$$

So in that case, the capacity of the arcs C_1 out of S must be smaller than the flow originating in S or entering S using arcs C_2. According to the work of Atamtürk (2001) and Chouman, Crainic, and Gendron (2009) the flow pack inequalities

$$\sum_{k \in Q} \sum_{a \in C_1 \cup C_5} x_a^k - \sum_{a \in C_5} \min\Big\{ d_a^Q, -\beta \Big\} y_a \leq \sum_{a \in C_1} d_a^Q - \sum_{a \in C_2} \tilde{r}_a^Q (1 - y_a) + \sum_{a \in C_3 \cup C_4} x_a^k \tag{7.18}$$

can be used to strengthen the LP-relaxation of MCFND. Please note that $\tilde{r}_a^Q = \max\Big\{ 0, d_a^Q + \beta \Big\}$ and $C_5 \subseteq \delta^+(S) \setminus C_1$. A visualization of the arc sets used in the previous definitions is presented in Figure 7.2.

Spanning Tree Inequalities When using undirected graphs, Günlük (1999) and Agarwal and Aneja (2017) showed that the activated arcs in the induced cut $\delta(S_1, S_2, \ldots, S_p)$ of a p-partition of the nodes must form a spanning tree, when there is flow transported between all node sub sets S_i (the demand graph must be connected). Hence, the inequality

$$\sum_{a \in \delta(S_1, S_2, \ldots, S_p)} y_a \geq p - 1 \tag{7.19}$$

holds for the graph and all its subgraphs. Although these inequalities can be separated easily, the authors note that they are dominated by p-partition knapsack inequalities. Furthermore, these inequalities will only have a noticeable impact on the lower bound if the flow is considerable smaller than the arc's capacity.

MIR inequalities There have been several publications in literature that used mixed-integer rounding (MIR) techniques to derive new valid inequalities from existing ones. One of the most comprehensive papers in regard to these techniques has been presented by Atamtürk and Günlük (2007).

c-strong inequalities Atamtürk and Rajan (2002) and Atamtürk and Günlük (2007) utilize c-strong inequalities to solve instances of the unsplittable and p-splittable MCFND, where flow is forced to travel along a single or at maximum p different paths. As these kinds of flow are not of interest in the following, for details the reader is referred to the original paper.

7.4.2 Separation Algorithms

Several of the aforementioned inequalities can be considered as standard techniques that have been used in many applications of the MCFND. Some of them have been used so successfully that they are implemented in out-of-the-box solvers today. Especially, strong linking, knapsack and cut-set based inequalities like cut-set cover or flow cover are available within these solvers. However, separating such inequalities can be done in different ways, and while out-of-the-box solvers can only rely on properties of the underlying LP or MIP, we can use our knowledge on structures of the originating problem domain. Nevertheless, the generic (highly optimized) implementations shipped with the out-of-the-box solvers should not be neglected. In Section 15.1, we measured the performance of these built-in cut separation algorithms and how they improve the overall solution process.

Additionally, we decided to implement separation algorithms for strong linking and four cut-set-based inequalities, namely knapsack, flow cut-set, cut-set cover and minimum cardinality. Since all of these inequalities were successfully applied in recent research projects (Hewitt, Nemhauser, and Savelsbergh 2009; Gendron and Larose 2014; Chouman, Crainic, and Gendron 2016; Chouman, Crainic, and Gendron 2018), we considered these to be a reasonable choice.

Whenever an out-of-the-box solver has solved the LP-relaxation of a node within the branch-and-bound tree to optimality, it continues with the separation of cuts. Afterwards, it alternates between solving the LP-relaxation and generating cuts, until no further cuts can be separated. Given is the current solution of the LP-relaxation $(\bar{x}, \bar{y})$. For the separation of strong linking inequalities, it is sufficient to iterate over all flow variables $\bar{x}_a^k$ and verify if the inequality

$$\bar{x}_a^k > d^k \bar{y}_a \qquad \exists k \in K, a \in A^k \tag{7.20}$$

holds. In that case, a strong linking inequality can be added to the LP to cut off the current LP solution. Please note that this inequality is only valid for commodities k without any arc-requirements or precedence constraints ($\Omega^k = \emptyset$ and $\phi^k(a) = \emptyset$ for all $a \in A^k$). Although the separation will be performed in each node of the branch-and-bound tree, the inequalities will be incorporated into the LP of the root node.

Separation of cut-set-based inequalities requires a more complex implementation. First, a cut-set $S \subset V$ needs to be defined, which turned out to be not a trivial task in our case. As activation variables y_a have been used only to model piecewise-linear concave cost functions on an arc (which occur seldomly), we expect that for most arcs $y_a = 1$ will be fixed a priori. Hence, finding a cut-set S that contains an arc a with $0 < \bar{y}_a < 1$ in its induced cut $\delta^+(S)$ can be hard. We implemented Algorithm 7.2 for that purpose.

This algorithm starts by iterating over all arcs $\tilde{a}$ that show a fractional value $\bar{y}_{\tilde{a}}$ in the current LP solution and searches backward from $tail(\tilde{a}) \in V$ until it finds a source node $\tilde{v} \in V$ with $b^k_{\tilde{v}} < 0$ for any $k \in K$. During backward search all visited nodes are collected and node $head(\tilde{a}) \in V$ will be skipped, which leads to a cut-set S_1 with $\tilde{a} \in \delta^+(S_1)$. Typically, $\tilde{v}$ will be part of a larger compound structure consisting of several logical nodes connected by high-capacity (or even uncapacitated) arcs. That avoids the separation of violated inequalities, if these high-capacity arcs are part of $\delta^+(S_1)$. Therefore, the algorithm continues by performing iterative forward and backward searches starting at all nodes contained in S_1 until the induced cut $\delta^+(S_2)$ consists only of "real" transport arcs. We denote the resulting cut-set as S_2. Obviously, $|S_1| \leq |S_2|$. Enforcing transport arcs within the induced cut furthermore increases the probability to have more than one arc a with fractional value $0 < \bar{y}_a < 1$ in that cut.

After being able to generate cut-sets, we started the implementation of algorithms to separate above cuts. Separation of knapsack inequalities is straightforward and should not require any explanation. Please note that the subset $Y_1 \subset \delta^+(S)$ of arcs with $y_a = 1$ fixed a priori will emerge in the inequality as a constant term of form

$$\sum_{a \in \delta^+(S) \setminus Y_1} u_a y_a \geq d(S, \bar{S}) - \sum_{a \in Y_1} u_a \tag{7.21}$$

and thus can be incorporated by adopting $d(S, \bar{S})$ accordingly and ignoring all arcs in Y_1. The same simplification has been used for cut-set cover and minimum cardinality cuts. The latter have been separated by sorting the remaining arcs $\delta^+(S) \setminus Y_1$ in non-increasing order of their capacity. For the separation of flow cut-set inequalities a set of commodities $Q \subseteq K$ and a partition $\{ C_1, C_2 \}$ of the cut-set $\delta^+(S)$ needs to be defined. We aimed at increasing the probability that any variable y_a violates that inequality and choose $C_1 = Y_1$, $C_2 = \delta^+(S) \setminus Y_1$ and the set Q to contain all commodities having a source node within S. Finally, for

Algorithm 7.2 Cut-set Detection

```
 1: procedure FindCutsets(ȳ, 𝒩 = (V, A, u, c, c̃, b))
 2:     𝒮 ← ∅                                              ▷ Set of cut-sets
 3:     Ã ← { a ∈ A | 0 < ȳ_a < 1 }                        ▷ Candidates for induced cut
 4:     for ã ∈ Ã do
 5:         S ← FindInitialCutset(𝒩, ã)
 6:         for a ∈ δ⁻(S) do
 7:             if IsNoTransportArc(a), tail(a) ≠ head(ã) then
 8:                 S ← S ∪ { tail(a) }
 9:             end if
10:         end for
11:         for a ∈ δ⁺(S) do
12:             if IsNoTransportArc(a), head(a) ≠ head(ã) then
13:                 S ← S ∪ { head(a) }
14:             end if
15:         end for
16:         𝒮 ← 𝒮 ∪ { S }
17:         Ã ← Ã \ { ã }
18:     end for
19:     return 𝒮
20: end procedure
21:
22: procedure FindInitialCutset(𝒩 = (V, A, u, c, c̃, b), ã)
23:     S ← ∅                                              ▷ New cut-set
24:     Q ← { tail(ã) }                                    ▷ Queue of pending nodes
25:     while Q ≠ ∅ do
26:         v ← SelectFirstNode(Q)
27:         Q ← Q \ { v }
28:         S ← S ∪ { v }
29:         if ∃k ∈ K, b(v, k) < 0 then                    ▷ If source node has been found
30:             break
31:         end if
32:         for a ∈ δ⁻(v) do
33:             if tail(a) ∉ S, tail(a) ≠ head(ã) then
34:                 Q ← Q ∪ { tail(a) }
35:             end if
36:         end for
37:     end while
38:     return S
39: end procedure
```

the separation of cut-set-cover inequalities defining a (minimal) cover $C \subseteq \delta^+(S)$ is necessary, however, Gu, Nemhauser, and Savelsbergh (1999) showed that this is an NP-hard problem. Hence, we decided to utilize a modified version of the heuristic approach presented by Gu, Nemhauser, and Savelsbergh (1998) (see Algorithm 7.3).

Algorithm 7.3 Find Minimal Cover

```
1:  procedure FindMinCover(ȳ, S, 𝒩 = (V, A, u, c, c̃, b))
2:      A_0 ← Sort({ a ∈ δ⁺(S) | ȳ_a = 0 })          ▷ Sorted set of closed arcs
3:      A_1 ← Sort({ a ∈ δ⁺(S) | ȳ_a = 1 })          ▷ Sorted set of opened arcs
4:      A_f ← Sort(δ⁺(S) \ A_0 ∪ A_1)                ▷ Sorted set of free arcs
5:      C ← ∅                                        ▷ Resulting minimal cover
6:      u_C̄ ← Σ_{a∈δ⁺(S)} u_a                        ▷ Non-cover capacity
7:
8:      FillUpCover(C, u_C̄, A_f)
9:      if u_C̄ ≥ d(S, S̄) then                        ▷ Fill up with opened arcs
10:         FillUpCover(C, u_C̄, A_1)
11:     end if
12:     if u_C̄ ≥ d(S, S̄) then                        ▷ Fill up with closed arcs
13:         FillUpCover(C, u_C̄, A_0)
14:     end if
15:
16:     for a ∈ C do                                 ▷ Ensure cover is minimal
17:         if u_C̄ + u_a < d(S, S̄) then
18:             C ← C \ { a }
19:         end if
20:     end for
21:     return C
22: end procedure
23:
24: procedure FillUpCover(C, u_C̄, A)
25:     for a ∈ A do
26:         C ← C ∪ { a }
27:         u_C̄ ← u_C̄ − u_a
28:         if u_C̄ < d(S, S̄) then
29:             break
30:         end if
31:     end for
32: end procedure
```

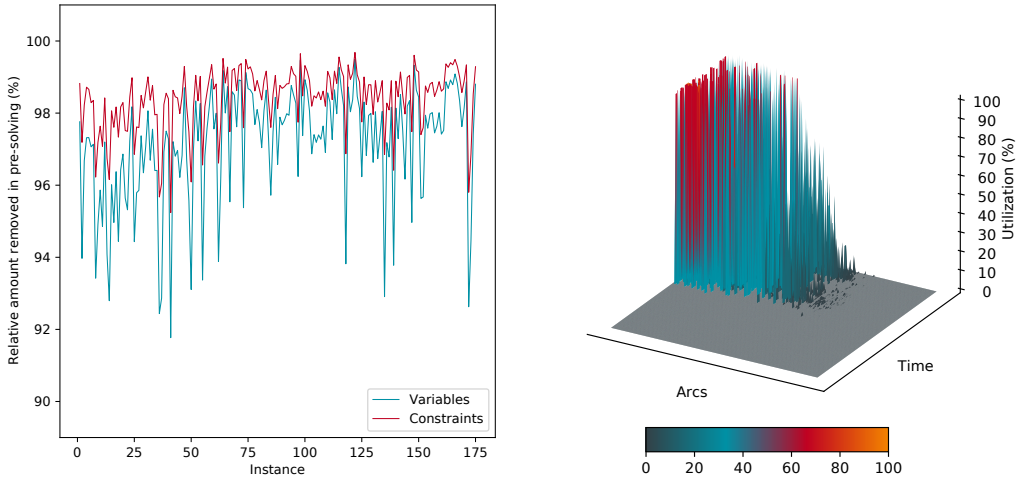
This algorithm first separates the arcs of the induced cut $\delta^+(S)$ into opened arcs A_1 and closed arcs A_0 depending on the solution of the LP-relaxation ($\bar{y}_a = 0$ or $\bar{y}_a = 1$). The remaining "free" arcs $a \in A_f = \delta^+(S) \setminus A_0 \cup A_1$ are sorted by decreasing values of $\bar{y}_a$, while ties are broken by decreasing order of u_a. Next, an initial cover C is created from the ordered arcs of A_f, followed by A_1 and A_0, by adding arcs to C until the capacity of the arcs which are not contained in the cover is smaller than the demand out of S ($u_{\bar{C}} < d(S, \bar{S})$). Finally, the algorithm removes all arcs from C which prevent that the cover is minimal.

Unfortunately, during our computational study we figured out that this approach is unable to separate a noticeable amount of violated inequalities. On most of our benchmark instances not a single inequality has been separated. We could not determine with certainty if this poor performance is related to our implementation or if these inequalities are just hardly applicable to our problem instances. Because the built-in separation of flow cover and cover inequalities of Gurobi showed poor performance, too (see Section 15.1), we assume that it is mainly related to the structure of our problem instances.

7.5 Interim Conclusion

Although, branch-and-cut is a standard technique that has been successfully used to solve many large-scale problems, it did not work out well on our benchmark instances for several reasons. First, modeling our problem domain using the arc-based formulation presented in Section 5.5 raised several difficulties as not all business constraints could be easily incorporated. We were forced to distinguish between commodities with and without arc-requirements or precedence constraints, which significantly increased the complexity of our model and implementation. This difference prevented a stronger aggregation of commodities, which raised the size of our MIP instances. Additionally, the validity of instance solutions depends on the generation of lazy constraints and increased the complexity even further.

Besides these modeling challenges, also the size of the problem instances raises questions. During our benchmarks, we used small to medium size instances (see Chapter 15), but already ran into out-of-memory issues. It is unrealistic to consider loading one of the real-life instances into computer memory. Furthermore, we observed that the majority of variables and constraints are not required for solving



(a) (b)

Figure 7.3: Charts showing (a) the relative amount of removed variables and constraints during pre-solving for small and medium size benchmark instances and (b) the typical utilization of the arcs (as height and colored) in the time-expanded graph when looking at an optimal solution (example shows solution to instance C5D5T-1).

one of the instances to optimality. During pre-solving Gurobi was able to noticeably decrease the size of the resulting MIPs (see Figure 7.3a). Additionally, within an optimal solution only a small fraction of arcs of the time-expanded graph are saturated or used at all (see Figure 7.3b). Such observations are typical for real-world transport networks and have been confirmed by other researchers (see Chardaire and Lisser 2002; Babonneau, du Merle, and Vial 2006).

Finally, the separation of cuts, either built-in or customized, did not support the solution process as we would have expected. Most of the medium size benchmark instances could not been solved to optimality. And cuts that have been used successfully for many applications of the MCFND showed a poor performance on instances of PCIMCFND. Obviously, there is a considerable gap between the capabilities of an arc-based formulation and what is required to solve one of our larger problem instances to optimality. For us, the presented branch-and-cut approach felt like the wrong tool to tackle our problem domain. Hence, we decided to switch over to a branch-and-price-and-cut approach, which will be presented in the next chapter.

8 | Branch-and-Price-and-Cut

In the previous chapter, we have presented a branch-and-cut approach to solve the arc-based formulation of our problem, that has been listed in MIP (5.10). However, besides some modeling issues, our computational study revealed that this approach is unable to solve real-world instances. While the poor performance of the cut separation algorithms led to slowly improving lower bounds, the major issue for larger instances has been the size of the MIPs. Most of the instances could not be loaded into computer memory. It might be possible to further improve this method and generate considerable smaller MIPs upfront. At least the used out-of-the-box solver was able to remove more than 90 % of the variables and constraints during pre-solving. Nevertheless, we expect that iteratively generating only these variables and constraints, that are required for an optimal solution, is a less complex approach and has been used by many researchers before (Gong 1996; Barnhart, Hane, and Vance 2000; Alvelos 2005; Gendron and Larose 2014; Rothenbächer, Drexl, and Irnich 2016). Typically, such an approach is implemented as a combination of branch-and-bound and column generation for solving the nodes LP-relaxation, and is called branch-and-price. If valid inequalities are created additionally, we speak of branch-and-price-and-cut. Such an approach will be presented in the remainder of this chapter.

8.1 Path-based Formulation

In the literature, there exist multiple extended formulations to tackle multi-commodity flow problems using column generation (or in our case using branch-and-price). However, Jones et al. (1993) showed that using a path-based formulation seems to be the most promising approach with regards to solution times. The same has been observed by Alvelos (2005). We will follow their advice in the following and show how an extended (path-based) formulation can be acquired from the arc-based

MIP (5.10) using the reformulation technique from Dantzig and Wolfe (1960). To simplify notation, first the MIP will be written in the standard form

$$\begin{aligned} \min\ & \boldsymbol{\gamma}^T \boldsymbol{x} && (8.1)\\ \text{s.t.}\ & \boxed{\mathcal{A}\boldsymbol{x} \geq \boldsymbol{\alpha}} \\ & \boxed{\mathcal{B}\boldsymbol{x} \geq \boldsymbol{\beta}} \\ & \boldsymbol{x} \in \mathbb{Z}^n_{\geq 0} \end{aligned}$$

This form can easily be achieved by reordering some constraints and moving terms from the left-hand side of the equations to the right hand-side or vice-versa.

$$\min \sum_{k \in K} \sum_{a \in A^k} c_a^k x_a^k + \sum_{a \in A} \tilde{c}_a y_a \tag{8.2a}$$

$$\text{s.t.} \quad u_a y_a - \sum_{k \in K : a \in A^k} x_a^k \geq 0 \qquad \forall a \in A \tag{8.2b}$$

$$\sum_{a \in \delta^+(v)} x_a^k - \sum_{a \in \delta^-(v)} x_a^k = b_v^k \qquad \forall k \in K, \forall v \in V^k \tag{8.2c}$$

$$\sum_{\tilde{a} \in R} x_{\tilde{a}}^k - x_a^k \geq 0 \qquad \forall k \in K, \forall a \in A^k, \forall R \in \phi^k(a) \tag{8.2d}$$

$$\sum_{a \in R} x_a^k \geq \sum_{v \in V_o^k} b_v^k \qquad \forall k \in K, \forall R \in \Omega^k \tag{8.2e}$$

$$(|\theta_a| + 1) \sum_{\tilde{a} \in \delta^+(\theta_a)} x_{\tilde{a}}^k - x_a^k \geq 0 \qquad \forall k \in K, \forall a \in A^k : |\theta_a| > 0 \tag{8.2f}$$

$$x_a^k \in \mathbb{Z}_{\geq 0} \qquad \forall k \in K, \forall a \in A^k \tag{8.2g}$$

$$y_a \in \{0, 1\} \qquad \forall a \in A \tag{8.2h}$$

Please note that we have added lazy constraints (5.11) to that MIP formulation. It is easy to see that the subsystem $\mathcal{A}\boldsymbol{x} \geq \boldsymbol{\alpha}$ of the standard form can be represented by Equation (8.2b) and the subsystem $\mathcal{B}\boldsymbol{x} \geq \boldsymbol{\beta}$ by Equations (8.2c) to (8.2f). Furthermore, the second set of constraints has a block diagonal structure and can be split into $|K|$ independent subsystems of form $\mathcal{B}^k \boldsymbol{x}^k \geq \boldsymbol{\beta}^k$, one for each commodity $k \in K$.

We define the discrete sets

$$X^k = \left\{ \boldsymbol{x} \in \mathbb{Z}_{\geq 0}^{|A^k|} \;\middle|\; \boldsymbol{x} \text{ is feasible for (8.2c) to (8.2f)}, 0 \leq x_a^k \leq u_a \right\}$$

which represent all feasible integer flows of commodity $k \in K$.

Using this definition, we can reformulate MIP (8.2) to

$$\begin{aligned} \min \sum_{k\in K}\sum_{a\in A^k} c_a^k x_a^k + \sum_{a\in A} \tilde{c}_a y_a & & (8.3)\\ \text{s.t. } u_a y_a - \sum_{k\in K: a\in A^k} x_a^k \geq 0 & \qquad \forall a \in A \\ (x_a^k)^{|A^k|} \in X^k & \qquad \forall k \in K \\ y_a \in \{0,1\} & \qquad \forall a \in A \end{aligned}$$

Following a *discretization* approach (Lübbecke and Desrosiers 2005), it is a well-known result that the sets X^k can be represented by

$$X^k = \left\{ \boldsymbol{x} \in \mathbb{Z}_{\geq 0}^{|A^k|} \;\middle|\; \boldsymbol{x} = \sum_{p\in P^k} \lambda_p^k \boldsymbol{x}_p^k, \sum_{p\in P^k} \lambda_p^k = 1, \lambda_p^k \in \mathbb{Z}_{\geq 0} \right\}$$

with P^k enumerating a finite set of integer points $\left\{ \boldsymbol{x}_p^k = (x_{ap}^k)^{|A^k|} \right\}_{p\in P^k} \subseteq X^k$ (see Wolsey and Nemhauser 1999, pp. 104 sq.). The finite set of integer rays, that has been used in the original theorem, can be ignored here, since all X^k are bounded. Using previous representation of X^k, MIP (8.2) is reformulated again to the *integer master problem*

$$\begin{aligned} \min \sum_{k\in K}\sum_{p\in P^k} c_p^k \lambda_p^k + \sum_{a\in A} \tilde{c}_a y_a & & (8.4)\\ \text{s.t. } u_a y_a - \sum_{k\in K: a\in A^k}\sum_{p\in P^k} \lambda_p^k x_{ap}^k \geq 0 & \qquad \forall a \in A \\ \sum_{p\in P^k} \lambda_p^k = 1 & \qquad \forall k \in K \\ \lambda_p^k \in \mathbb{Z}_{\geq 0} & \qquad \forall k \in K, \forall p \in P^k \\ y_a \in \{0,1\} & \qquad \forall a \in A \end{aligned}$$

with

$$c_p^k = \sum_{a\in A^k} c_a^k x_{ap}^k \qquad \forall k \in K, \forall p \in P^k.$$

This is the most general form of the integer master problem, that selects for each commodity k a feasible integer flow $\boldsymbol{x}_p^k$ from the set X^k by choosing $\lambda_p^k = 1$ for exactly one $p \in P^k$. Sufficient arc capacities are ensured by summing up these selected flows over all commodities. However, initially we were interested in a path-based formulation. Hence, in the following we assume that each commodity

is disaggregated to have a single origin node v_o^k and destination node v_d^k. Following the results presented in Section 7.3, each integer flow $\boldsymbol{x}$ can be composed of a set of paths (and cycles). Redefining P^k to enumerate all (v_o^k, v_d^k)-paths with $\alpha_{ap}^k \in \mathbb{Z}_{\geq 0}$ indicating how often arc $a \in A^k$ occurs within path $p \in P^k$, leads to a new representation of

$$X^k = \left\{ \boldsymbol{x} \in \mathbb{Z}_{\geq 0}^{|A^k|} \;\middle|\; \boldsymbol{x} = \sum_{p \in P^k} \boldsymbol{\alpha}_p^k \lambda_p^k, \sum_{p \in P^k} \lambda_p^k = b(v_d^k, k), \lambda_p^k \in \mathbb{Z}_{\geq 0} \right\}.$$

Here λ_p^k determines how many times path p is used by commodity k. Please note that this transformation of X^k has not been exact, as it ignores all cycles that are not connected to any (v_o^k, v_d^k)-path. Nevertheless, we are not interested in these cycles, and they will not change the objective value of our MIP, so we accept this inaccuracy. Using this new definition of X^k finally results in the desired path-based reformulation of MIP (8.2)

$$\begin{aligned}
\overset{*}{z} = \min & \sum_{k \in K} \sum_{p \in P^k} c_p^k \lambda_p^k + \sum_{a \in A} \tilde{c}_a y_a && && \text{(8.5a)} \\
\text{s.t.} & \sum_{k \in K : a \in A^k} \sum_{p \in P^k} \alpha_{ap}^k \lambda_p^k \leq u_a y_a && \forall a \in A && \text{(8.5b)} \\
& \sum_{p \in P^k} \lambda_p^k = b(v_d^k, k) && \forall k \in K && \text{(8.5c)} \\
& \lambda_p^k \in \mathbb{Z}_{\geq 0} && \forall k \in K, \forall p \in P^k && \text{(8.5d)} \\
& y_a \in \{0, 1\} && \forall a \in A && \text{(8.5e)}
\end{aligned}$$

with

$$c_p^k = \sum_{a \in A^k} c_a^k \alpha_{ap}^k \qquad \forall k \in K, \forall p \in P^k.$$

Since the set of paths P^k grows exponentially with the size of the graph, MIP (8.5) will do the same. Hence, generating all paths P^k and solving the LP-relaxation of the path-based formulation directly is not an option for larger instances and especially for time-expanded graphs. Instead, column generation will be used.

8.2 Column Generation

Column generation is a well-known technique to solve large LPs. It has been pioneered by the first papers of Dantzig and Wolfe (1960) and Gilmore and Gomory (1961), and has been used for a long time to solve the LP-relaxation of MIPs, especially those with exponential amount of variables. The central idea behind column generation is, to start with an LP that contains only a small subset of all variables and to generate more of them, if this is required to find an optimal solution. In our case, only a subset $\tilde{P}^k \subseteq P^k$ of a commodity's paths will be selected and the following LP, which is called the *reduced master problem*, can be derived from MIP (8.5) by relaxing its integrality constraints.

$$
\begin{aligned}
\tilde{z} = \min & \sum_{k \in K} \sum_{p \in \tilde{P}^k} c_p^k \lambda_p^k + \sum_{a \in A} \tilde{c}_a y_a && && (8.6)\\
\text{s.t.} & \sum_{k \in K : a \in A^k} \sum_{p \in \tilde{P}^k} \alpha_{ap}^k \lambda_p^k \leq u_a y_a && \forall a \in A && (\pi_1)\\
& \sum_{p \in \tilde{P}^k} \lambda_p^k \geq b(v_d^k, k) && \forall k \in K && (\pi_0)\\
& \lambda_p^k \in \mathbb{R}_{\geq 0} && \forall k \in K, \forall p \in \tilde{P}^k && \\
& y_a \in [0,1] && \forall a \in A && (\pi_2)
\end{aligned}
$$

Please note that we have changed the convexity constraint from a set partitioning to a set covering formulation, which is known to improve the convergence of LP solving and does not change the optimal objective value in our case (Lübbecke and Desrosiers 2005). Furthermore, in the rightmost column we show the dual variables that correspond to the primal constraints. Hence, the dual of the aforementioned reduced master problem has the form

$$
\begin{aligned}
\tilde{z} = \max & \sum_{k \in K} b(v_d^k, k) \pi_0^k - \sum_{a \in A} \pi_{2,a} && && (8.7)\\
\text{s.t.} & \ \pi_0^k - \sum_{a \in A^k} \alpha_{ap}^k \pi_{1,a} \leq c_p^k && \forall k \in K, \forall p \in \tilde{P}^k && (\lambda_p^k)\\
& \pi_0^k \in \mathbb{R}_{\geq 0} && \forall k \in K && \\
& u_a \pi_{1,a} - \pi_{2,a} \leq \tilde{c}_a && \forall a \in A && (y_a)\\
& \pi_{1,a}, \pi_{2,a} \in \mathbb{R}_{\geq 0} && \forall a \in A &&
\end{aligned}
$$

The *sub-problem* (or *pricing problem*), that must be solved for each commodity $k \in K$ to verify, if further variables must be added to the reduced master problem, is given in its standard form as

$$\begin{aligned} \overset{*}{x}^k = &\min \left(\boldsymbol{\gamma}^{k^T} - \boldsymbol{\pi}^T \mathcal{A}\right)\boldsymbol{x}^k - \pi_0^k \\ &\text{s.t. } \mathcal{B}^k \boldsymbol{x}^k \geq \boldsymbol{\beta}^k \\ &\boldsymbol{x}^k \in \mathbb{Z}^{n_k}_{\geq 0} \end{aligned} \tag{8.8}$$

with $\boldsymbol{\pi} = (\boldsymbol{\pi}_1, \boldsymbol{\pi}_2)^T$ being the vector of dual values, that has been acquired from the last (maybe suboptimal) solution of LP (8.6). As in our case the sub-problem is a shortest path problem, that can be solved efficiently using the algorithm presented in Chapter 6, we are not interested in solving MIP (8.8) directly, but only want to know the arc costs that must be used during path search. These can be derived from the objective function coefficients of variables $\boldsymbol{x}_a^k$ and are defined as

$$\dot{c}_a^k = c_a^k + \pi_{1,a} \qquad \forall k \in K, \forall a \in A^k. \tag{8.9}$$

These costs will be used as arc costs in our path search algorithm. If that search finds a (v_o^k, v_d^k)-path p with $\overset{*}{x}^k = \sum_{a \in p} \alpha_{ap}^k \dot{c}_a^k - \pi_0^k < 0$ for any commodity $k \in K$, a corresponding variable λ_p^k will be added to LP (8.6). Otherwise, that LP already has been solved to optimality. We will use dual value π_0^k during path search as an upper bound for the path costs in order to allow pruning by bounds, which has been explained in Section 6.4.

8.3 Column-and-Row Generation

One of the major drawbacks of the "classical" column generation method presented above is, that all constraints must be explicitly known upfront. Otherwise, there would be insufficient dual information to solve the pricing problem. We already know that for most instances of MIP (8.5) only a minority of the arcs will be used by any path in an optimal solution (see Figure 7.3b). However, we are forced to generate a capacity constraint in LP (8.6) for each of these arcs to gather the required dual information, even if there is no path using that arc. This noticeably increases the size of the LP.

In recent years several authors have published extensions to the column generation methodology, that relax the requirement to know all constraints in advance, but instead generate the (structural[1]) constraints if needed. These extensions are known in literature as *column-and-row generation* methods. The three most outstanding frameworks in that area are the ones from Muter, Birbil, and Bülbül (2013), Sadykov and Vanderbeck (2011), and Frangioni and Gendron (2013). We developed a similar approach to solve LP (8.6) without generating all capacity constraints upfront.

Assume that we are in an arbitrary pricing iteration and arc $\tilde{a} \in A$ is given, which is not contained in any path so far. Obviously, the reduced master problem will contain capacity constraint $0 \leq u_{\tilde{a}} y_{\tilde{a}}$ without any path variables λ_p^k in that case. To perform pricing, the corresponding dual value of variable $\pi_{1,\tilde{a}}$ is required. Looking at the dual LP (8.7), variable $\pi_{1,\tilde{a}}$ is affected only by constraint

$$u_{\tilde{a}} \pi_{1,\tilde{a}} - \pi_{2,\tilde{a}} \leq \tilde{c}_{\tilde{a}}.$$

It is trivial to see that in an optimal solution of the dual program there will be always $\pi_{2,\tilde{a}} = 0$, provided that arc $\tilde{a}$ is not contained in any path. This implies that the dual value $\pi_{1,\tilde{a}}$ can be chosen arbitrarily in the range

$$0 \leq \pi_{1,\tilde{a}} \leq \frac{\tilde{c}_{\tilde{a}}}{u_{\tilde{a}}}$$

without affecting the optimal solution value. Hence, we are able to gather all dual information required for pricing without explicitly creating all "empty" capacity constraints in LP (8.6). As long as arc $\tilde{a}$ is not used within any path, we assume the dual value $\pi_{1,\tilde{a}} = 0$, which implies for our path search algorithms that $\dot{c}_{\tilde{a}}^k = c_{\tilde{a}}^k$. We add the missing capacity constraint and variable $y_{\tilde{a}}$ on-demand, when arc $\tilde{a}$ is used in some path p for the first time and the corresponding path variable λ_p^k has been created during pricing. As a result, we keep the size of the LPs as small as possible.

[1] We are not talking about branch-and-price-and-cut methods that will generate redundant constraints to strengthen the LP-relaxation, but generating structural constraints of the LP-model.

8.4 Convergence and Feasibility

When discussing the convergence behavior of a column generation approach, typically the optimal solution value $\tilde{z}$ of the reduced master problem is compared to its Lagrangian bound $\mathcal{L}(\boldsymbol{\pi})$, which is a lower bound on the optimal solution value of the (non-reduced) master problem $\overset{*}{z}$. Following the explanation of Lübbecke and Desrosiers (2005), for a given vector $\boldsymbol{\pi}$ of dual values this bound is defined as

$$\mathcal{L}(\boldsymbol{\pi}) = \min_{\boldsymbol{x}} \boldsymbol{\gamma}^T \boldsymbol{x} - \boldsymbol{\pi}^T (\mathcal{A}\boldsymbol{x} - \boldsymbol{\alpha}) = \tilde{z} + \sum_{k \in K} \overset{*}{x}^k \leq \overset{*}{z} \leq \tilde{z}. \tag{8.10}$$

Hence, after a pricing iteration that solved the sub-problem of each commodity $k \in K$ to optimality (acquiring solution values $\overset{*}{x}^k$) a lower bound for the optimal LP solution value is given by Equation (8.10). In contrast, the optimal solution value $\tilde{z}$ of the reduced master problem forms an upper bound. Plotting these bounds after each pricing iteration leads to charts similar to the ones in Figure 8.1.

These charts show the major drivers of convergence issues, which have been observed regularly in applications of column generation. When using a dual simplex or barrier solver, the Lagrangian (lower) bound is oscillating heavily (yo-yo

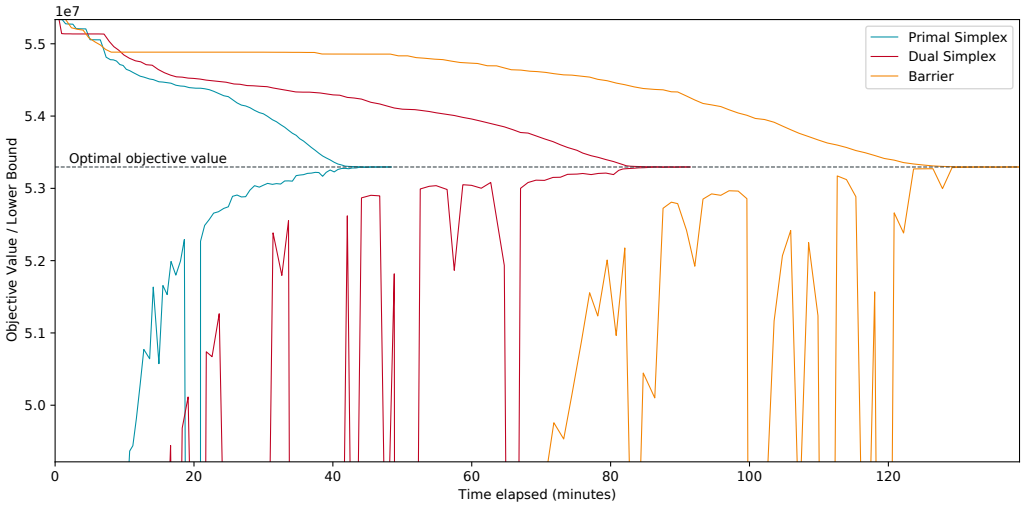


Figure 8.1: Charts showing the convergence of the column generation procedure when solving the root node LP of the MIP for benchmark instance C50D10TTL-1. The different charts show the convergence obtained when using different LP-solvers.

phenomenon according to Vanderbeck 2005). The root cause of this oscillation is the convergence of the dual solution values, which typically do not converge smoothly against the optimal solution but jump back and forth from iteration to iteration (bang-bang effect). The primal simplex solver shows such oscillations as well, however they do not seem to dominate the convergence of the lower bound. Another phenomenon observed for all three solvers is the poor lower bound at the beginning of the solution process (heading-in) and the slow convergence close to the optimal solution (tailing-off). As the generation of columns in the primal LP is equivalent to separating cuts in the dual, the latter can be explained by valid inequalities that cut-off only a small part of the dual solution space. Besides that, it has been observed that primal LPs tend to degenerate when using column generation, which implies slow convergence in the dual, too (Sadykov 2019).

There exists some evidence that the absence of oscillation is a desirable property that might heavily impact efficiency of the column generation method (Lübbecke and Desrosiers 2005). Hence, several stabilization techniques have been suggested in literature to cope with this issue. Following the notation of Vanderbeck (2005), these techniques can be grouped into three different families. First, oscillation can be limited by forcing the dual solutions to remain in the proximity of a *stability center*. Adding a penalty function to the dual objective is the primary approach to achieve this. Second, using *smoothing techniques*, the current dual solution can be corrected by combining it with previous dual solutions. This limits the change of the dual values to a certain extent and finally limits the bang-bang effect. The methods proposed by Wentges (1997) and Neame (2000) are two widely used examples. Third, it has been observed that convergence can be improved by separating dual solutions from the interior of the dual solution space rather than an extreme point (so called *centralized prices*). The analytic center cutting plane method (ACCPM) proposed by Goffin and Vial (2002) is a well known approach from that family.

We analyzed the convergence behavior for several benchmark instances when solving their root node LPs. For all instances the results looked similar to the charts presented in Figure 8.1. While the superiority of the primal simplex solver is obvious in our case, we wondered if its convergence behavior could be further improved by stabilization techniques. We followed the suggestion of Sadykov (2019) and focused our attention on smoothing. Although other approaches typically show better performance, they are harder to parameterize, especially when used in

conjunction with cutting and branching. We implemented the "smoothing with self-adjusting parameter" proposed by Pessoa et al. (2018) and compared that to the results gathered without stabilization. As we could not measure any impact on the convergence within preliminary testing, we decided to not further pursue any stabilization approach. These techniques might be examined in more detail in future research. Nevertheless, we decided to use only a primal simplex solver within all experiments, as this approach showed the best performance during our evaluation on all instances.

Column generation requires feasibility of the reduced master problem, otherwise, there will be no dual solution values that can be used for pricing. There are at least two different approaches to cope with this requirement: first, to apply Farkas pricing, which does not require a feasible dual solution, or second, to ensure that the reduced master problem is always feasible. We already introduced shortcut arcs in Section 5.6 that ensure feasibility of the reduced master problem and simplify finding an initial feasible solution. Alvelos (2005) used a similar approach within a branch-and-price application and compared it to relaxing the capacity constraints with slack variables, which clearly showed worse performance. Besides feasibility considerations, also the initialization of the column generation procedure needs to be taken into account. An initial set of columns must be provided to solve the reduced master problem for the first time, containing at least one path flow variable for each commodity. Such a set can easily be defined when using shortcut arcs. Hence, during our experiments we followed this approach and added a variable representing the path over the shortcut arc for each commodity. Furthermore, if feasible integer solutions have been determined in advance (e.g., by using a heuristic algorithm) we add all paths used in these solutions. Although an integer solution might not be a good starting point to find the optimal solution of the corresponding LP-relaxation (Vanderbeck 1994), we expect it to be better than any solution consisting only of shortcut arcs.

8.5 Removal of Columns

The previously described heading-in effect observed in many column generation applications (see Figure 8.1) indicates, that columns generated during the first iterations are likely of poor quality and their majority will not be used within an optimal solution. As a consequence, these columns might only increase the size

of the LP (and hence increase solution times), but do not provide any value in finding an optimal solution. Thus, removing these columns might speed up the overall solution process. In his branch-and-price approach Alvelos (2005) discussed different column removal approaches, ranging from never removing any columns to removing all, that are non-basic in the current solution. His results indicate that column removal should only be considered, if LP solving requires a noticeable amount of time. As we expect this to be true at least for our largest instances, we decided to use column removal within our approach.

We used SCIP (Achterberg 2009b) as framework to implement our branch-and-price-and-cut algorithm, which already offers a column aging mechanism. For each column the framework tracks, in which iteration it has been basic most recently and automatically removes the column, if it has been non-basic for a predefined amount of iterations. In contrast, the column will be re-incorporated from a column pool as soon as it shows negative reduced costs. Instead of implementing these mechanisms on our own, we decided to use the default implementation of SCIP. Nevertheless, we evaluated the impact of different configurations on the performance of our approach, which will be presented in Section 16.1.

Please note that we did not implement the removal of capacity constraints, in case all variables from that constraint have been removed. We assumed such situations to seldomly occur anyway and thus accepted the slightly larger LPs. As a result there might be "empty" capacity constraints within an LP (but not in the MIP). Similarly, we did not use row aging to remove any (structural) constraints from the LP. However, we used it to remove cuts that have become obsolete. We will explain this approach in detail in Section 8.7.

8.6 Accelerated Pricing

While the previous sections dealt with the efficiency of solving the reduced master problem, in the following we focus on the pricing sub-problem. Typically, that part of the column generation method will consume a considerable amount of time, hence speeding up might be worth the effort. Although, there are numerous heuristic approaches available to accelerate pricing (see Chapter 10), in the remainder of this chapter we will consider only methods that preserve the exact nature of our branch-and-price-and-cut procedure.

Since our sub-problem revealed a block diagonal structure (see Section 8.1), accelerating the pricing process is fairly simple in our case. Instead of solving each of the $|K|$ sub-problems sequentially, we can solve them in parallel. Even though a linear speed-up is unlikely in such computing scenarios, we expect it to be noticeable and definitely worthwhile to follow that approach. As there will be pricing problems that consume more computing time than others, distributing these problems to different CPU cores one-by-one allows for a better load balancing and overall utilization of computing resource. However, we figured out in our experiments that communication overhead between the different processing threads is remarkably high. Hence, we decided to distribute the sub-problems not one-by-one to the CPU cores but in small (random) chunks of ξ problems, resulting in $\left\lceil \frac{|K|}{\xi} \right\rceil$ chunks that must be distributed. Section 16.2 will report our computational results related to this strategy.

Although accelerating the pricing of columns by parallelization has been straightforward within our approach, avoiding to solve a sub-problem at all will be even more efficient. Due to the block diagonal structure mentioned before, we are in the advantageous position that we are able to simultaneously generate many columns within a single pricing iteration (one path flow variable per commodity). However, column generation does not require to solve all of them (*full pricing*) to guarantee an exact solution (except in the last pricing iteration). Furthermore, none of them must be solved exactly (we come back to that in Chapter 10). The only prerequisite that must be ensured is that at least one column with negative reduced costs is added, if there is any. That offers opportunities for further accelerating the pricing process by solving only a subset of all pricing problems (*partial pricing*). While this will obviously speed up a single pricing iteration, it might increase the overall amount of iterations necessary to find the optimal solution. Finding a proper balance between the time spent in a single iteration and the total amount of iterations is essential for the success of this approach.

We implemented Algorithm 8.1 for pricing a limited amount of ϱ_{min} variables in each iteration. That algorithm solves the first ϱ_{min} sub-problems by the parallelized approach presented above and adds all priced columns to the candidate set C. If not enough candidates have been found after the first round ($|C| < \varrho_{min}$) another round is started solving additional $\min\{\, \varrho_{min}, 2(\varrho_{min} - |C|) \,\}$ sub-problems in a round-robin fashion (*cyclic pricing*). That procedure continues until C is filled up with sufficiently many candidate columns ($|C| \geq \varrho_{min}$). Please note that pricing

Algorithm 8.1 Candidate Columns Pricing

```
1: procedure PriceCandidateColumns(K, ϱ_min, ξ, i_offset)
2:     C ← ∅                                   ▷ Resulting candidate columns
3:     i ← 0                                   ▷ Amount of priced sub-problems
4:     while i < |K|, |C| < ϱ_min do
5:         ϱ ← min{ ϱ_min, 2(ϱ_min − |C|), |K| − i }   ▷ Problems in current round
6:         C ← C ∪ DoParallelPricing(K, ξ, i_offset, ϱ)
7:         i ← i + ϱ
8:         i_offset ← i_offset + ϱ
9:     end while
10:    return (C, i_offset)
11: end procedure
```

will be terminated, if all $|K|$ sub-problems have been solved without finding any columns with negative reduced costs ($C = \emptyset$). To ensure that the first sub-problems within our list are not preferred over later ones, we store the last priced sub-problem (i_{offset}) and carry it over during all pricing iterations as the starting point for the round-robin approach.

After having priced the set of candidate columns C, the simplest next step would be to add all of them to the current LP. Nevertheless, the LP might grow quickly by using this simple approach, which caused us to evaluate different strategies for selecting a limited set of candidates $\tilde{C} \subseteq C$ that actually will be added. In a classical approach the candidate columns would be sorted by their reduced costs and $\tilde{C}$ would be filled with the χ "most negative" columns. However, that approach neglects, that during pricing all sub-problems reacted to the same dual solution values, which leads to paths that are likely using similar arcs. Adding all of these paths to the LP might be of less value if only a few of them can be used simultaneously in an optimal solution, due to the capacity constraints. Furthermore, it might be reasonable to explore other regions of the solution space by supporting the usage of arcs that have not been used before. Hence, we evaluated different *scoring* approaches to order the candidate columns by a measure different to the classical one. Within our pricing algorithm, set $\tilde{C}$ will be filled selecting $\zeta \cdot \chi$ candidates by their reduced costs ($0 \leq \zeta \leq 1$) and $(1 - \zeta) \cdot \chi$ candidates by one of the following scoring schemes.

Local arc utilization score Given is set $\tilde{P}_i^k$ containing all paths of commodity $k \in K$ for which corresponding columns have been added to the candidate set C

in the current i-th pricing iteration. We calculate arc flow $x^k_{a,i}$ for each arc $a \in A^k$ by sending a single unit of flow along each path $p \in \tilde{P}^k_i$. Please note that this flow might violate capacity constraints. The *maximum aggregated arc flow* is given by

$$\hat{x}_i = \max_{a \in A} \{ x_{a,i} \} = \max_{a \in A} \left\{ \sum_{k \in K} x^k_{a,i} \right\} = \max_{a \in A} \left\{ \sum_{k \in K} \sum_{p \in \tilde{P}^k_i} \alpha^k_{ap} \right\} \tag{8.11}$$

with $x_{a,i} = \sum_{k \in K} x^k_{a,i}$ being the *aggregated arc flow* of arc a. For a new candidate column $c \in C$ and its corresponding path $\tilde{p}_c = (a_1, a_2, \ldots, a_n)$ with length n, the *local arc utilization score* s_L is determined by formula

$$s_L(\tilde{p}_c) = \frac{1}{n} \sum_{a \in \tilde{p}_c} \hat{x}_i - x_{a,i}. \tag{8.12}$$

This score supports the selection of paths that contain arcs which seldom are used by other paths, which is kind of the "opposite" of what is achieved when selection is based on reduced costs. We expect that this scoring approach will imply a broader exploration of the solution space, which might improve the performance of column generation and the overall branch-and-price-and-cut. Please note that we scaled the score with factor $\frac{1}{n}$ to avoid that long paths are preferred over short paths. The candidate column $c = \text{argmax}_{c \in C} \{ s_L(\tilde{p}_c) \}$ is chosen to be added next to the reduced master problem.

Global arc utilization score Hence, the local arc utilization score is biased towards the arcs that have been priced in the current iteration, it might be a poor measure if a path really explores further areas of the solution space. There is a chance that in previous iterations similar paths have been priced already. Thus, instead of looking only at the set of paths $\tilde{P}^k_i$ found in the current pricing run, the *global arc utilization score* s_G utilizes all arcs $\tilde{P}^k$ that have been added to the reduced master so far. Consequently, the calculation of the maximum aggregated arc flow must be changed to

$$\hat{x}_i = \max_{a \in A} \left\{ \sum_{k \in K} \sum_{p \in \tilde{P}^k_i} \alpha^k_{ap} + \sum_{p \in \tilde{P}^k} \alpha^k_{ap} \right\}. \tag{8.13}$$

Weighted arc utilization score As the global arc utilization score looks at all paths that have been generated in any previous iteration, the score does not respond well to the situation given in the current iteration. As both scores presented above seem to have benefits, we wondered if they can be combined easily and defined the *weighted arc utilization score*

$$s_W(\tilde{p}_c) = \frac{s_L(\tilde{p}_c) + s_G(\tilde{p}_c)}{2} \tag{8.14}$$

as the average of both scores. Whether this weighted aggregation brings any benefits, will be answered in the computational study in Section 16.3.

Obviously, while adding further candidate columns using one of the above scores, these columns might influence the score of the remaining candidates. We compared an approach of ignoring that impact on the score to a re-scoring procedure, that updates the score of all remaining candidates when additional ς columns have been added to the reduced master problem. Computational results of that comparison as well as on the performance of the scoring approach in general can be found in Sections 16.3 and 16.4. These results further contain measurements using a random selection of candidate columns as a baseline. Please note that in all approaches the candidates $C \setminus \tilde{C}$ that have not been selected at all, will be added to a column pool for being re-priced quickly in an upcoming pricing iteration.

Even though column generation is intended for solving an LP to optimality, within a branch-and-bound scheme it might be beneficial to terminate pricing new columns without having an optimal solution. Looking at the convergence of our column generation method in Figure 8.1 again, it is obvious that due to the tailing-off effect, a substantial amount of processing time is spent in closing the last few percent of the optimality gap. However, that effort is not required to ensure that finally an optimal integer solution is found. Solving the LPs for each node of the branch-and-bound tree only delivers bounds that might help pruning sub-trees. Hence, "optimal" bounds support keeping the size of the tree small, which often implies an overall speed-up of the process. However, it might be sufficient to have just "good" bounds as long as these can be calculated considerably faster and do not lead to a noticeable larger tree. *Early branching* is a well-known technique used within branch-and-price methods to stop pricing additional columns as soon as a predefined criterion is meet (Desaulniers, Desrosiers, and Solomon 2002; Lübbecke and Desrosiers 2005). Typically, such a criterion looks at the remaining gap or

monitors the progress of the LP solution value to detect when tailing-off occurs. We decided to implement the first variant that uses a predefined gap limit as criterion. However, this approach requires to calculate a Lagrangian bound for determining the gap and using it as lower bound in the branch-and-bound scheme, if column generation has been terminated early. Obviously, that does not fit into the partial pricing procedure presented above, since a Lagrangian bound can only be calculated, if all sub-problems have been solved to optimality. Therefore, we modified our approach to regularly perform a full pricing iteration, determine the Lagrangian bound and perform early branching, if the gap limit has been reached. Section 16.6 explains our computational results regarding this strategy.

8.7 Generating Cuts

In the course of the branch-and-cut approach presented in Chapter 7, we explained several valid inequalities that could be used to strengthen the LP-relaxation of PCIMCFND instances (see Section 7.4). Although most of them showed poor performance on the arc-based formulation, they might be useful to speed up solving the LP-relaxation with column generation. Furthermore, they might lead to improved lower bounds within the branch-and-bound procedure.

While adding separated cuts to an instance is rather simple when using an out-of-the-box solver, that is not always the case within a branch-and-price framework. We need to distinguish between cuts that involve activation variables y_a only and those that contain flow variables x_a^k (or λ_p^k). The first can just be added to the reduced master problem. However, the later might have an impact on the pricing problem and need to be considered there. In the following, we will discuss valid inequalities of both categories and explain their impact on the pricing problem.

8.7.1 Valid Inequalities

Strong linking / forcing inequalities Strong linking inequalities performed worst among all cuts evaluated within our branch-and-cut approach. Nevertheless, they have been used successfully in many applications even in context of column generation (see Gendron and Larose 2014; Rothenbächer, Drexl, and Irnich 2016), and separation and implementation are fairly simple. Hence, we assumed that it is worth the effort to evaluate their performance within our branch-and-price-and-cut approach.

The arc-based separation algorithm for that kind of cut seeks for violated inequalities of form $x_a^k \leq d^k y_a$ with $d^k = \sum_{v \in V : b_v^k > 0} b_v^k$. Transforming these inequalities to our path-based formulation leads to constraints

$$\sum_{p \in P^k} \alpha_{ap}^k \lambda_p^k \leq d^k y_a \qquad \text{with} \quad d^k = b(v_d^k, k) \qquad \forall k \in K, a \in A^k \tag{8.15}$$

that can be added to MIP (8.5). Of course, the limitation stated early, that these cuts are only valid as long as it is guaranteed that each commodity path uses arc a only once, still holds. However, what has been a noticeable restriction in the branch-and-cut case, is by far not that complicating when using a path-based formulation. Here, all paths and the cardinality of their used arcs are known upfront and can be considered when separating such cuts. Above inequality is valid, if $\alpha_{ap}^k \in \{\, 0, 1 \,\}$ holds for all $p \in P^k$. Nevertheless, if there exists any p_1 with $\alpha_{ap_1}^k > 1$ and an optimal solution is given with $\lambda_{p_1}^k = d^k$, that solution would violate the inequality above, but would be valid for

$$\sum_{p \in P^k} \alpha_{ap}^k \lambda_p^k \leq \hat{\alpha}_a^k d^k y_a \qquad \text{with} \quad \hat{\alpha}_a^k = \max_{p \in P^k} \Big\{ \alpha_{ap}^k \Big\} \qquad \forall k \in K, a \in A^k. \tag{8.16}$$

Obviously, we will separate such (weaker) inequalities only if $\hat{\alpha}_a^k d^k < u_a$. Since in a column generation approach not all paths $p \in P^k$ of commodity $k \in K$ are known when constraint (8.16) is incorporated into the MIP, it might be necessary to adopt $\hat{\alpha}_a^k$ whenever further paths are priced and their flow variables λ_p^k are added to the constraint.

Strong linking inequalities contain flow variables and hence will have an impact on the pricing sub-problem. However, they do not change the sub-problem's structure, but affect only the arc costs used during path search. For each constraint (8.16)

that has been added to the reduced master problem, there will be an additional variable $\pi^k_{3,a} \in \mathbb{R}_{\geq 0}$ within its dual. It is trivial to see that the arc costs within the pricing sub-problem are determined by

$$\dot{c}^k_a = c^k_a + \pi_{1,a} + \pi^k_{3,a} \qquad \forall k \in K, \forall a \in A^k. \tag{8.17}$$

Knapsack inequalities Partitioning the set of nodes V into subsets S and $\bar{S}$ and defining the induced cut $\delta^+(S)$ of S, leads to the definition of the knapsack inequalities that have been presented in Section 7.4

$$\sum_{a \in \delta^+(S)} u_a y_a \geq d(S, \bar{S}) \qquad \forall S \subset V, S \neq \emptyset. \tag{8.18}$$

Again, $d(S, \bar{S})$ is the amount of commodity flow originating in S and being destined for a node in $\bar{S}$. Since these constraints do not contain any flow variables they will not impact the pricing and can just be added to the reduced master problem. We used the same separation algorithm that has been introduced in Section 7.4.2. However, there is one major difference to the separation done within our branch-and-cut approach. Additional activation variables y_a might be created within our column-and-row generation procedure after the corresponding knapsack inequalities have been added to the reduced master problem. Hence, whenever such variables are added to the LP, it must be ensured that all affected knapsack inequalities are updated, too.

Minimum cardinality inequalities Similar to knapsack inequalities, it is possible to add minimum cardinality inequalities

$$\sum_{a \in \delta^+(S)} y_a \geq \max \left\{ h \;\middle|\; \sum_{i=1,\ldots,h} u_{a_{(i)}} < d(S, \bar{S}) \right\} + 1 \tag{8.19}$$

to the reduced master problem without impacting the pricing problem. However, we faced the same challenge, that the constraint must be updated whenever additional activation variables y_a are incorporated. Here, it is not sufficient to just add the variables to the constraint, but updating the right-hand side might be required, too.

Cut-set cover inequalities The former definition of a minimal cover $C \subseteq \delta^+(S)$ (see Section 7.4) allows us to separate cut-set cover inequalities

$$\sum_{a \in C} y_a \geq 1. \tag{8.20}$$

Again, they will not have any impact on the pricing problem and we will reuse the separation procedure that has been presented in Chapter 7. Nevertheless, a minor change to Algorithm 7.3 is required to ensure that it will find a minimal cover. When categorizing the arcs of the cut-set into opened, closed and free arcs, all arcs without an activation variable y_a need to be considered as closed arcs, although they will not occur within the inequalities until the corresponding variable y_a has been created.

Flow cut-set inequalities For a given set $Q \subseteq K$ of commodities and a partition $\{C_1, C_2\}$ of cut $\delta^+(S)$, within our branch-and-cut approach, we separated flow cut-set inequalities

$$\sum_{k \in Q} \sum_{a \in C_1} x_a^k + \sum_{a \in C_2} u_a y_a \geq d_Q(S, \bar{S}) \tag{8.21}$$

where $d_Q(S, \bar{S})$ is a lower bound on the demand of commodities Q that must be transported from S to $\bar{S}$. Transforming these inequalities to the path-based formulations leads to constraints

$$\sum_{k \in Q} \sum_{a \in C_1} \sum_{p \in P^k} \alpha_{ap}^k \lambda_p^k + \sum_{a \in C_2} u_a y_a \geq d_Q(S, \bar{S}). \tag{8.22}$$

We will use the same procedure that has been described in Section 7.4.2 for separation of these inequalities. Hence, the partition of cut $\delta^+(S)$ is chosen to be $C_1 = Y_1$, $C_2 = \delta^+(S) \setminus Y_1$ with $Y_1 \subset \delta^+(S)$ being the subset of arcs with $y_a = 1$ fixed a priori while Q contains all commodities having its source node within S. Again, it must be ensured that constraints (8.22) are updated, if related path flow variables λ_p^k or activation variables y_a are priced. Similar to strong linking inequalities, these constraints contain path flow variables and thus will have an impact on the arc costs within the pricing sub-problem. We assume an additional dual variable $\pi_{3,S} \in \mathbb{R}_{\geq 0}$ for each flow cut-set inequality that has been added to the LP. That implies arc costs in the pricing sub-problem of

$$\dot{c}_a^k = c_a^k + \pi_{1,a} - \pi_{3,S} \qquad \forall k \in Q, \forall a \in C_1. \tag{8.23}$$

8.7.2 Price-and-Cut Loop

Combining column generation with cut separation requires a well defined interaction between the pricing and the separation step, since both might change the currently optimal LP solution and its corresponding dual values. As we used SCIP (Achterberg 2009b) to implement our branch-and-price-and-cut approach, that framework determines this interaction for us. LP solving starts with a predefined set of variables and constraints and solves the reduced master problem to optimality. The corresponding LP solution and its dual values are used to call the variable pricer, that might add additional variables and, in our case of column-and-row generation, also additional capacity constraints. Afterwards, the extended reduced master problem is solved again, repetitively until no further variables and constraints can be found by the pricer. At this point the solving process switches to the cut separation step and tries to generate multiple cuts from the different separation algorithms using the current optimal LP solution. If additional cuts have been added to the reduced master problem, the current solution should have become invalid and the LP needs to be resolved again. Afterwards, another call to the variable pricer occurs and the overall solving process starts from the beginning. That loop continues until the reduced master problem does not change for an iteration of pricing and cut separation and the algorithm terminates with an optimal solution to the (non-reduced) master problem.

As there might be a considerable amount of price-and-cut iterations before an optimal LP solution is found, it could be a reasonable approach to stop separation, even if further cuts can be found. This offers the opportunity to start with branching, if the lower bound is improving hardly. Typically, this is achieved by setting a limit on the amount of separation iterations per node of the branch-and-bound tree. SCIP already implemented such strategy by using the parameter `separating/maxrounds`. We evaluated that approach within our computational study (Section 16.5).

Besides limiting the amount of iterations, it might be desirable to also limit the amount of cuts that are added in each of these iterations. Otherwise, the LP might grow quickly, which will increase computing times of the simplex algorithm. SCIP tries to estimate the impact of a cut upfront, by calculating a score from the cut's efficacy, its orthogonality to other cuts and its parallelism to the objective function. Only a limited amount of the cuts with the highest score will actually enter the

LP. For more details on that procedure, the reader is referred to the dissertation of Achterberg (2009a, pp.48).

Finally, all cuts previously generated might become obsolete, since stronger cuts might have been added or additionally priced variables have been incorporated into the original separated cut. So after some iterations of pricing and cut separation, there might be a noticeable amount of cuts without any value for finding the optimal solution. However, these cuts increase the size of the LP and thus influence performance. Fortunately, SCIP keeps track of the amount of LP solving iterations in which a cut has been identified as redundant (SCIP calls these cuts inactive) and offers a row aging mechanism that removes such obsolete cuts as soon as the amount reaches a certain threshold. In our computational study, we examined, if row aging can speed up the solving process for our benchmark instances (Section 16.5).

8.8 Branching Rules

In the previous sections, we were only interested in solving the LP-relaxation of MIP (8.5) by column generation and cut separation. Now we are going to solve the MIP itself, using branch-and-price-and-cut. One of the most important ingredients of such a procedure, are the rules that are used to perform branching. When solving LP (8.6) to optimality, there might be variables λ_p^k or y_a with a fractional solution value. By branching on these variables, we can achieve integrality. We follow the suggestion of Lübbecke and Desrosiers (2005) and Rothenbächer, Drexl, and Irnich (2016) and will impose branching decisions first, that might have the highest impact on partitioning the solution space. So we will always branch on variables y_a first, since these determine if an arc of the network can be used at all, which might impact many commodities and even can lead to infeasibility. If all values of variables y_a are integral, we will continue with branching on the flow of arcs. This technique has turned out to be beneficial for multi-commodity flow problems in the work of Alvelos (2005). Even though all arc flow values are integral, it might happen that the corresponding values of the path flow variables λ_p^k are not (e.g., due to paths with identical costs). To ensure an integral solution of MIP (8.5) we will branch on λ_p^k last. In the following sections, we will analyze in more detail each kind of these branching decision.

8.8.1 Branching on Activation Variables

When branching on the (binary) arc activation variables y_a, there are only two cases that need to be considered: $y_a = 0$ or $y_a = 1$. Looking at an arbitrary node ν in the branch-and-bound tree, there might be several branching decisions that have been taken from the root of the tree to this node. Lets define $\mathcal{L}_y^\nu \subseteq A$ as the set of arcs $a \in A$ for which variable y_a has been branched to $y_a = 0$. Similarly, the set $\mathcal{U}_y^\nu \subseteq A$ contains all arcs $a \in A$ with the corresponding branching decision $y_a = 1$. With $\bar{A}^\nu = A \setminus (\mathcal{L}_y^\nu \cup \mathcal{U}_y^\nu)$ being the set of all arcs without any branching decisions regarding y_a, the reduced master problem in node ν will look like

$$
\begin{aligned}
\min \sum_{k \in K} \sum_{p \in \tilde{P}^k} c_p^k \lambda_p^k + \sum_{a \in \bar{A}^\nu} \tilde{c}_a y_a & & & \\
\text{s.t.} \sum_{k \in K: a \in A^k} \sum_{p \in \tilde{P}^k} \alpha_{ap}^k \lambda_p^k \leq u_a y_a & & \forall a \in \bar{A}^\nu & \quad (\pi_1) \\
\sum_{k \in K: a \in A^k} \sum_{p \in \tilde{P}^k} \alpha_{ap}^k \lambda_p^k \leq u_a & & \forall a \in \mathcal{U}_y^\nu & \quad (\pi_1) \\
\sum_{k \in K: a \in A^k} \sum_{p \in \tilde{P}^k} \alpha_{ap}^k \lambda_p^k \leq 0 & & \forall a \in \mathcal{L}_y^\nu & \quad (\pi_1) \\
\sum_{p \in \tilde{P}^k} \lambda_p^k \geq b(v_d^k, k) & & \forall k \in K & \quad (\pi_0) \\
\lambda_p^k \in \mathbb{R}_{\geq 0} & \qquad \forall k \in K, \forall p \in \tilde{P}^k & & \\
y_a \in [0,1] & & \forall a \in \bar{A}^\nu & \quad (\pi_2)
\end{aligned}
$$

Please note that the constant terms occurring in the objective function for all arcs with $y_a = 1$ can be neglected. We can derive the dual linear program of this LP as

$$
\begin{aligned}
\max \sum_{k \in K} b(v_d^k, k) \pi_0^k - \sum_{a \in \mathcal{U}_y^\nu} u_a \pi_{1,a} - \sum_{a \in \bar{A}^\nu} \pi_{2,a} & & & \\
\text{s.t. } \pi_0^k - \sum_{a \in A^k} \alpha_{ap}^k \pi_{1,a} \leq c_p^k & \qquad \forall k \in K, \forall p \in \tilde{P}^k & & \quad (\lambda_p^k) \\
u_a \pi_{1,a} - \pi_{2,a} \leq \tilde{c}_a & & \forall a \in \bar{A}^\nu & \quad (y_a) \\
\pi_0^k \in \mathbb{R}_{\geq 0} & & \forall k \in K & \\
\pi_{1,a} \in \mathbb{R}_{\geq 0} & & \forall a \in A & \\
\pi_{2,a} \in \mathbb{R}_{\geq 0} & & \forall a \in \bar{A}^\nu &
\end{aligned}
$$

What has changed compared to the original LP, due to the branching decisions that have been taken? All arcs $a \in \mathcal{L}_y^\nu$ cannot transport any flow and all paths using one of these arcs are forced to transport no flow either ($\lambda_p^k = 0$). Hence, these arcs can safely be ignored during pricing of new paths and already generated λ_p^k variables can be removed (only in the local node, not in the root of the branch-and-bound tree). In contrast, looking at the arcs $a \in \mathcal{U}_y^\nu$, the branching decisions do not have any impact on the reduced costs and Equation (8.9) is still valid.

8.8.2 Branching on Arc Flow

Branching on the flow of an arc $a \in A$ is not directly related to a variable of MIP (8.5), but to a set of paths which use this arc (accordingly a set of variables λ_p^k). Following the Dantzig-Wolfe reformulation in Section 8.1 the arc flow x_a is defined as

$$x_a = \sum_{k \in K} \sum_{p \in P^k} \alpha_{ap}^k \lambda_p^k \qquad \forall a \in A. \tag{8.24}$$

Given a solution to LP (8.6) and the (fractional) solution value $\bar{x}_a$ for arc $a \in A$, branching on the arc flow x_a requires to add constraints of form

$$x_a \leq \lfloor \bar{x}_a \rfloor \quad \Longleftrightarrow \quad \sum_{k \in K} \sum_{p \in P^k} \alpha_{ap}^k \lambda_p^k \leq \lfloor \bar{x}_a \rfloor \tag{8.25a}$$

$$x_a \geq \lceil \bar{x}_a \rceil \quad \Longleftrightarrow \quad \sum_{k \in K} \sum_{p \in P^k} \alpha_{ap}^k \lambda_p^k \geq \lceil \bar{x}_a \rceil \tag{8.25b}$$

to the children of the current node. Let's define the set $\mathcal{L}_x^\nu$, that contains tuples $(a, \bar{x}_a)$ representing branching decision of form (8.25a), while $\mathcal{U}_x^\nu$ will contain branching decisions related to (8.25b).

The reduced master problem can be extended to

$$
\begin{aligned}
\min \sum_{k\in K}\sum_{p\in\tilde{P}^k} c_p^k\lambda_p^k + \sum_{a\in A}\tilde{c}_a y_a & & & \\
\text{s.t.} \sum_{k\in K:a\in A^k}\sum_{p\in\tilde{P}^k} \alpha_{ap}^k\lambda_p^k \le u_a y_a & & \forall a\in A & \quad (\pi_1)\\
\sum_{k\in K:a\in A^k}\sum_{p\in\tilde{P}^k} \alpha_{ap}^k\lambda_p^k \le \lfloor \bar{x}_a \rfloor & & \forall (a,\bar{x}_a)\in\mathcal{L}_x^\nu & \quad (\pi_3)\\
\sum_{k\in K:a\in A^k}\sum_{p\in\tilde{P}^k} \alpha_{ap}^k\lambda_p^k \ge \lceil \bar{x}_a \rceil & & \forall (a,\bar{x}_a)\in\mathcal{U}_x^\nu & \quad (\pi_4)\\
\sum_{p\in\tilde{P}^k} \lambda_p^k \ge b(v_d^k,k) & & \forall k\in K & \quad (\pi_0)\\
\lambda_p^k \in \mathbb{R}_{\ge 0} & & \forall k\in K, \forall p\in\tilde{P}^k & \\
y_a \in [0,1] & & \forall a\in A & \quad (\pi_2)
\end{aligned}
$$

These additional constraints will have an impact on the form of the dual linear program

$$
\begin{aligned}
\max \sum_{k\in K} b(v_d^k,k)\pi_0^k - \sum_{a\in A}\pi_{2,a} - \sum_{(a,\bar{x}_a)\in\mathcal{L}_x^\nu}\lfloor\bar{x}_a\rfloor\pi_{3,a\bar{x}_a} + \sum_{(a,\bar{x}_a)\in\mathcal{U}_x^\nu}\lceil\bar{x}_a\rceil\pi_{4,a\bar{x}_a} & & & \\
\text{s.t. } \pi_0^k - \sum_{a\in A^k}\alpha_{ap}^k\pi_{1,a} - \sum_{(a,\bar{x}_a)\in\mathcal{L}_x^\nu}\alpha_{ap}^k\pi_{3,a\bar{x}_a} & & & \\
\quad + \sum_{(a,\bar{x}_a)\in\mathcal{U}_x^\nu}\alpha_{ap}^k\pi_{4,a\bar{x}_a} \le c_p^k & & \forall k\in K,\forall p\in\tilde{P}^k & \quad (\lambda_p^k)\\
u_a\pi_{1,a} - \pi_{2,a} \le \tilde{c}_a & & \forall a\in A & \quad (y_a)\\
\pi_0^k \in \mathbb{R}_{\ge 0} & & \forall k\in K & \\
\pi_{1,a},\pi_{2,a} \in \mathbb{R}_{\ge 0} & & \forall a\in A & \\
\pi_{3,a\bar{x}_a} \in \mathbb{R}_{\ge 0} & & \forall (a,\bar{x}_a)\in\mathcal{L}_x^\nu & \\
\pi_{4,a\bar{x}_a} \in \mathbb{R}_{\ge 0} & & \forall (a,\bar{x}_a)\in\mathcal{U}_x^\nu &
\end{aligned}
$$

Consequently, the calculation of the reduced costs of the arcs will change to

$$
\dot{c}_a^k = c_a^k + \pi_{1,a} + \sum_{(a,\bar{x}_a)\in\mathcal{L}_x^\nu}\pi_{3,a\bar{x}_a} - \sum_{(a,\bar{x}_a)\in\mathcal{U}_x^\nu}\pi_{4,a\bar{x}_a} \qquad \forall k\in K,\forall a\in A^k. \tag{8.26}
$$

Please note, until now $\dot{c}_a^k$ has always been non-negative (except when separating flow cut-set inequalities). For $\mathcal{U}_x^\nu \neq \emptyset$ there might be arcs with negative costs in the pricing problem which must be handled by the path search algorithms.

8.8.3 Branching on Path Flow Variables

The last branching decision that should be analyzed covers path flow variables λ_p^k. Given a fractional variable value $\bar{\lambda}_p^k$ from the solution of the reduced master problem, we define the sets $\mathcal{L}_\lambda^\nu$ and $\mathcal{U}_\lambda^\nu$ containing tuples of form $(k, p, \bar{\lambda}_p^k)$ that represent branching decisions $\lambda_p^k \leq \left\lfloor \bar{\lambda}_p^k \right\rfloor$ and $\lambda_p^k \geq \left\lceil \bar{\lambda}_p^k \right\rceil$. Including these constraints into LP (8.6) leads to a redefined reduced master problem

$$
\begin{aligned}
\min \sum_{k \in K} \sum_{p \in \tilde{P}^k} c_p^k \lambda_p^k + \sum_{a \in A} \tilde{c}_a y_a & & & \\
\text{s.t.} \sum_{k \in K : a \in A^k} \sum_{p \in \tilde{P}^k} \alpha_{ap}^k \lambda_p^k \leq u_a y_a & & \forall a \in A & \quad (\pi_1) \\
\lambda_p^k \leq \left\lfloor \bar{\lambda}_p^k \right\rfloor & & \forall (k, p, \bar{\lambda}_p^k) \in \mathcal{L}_\lambda^\nu & \quad (\pi_3) \\
\lambda_p^k \geq \left\lceil \bar{\lambda}_p^k \right\rceil & & \forall (k, p, \bar{\lambda}_p^k) \in \mathcal{U}_\lambda^\nu & \quad (\pi_4) \\
\sum_{p \in \tilde{P}^k} \lambda_p^k \geq b(v_d^k, k) & & \forall k \in K & \quad (\pi_0) \\
\lambda_p^k \in \mathbb{R}_{\geq 0} & & \forall k \in K, \forall p \in \tilde{P}^k & \\
y_a \in [0, 1] & & \forall a \in A & \quad (\pi_2)
\end{aligned}
$$

Obviously, it is not trivial to derive the impact on reduced costs and pricing implied by these branching decisions. There is no straightforward transformation to express the same branching decision in original arc flow variables x_a^k. However, this is required to include the branching decisions in sub-problem (8.8). Nevertheless, recall that branching on path variables will only be used, if there are no remaining fractional activation variables y_a and arc flow values x_a. In this way, the current LP solution will already offer an optimal integer multi-commodity flow that could be transformed to path flows using the algorithm presented in Section 7.3. However, it is possible that the arc flow is integral but the corresponding path flows are not, e.g., if there are paths with equal costs. As such a situation is rather unlikely to happen, we will rely on the LP solver to resolve it by a simple rounding heuristic or by branching. Such branching on path flow variables will not change the current optimal solution value, but only ensure integrality. Hence, there can be no further λ_p^k with negative reduced costs and it is not required to perform further pricing iterations. Consequently, determining the impact of these branching decisions on the pricing problem is not required at all, we just stop pricing of further variables in the corresponding sub-tree.

8.8.4 Combining all Branching Decisions

When traversing the branch-and-bound tree from its root to a certain child node ν, there might be different branching decisions that have been taken to reach this node. We have shown in the previous sections that all branching decisions only impact the arc costs used in the sub-problem but do not change its structure of a resource-constraint shortest path problem. Of course, there exists the most general case that all different types of decisions have been taken for a node ν. It is possible to combine the impact of each decision to an aggregated calculation formula for the modified arc costs used by commodity $k \in K$ during path search:

$$\dot{c}_a^k = \begin{cases} \infty & \forall a \in A^k \cap \mathcal{L}_y^\nu \\ c_a^k + \pi_{1,a} + \sum\limits_{(a,\bar{x}_a) \in \mathcal{L}_x^\nu} \pi_{3,a\bar{x}_a} - \sum\limits_{(a,\bar{x}_a) \in \mathcal{U}_x^\nu} \pi_{4,a\bar{x}_a} & \forall a \in A^k \setminus \mathcal{L}_y^\nu \end{cases} \tag{8.27}$$

This formula gives some hints about the first steps of our pricing algorithm. The sets $\mathcal{L}_y^\nu$, $\mathcal{L}_x^\nu$ and $\mathcal{U}_x^\nu$ must be defined and all dual variable values must be calculated (or just requested from the LP-solver). Furthermore, the costs of each arc must be updated using this formula before doing a path search on the resulting network.

8.9 Branching Rule and Node Selection

We have already mentioned that we follow a straightforward approach when selecting the next branching rule that should be applied at the current node: always use the rule that has the highest impact on partitioning the solution space. Hence, we have explained that we prefer branching on activation variables y_a over branching on arc flow variables x_a over branching on path flow variables λ_p^k. However, there might be many fractional variables of the same type (activation, arc flow or path flow) and it is required to choose a single one of them. We have tried to apply the same approach of looking at the impact of the corresponding branching decisions when selecting variables within the different categories.

For arc activation variables y_a a severe imbalance arises concerning the impact of branching between the up- and the down-branch. While forbidding arcs ($y_a = 0$) might cut off a noticeable part of the solution space, opening arcs ($y_a = 1$) does not have any impact but an increase in the objective function value. Compared to

the parent node, the solution space is not narrowed. As that might lead to larger sub-trees at the up-branches, we have expected that it is beneficial to explore as little as possible of them and always prefer the down-branches instead. We have defined the node selection priority in SCIP accordingly. Nevertheless, it is still unclear which of all possible down-branches should be explored first. We have assumed the impact to be largest if the flow on the corresponding arc is large. Hence, it has been decided to branch on the activation variable y_a with the largest (fractional) flow in the current solution of the reduced master problem ($a = \text{argmax}_{\tilde{a} \in A}\{ \bar{x}_{\tilde{a}} \}$). If there are multiple arcs with equal flow, the one that appears earliest in the time-expanded network will be chosen. Finally, in another tie we select randomly.

When branching on arc flow, we have assumed a similar approach to be beneficial. Again, we choose the arc with the highest flow in the current LP solution, since this might affect the most commodities (or path flow variables). Ties are broken by preferring arcs early in the time-space network and finally by random selection. In contrast, there seems to be no obvious imbalance of the down- and up-branch so we treat them with the same node selection priority. For path flow variables, we have applied the same approach. However, the selection of a variable λ_p^k is not determined by the variable's value in the current solution, but by the sum of the flow on all arcs that are incorporated into the corresponding path. So again, we have tried to maximize the amount of commodities that are affected by the branching decision.

Please note that the next node for exploration is chosen by its lower bound (best-first strategy), if there are multiple nodes with equal selection priority. Since our computational experiments have revealed that good feasible solutions can be found quickly and most of the computing time is spend in proving optimality, we have followed the suggestion of Gendron and Larose (2014) that best-first selection is a superior strategy in this case. In comparison to depth-first selection, which focuses on finding feasible solutions, best-first selection concentrates on improving the lower bound.

Part III

Heuristic Approaches

9 | Commodity Aggregation

In Section 5.7, we explained how commodities can be aggregated without changing the optimal solution of MIP (5.10). In the following, some constraints of that approach will be relaxed to allow further commodities to be aggregated but without a guarantee that the resulting MIP instance will not lead to a sub-optimal solution.

9.1 Relaxing Aggregation Prerequisites

Looking at the prerequisites of aggregating commodities in Section 5.7, two different types of constraints can be identified. Requiring equality of source and / or sink nodes and allowing no arc-requirements (Equations (5.14) and (5.15)) are rather technical constraints which should ensure that MIP (5.10) is a correct formulation of the presented model and that a solution of the MIP can be interpreted. We assumed that the interpretation of a solution might be challenging when relaxing one of these technical requirements and thus we decided to postpone such an approach to future research.

However, there is another type of constraints that can be considered. Ensuring that two commodities $k_1, k_2 \in K$ must be allowed to use the same arcs (Equation (5.12)) at the same costs (Equation (5.13)) are no technical requirements, but directly related to the problem domain. It can be discussed with a logistics planner if equality regarding these constraints is really required from a business perspective or if it would be sufficient to be "similar". Lets assume it would be acceptable that arc costs are just close to each other but not equal. In that case Equation (5.13) can be reformulated to

$$|c(a, k_1) - c(a, k_2)| \leq \epsilon_c \qquad \forall a \in A^{k_1} \tag{9.1}$$

with $\epsilon_c \in \mathbb{R}_{\geq 0}$ spanning a range of accepted cost differences. Similarly, it might be acceptable that the sets of allowed arcs A^{k_1} and A^{k_2} are not equal but both sets

contain a limited number of arcs that is not available in the other arc set. This would lead to a reformulation of Equation (5.12) similar to

$$\left|A^{k_1} \setminus A^{k_2}\right| + \left|A^{k_2} \setminus A^{k_1}\right| \leq \epsilon_A \tag{9.2}$$

with $\epsilon_A \in \mathbb{Z}_{\geq 0}$ specifying the amount of non-common allowed arcs. Both relaxations will enable the aggregation of further commodities and thus might be a tool for reducing the size of a MIP instance. Unfortunately, the aggregation algorithm will not be as straight forward as it has been using the approach from Section 5.7. If equality is ensured in Equations (5.14) and (5.15) the ability to aggregate commodities is a transitive property. Given commodities $k_1, k_2, k_3 \in K$, if k_1 and k_2 as well as k_2 and k_3 can be aggregated this implies that k_1 and k_3 can be aggregated, too. This property does not hold for the relaxed constraints above. In the next section an algorithmic approach to handle this issue will be presented.

9.2 Aggregation by Clique Partitioning

Given is a set of commodities K and a relation $R \subseteq K \times K$ containing a tuple (k_1, k_2) with $k_1, k_2 \in K$ if commodities k_1 and k_2 can be aggregated. It is expected that (k_1, k_2) is contained in R if the same holds for (k_2, k_1) and vice-versa. Besides that, there are no constraints on the construction of R. It is possible to use the definitions of Section 5.7, the relaxations presented in Section 9.1 or other commodity aggregation rules.

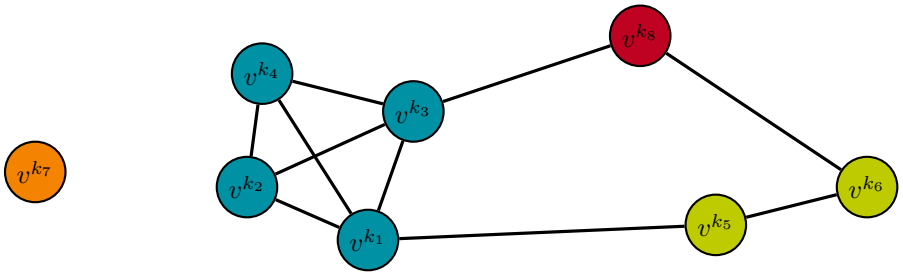


Figure 9.1: Example of a support graph that is used to aggregate commodities by searching for a minimum clique cover. The colors of the nodes visualize the cliques partitioning the graph.

The aggregation algorithm starts with the construction of a (undirected) support graph $\mathcal{G} = (V, E)$ with V containing a node v^k for each commodity $k \in K$. The edge set E contains an element $e = \left\{ v^{k_1}, v^{k_2} \right\}$ if commodities k_1 and k_2 can be aggregated ($(k_1, k_2) \in R$). An example of such a support graph is shown in Figure 9.1.

Finding a minimum set of aggregated commodities $\tilde{K}$ is equivalent to finding a minimum clique cover of support graph $\mathcal{G}$. Unfortunately, since Karp (1972) that problem is known to be NP-hard. However, there are efficient heuristic algorithms available to calculate a "near" minimum clique cover. For a comprehensive survey see Lewis (2016) and Sun (2018). We decided to implement the "DSATUR" algorithm described in Brélaz (1979) for coloring the vertices of a graph as it is simple to implement and has acceptable running times even on large graphs with reasonable solution quality. It is shown by Bhasker and Samad (1991) that such a vertex coloring on the complementary graph of $\mathcal{G}$ can easily be transformed to a clique cover of $\mathcal{G}$. The overall procedure of coloring the graph and deriving a clique cover afterwards is listed in Algorithm 9.1.

Algorithm 9.1 Minimum Clique Covering

```
1:  procedure MinCliqueCover(𝒢)
2:      G̃(V, Ẽ) ← ComplementaryGraph(𝒢)
3:      𝒞 ← ∅                                          ▷ Set of all color classes
4:      Q ← V                                          ▷ Queue of uncolored nodes
5:      while Q ≠ ∅ do
6:          v ← ChooseNextNode(Q)
7:          for j ← 1 ... |𝒞| do
8:              if C_j ∪ Neighbors(v) = ∅ then         ▷ No neighbor of v is in class C_j
9:                  C_j ← C_j ∪ { v }
10:                 break
11:             else
12:                 j ← j + 1
13:             end if
14:         end for
15:         if j > |𝒞| then                           ▷ Open new color class for node v
16:             C_j ← { v }, 𝒞 ← C_j
17:         end if
18:         Q ← Q \ { v }
19:     end while
20:     return 𝒞                                      ▷ The color classes of G̃ are a clique cover on 𝒢
21: end procedure
```

In each iteration of the algorithm an uncolored node v is chosen and assigned to a matching color class. If none of the existing color classes can be used, because another node u in the neighborhood of v is already part of that class, a new one is created. This is a simple greedy approach to color all nodes. The main contribution of the "DSATUR" algorithm is the implementation of function ChooseNextNode. It uses the *saturation degree* of a vertex as the primary criterion for choosing the next vertex to color. Let $color(v) = 0$ for any vertex $v \in V$ that has not been assigned to a color class, otherwise $color(v)$ indicates the index of the assigned color. The saturation degree $sat(v)$ of a given vertex v is the amount of different colors assigned to adjacent vertices:

$$sat(v) = |\{\, color(u) \mid u \in N(v), color(u) \neq 0 \,\}| \tag{9.3}$$

Using this definition, the next uncolored node chosen in the algorithm will be the one with maximal saturation degree. If there are multiple nodes with maximal saturation degree then ties are broken by choosing the vertex v among these with the largest degree $|\delta(v)|$. Further ties will be broken randomly. The central idea behind the usage of the saturation degree is that it prioritizes nodes that are already constrained by many different colors in their neighborhood. These nodes have only a few colors left to choose from if no additional color should be used. Coloring these constrained nodes first seems reasonable. We will evaluate the performance of this algorithm and how it can be used for commodity aggregation in Section 18.1.

10 | Heuristic Path Search & Pricing

During a run of our branch-and-price-and-cut solver a noticeable amount of computing time is consumed by our path search algorithms (see our computational study in Chapter 16). We have already introduced different techniques to speed up that process (see Sections 6.4 and 8.6). However, we have always assumed that the shortest path sub-problems will be solved to optimality for each commodity. Column generation does not require an optimal solution of all sub-problems to maintain its exact nature. It must only be ensured that a column with negative reduced costs is found, if there exists any. We will use that property to perform heuristic shortest path search in the following.

Figure 10.1 shows exemplary the typical convergence behavior of our path search algorithms for a few randomly selected commodities from our benchmark instance C500D30TT-1. Some of the algorithms are able to immediately find the optimal path and terminate, while others converge slowly against the optimal path and require some more iterations to prove optimality. We have derived different parameters from the box plots in Figure 10.1 which can be used to limit the amount of iterations of our path search algorithms. Parameter α_1 specifies the amount of iterations after which the path search will be terminated, if no path could have been found so far. The same holds for α_*, however, this parameter will only be used if there is any path available. Finally, parameter α_Δ limits the amount of iterations between improvements of the best known path. All of these parameters force our path search algorithms to terminate, even though there are still unprocessed labels that might lead to a better path. However, if configured correctly, we consider that it is unlikely that one of the remaining labels will lead to an improvement.

Within a column generation procedure the iteration-limited (heuristic) path search can be used to quickly find new columns. However, the algorithm might fail to find a new column, even if one exists. To maintain the exact nature of our branch-and-price-and-cut approach, it is required to do an exact path search as

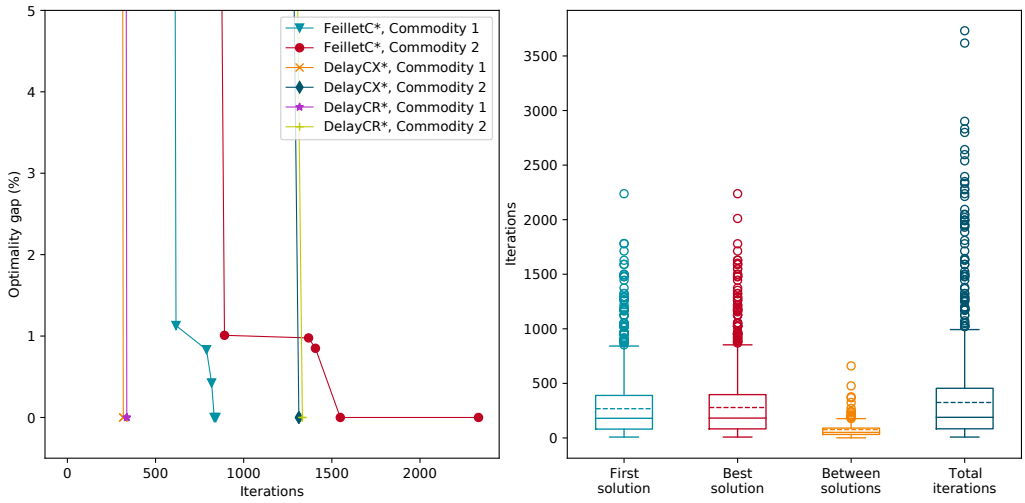


Figure 10.1: Charts showing the convergence of the solution value for different path search algorithms and some randomly selected commodities of benchmark instance C500D30TT-1. The box plots summarize the amount of required iterations to find the first and best solution. Furthermore, the total amount of iterations and the iterations between two consecutive solutions are reported.

soon as no new columns can be found heuristically. Only if the exact search does not find any new columns either, we can conclude that the pricing sub-problem has correctly been solved to optimality.

Besides accelerating the path search, there is also the opportunity to avoid searching for new columns altogether. In Section 8.6, we have explained that it might be beneficial to price more columns than the amount actually incorporated into the LP. Hence, in each pricing iteration there is a significant amount of columns with negative reduced costs that just will be ignored, because other columns have been preferred. Instead of dropping these columns, it might be beneficial to store them in a column pool and consider them in upcoming pricing iterations.

Combining above ideas, we have implemented a hierarchical pricing scheme similar to the one presented by Dell'Amico, Righini, and Salani (2006). Whenever pricing new columns for a set of commodities, at first the column store of each of these commodities is scanned for columns with negative reduced costs. Since only high-quality columns should be selected in that step, we reuse a column from the store only, if its reduced costs are better than the average reduced costs of the previous pricing iteration. If there is no favorable column in a commodity's column store,

we will do a heuristic path search by limiting the amount of allowed iterations of the algorithms. If iterating this procedure over all commodities does not lead to sufficiently many candidate columns C, pricing is continued with exact path search algorithms for all commodities which did not provide any column yet. The exact iteration is terminated as soon as there are no commodities left or it has revealed enough candidate columns ($|C| \geq \varrho_{min}$).

This final exact pricing iterations are crucial for guaranteeing correctness of our branch-and-price-and-cut solver. However, within our computational experiments we have shown that these final iterations consume a noticeable amount of computing time and, even worse, often lead to further (heuristic) pricing iterations (see Section 18.2). Hence, we have studied how skipping these exact pricing iterations will affect the quality of our MIP solutions. Obviously, the lower bounds calculated in each node of the branch-and-bound tree might be wrong (too large for our minimization problem), which in turn might cut-off the branch containing the optimal solution. Furthermore, the inaccurate lower bound prevents us from determining a reasonable MIP gap, which impedes any statement on the quality of the best found MIP solution. As a consequence, omitting the final exact pricing iterations transforms our exact branch-and-price-and-cut solver into a math-heuristic. Nevertheless, this might be acceptable for practical applications. We will investigate the performance of the hierarchical pricing scheme and the solution quality when doing heuristic pricing in Section 18.2.

11 | Flow Rounding

During our computational experiments, we observed that branching on arc-flow or path-flow variables occurs only occasionally. Instead, when there are no more fractional activation variables y_a, typically the resulting multi-commodity flow obtained from the LP-relaxation is either integral or shows only very few fractional values. Typically, less than 100 path-flow variables λ_p^k are fractional for our C50D10 instances, which almost always have more than 100 000 of these variables. Nevertheless, branching on these variables can consume a noticeable amount of time, since solving the LP-relaxation of the branched nodes can be expensive. We wondered, if it is possible to derive an integral multi-commodity flow from such a fractional solution using some kind of rounding heuristic. In this case, we could remove the integrality condition on variables λ_p^k from MIP (8.5) to speed up the solving process.

Rounding heuristics have been studied in the literature in the past and modern out-of-the-box MIP solvers normally use some of them as so called *primal heuristics*. Achterberg (2009a) and Berthold (2006) give a comprehensive overview on this topic. In the following, we will focus on heuristic approaches that use fractional path-flow values $\bar{\lambda}_p^k$ obtained from the LP-relaxation to derive an integral multi-commodity flow. We assume that decisions concerning the network's design already have been carried out and thus all arc activation variables y_a have an integral value in $\{ 0, 1 \}$.

11.1 Rounding & Shifting

The first two heuristics that we will examine are called *rounding* and *shifting* (Berthold 2006). Since these methods are closely connected, we will explain them together. For each path-flow variable λ_p^k, we define the amount of constraints down-locking or up-locking the variable to be $\zeta_{p,k}^-$ and $\zeta_{p,k}^+$. A constraint locks a

variable, if the fractional variable value $\bar{\lambda}_p^k$ cannot be rounded to the next integer without violating the constraint. Whether it is a down- or up-lock only depends on the direction the value is rounded. Within our problem domain, it always holds that $\zeta_{p,k}^- \in \{ 0, 1 \}$, since only the convexity constraint (8.5c) of commodity k can be violated by rounding down a variable value. In contrast, $\zeta_{p,k}^+ \in \mathbb{Z}_{\geq 0}$ is bounded by the length of path p, since it counts the amount of (arc) capacity constraints (8.5b) that are violated when rounding up. For a more detailed treatment of variable locks the reader is referred to Achterberg (2009a, p. 38).

The algorithm starts by investigating if any constraint (except the integrality condition) is violated by the current fractional solution. If the solution is feasible, the fractional variable with the largest lock $\max\left\{ \zeta_{p,k}^-, \zeta_{p,k}^+ \right\}$ is selected and rounded in the direction that violates less constraints. So it is rounded downwards for $\zeta_{p,k}^- \leq \zeta_{p,k}^+$ and upwards otherwise. Since all fractional variables must be rounded, this procedure avoids rounding of variables which have a high potential of breaking feasibility in the "wrong" direction.

As long as there exist violated constraints, one of them is selected and a fractional variable (contained in that constraint) is rounded in the direction reducing the violation. If there are no fractional variables the algorithm terminates without an integral solution. In contrast, if there are multiple variables, the one with the smallest variable locks (with respect to the direction of rounding) is chosen. Ties are broken by picking the variable that leads to minimal increase of the objective function (in case of a minimization problem). The rational behind this choice is that the possibility of breaking further constraints should be minimized, when trying to recover feasibility. Furthermore, the solution value should increase as little as possible, to find high quality integral solutions.

While the rounding algorithm is aborted, if there is no fractional variable decreasing the violation of a constraint, the shifting algorithm continues in this case by shifting the value of a non-fractional integer variable. As this algorithm can easily get stuck in a cycle, special care must be taken to ensure termination. For this, the shifting implementation keeps track of previous shiftings and penalizes repeated ones in opposite directions. Furthermore, the violated constraints are selected randomly and the algorithm terminates after a fixed amount of successive shiftings, which did not improve the amount of violated constraints or fractional variables. For a more detailed explanation of both algorithms, we refer to the diploma thesis of Berthold (2006).

11.2 Relaxation Enforced Neighborhood Search (RENS)

While the previous section dealt with heuristic approaches to find a feasible flow rounding, in the following we are going to reach for the optimal rounding (in regard to minimal cost increase), if there exists any. For this purpose, Berthold (2006) introduced a straightforward method called *relaxation enforced neighborhood search (RENS)*. Given a fractional solution vector $\bar{\boldsymbol{\lambda}}$ that has been obtained by solving the LP-relaxation, this algorithm solves the sub-MIP

$$\min \sum_{k \in K} \sum_{p \in P^k} c_p^k \lambda_p^k \tag{11.1a}$$

$$\text{s.t.} \sum_{k \in K: a \in A^k} \sum_{p \in P^k} \alpha_{ap}^k \lambda_p^k \leq u_a \qquad \forall a \in A, \bar{y_a} = 1 \tag{11.1b}$$

$$\sum_{k \in K: a \in A^k} \sum_{p \in P^k} \alpha_{ap}^k \lambda_p^k \leq 0 \qquad \forall a \in A, \bar{y_a} = 0 \tag{11.1c}$$

$$\sum_{p \in P^k} \lambda_p^k \geq b(v_d^k, k) \qquad \forall k \in K \tag{11.1d}$$

$$\lambda_p^k \in \left\{ \left\lfloor \bar{\lambda}_p^k \right\rfloor, \left\lceil \bar{\lambda}_p^k \right\rceil \right\} \qquad \forall k \in K, \forall p \in P^k \tag{11.1e}$$

which can be derived from MIP (8.5) by replacing all y_a with their constant value in $\{ 0, 1 \}$ and tightening the domain of λ_p^k to $\left\{ \left\lfloor \bar{\lambda}_p^k \right\rfloor, \left\lceil \bar{\lambda}_p^k \right\rceil \right\}$. Please note that integral solution values will lead to a fixing of the corresponding variable to $\lambda_p^k = \bar{\lambda}_p^k$. Furthermore, MIP (11.1) can be easily reformulated as a pure binary program by substituting $z_p^k = \lambda_p^k - \left\lfloor \bar{\lambda}_p^k \right\rfloor$. The resulting MIP can be solved by any out-of-the-box solver, which might take a considerable amount of computing time, if there are many fractional solution values in vector $\bar{\boldsymbol{\lambda}}$. Hence, we limited the amount of explored branch-and-bound nodes as suggested by Berthold (2006). Further computational results are provided in Section 18.3.

Since RENS will only be successful if there exists a feasible flow rounding, we wondered if we can modify this method to allow shifting of integral solution values similar to the shifting heuristic explained above. A straightforward approach would be to stop pricing within our branch-and-price-and-cut solver after the root LP-relaxation has been solved to optimality and solve the resulting (integer) reduced master problem by branch-and-cut. In this case, the path-flow will be represented by integer variables like in MIP (8.5). However, since we stopped pricing, now we can use all the powerful SCIP extensions like cut separators, presolvers, domain propagators and primal heuristics to solve the MIP. So SCIP will take care for

generating an integral flow from the fractional solution obtained in the root node LP. Nevertheless, equally to RENS that branch-and-cut phase might consume a significant amount of computing time. We will see in our computational study in Section 18.3, how this approach performs. Since all pricing is carried out before any branching or cut separation, we call this approach *price-and-branch-and-cut*. Please note that it is a heuristic method, as we cannot guarantee that all variables required for an optimal solution are generated before the branch-and-cut phase starts. In the worst case, this method might even fail to find a feasible solution. Nevertheless, this weakness could be overcome easily by continuing pricing until a feasible integral solution has been found.

Alternatively, it would be possible to solve the root node LP-relaxation to optimality and create a sub-MIP similar to the one of RENS, but without further limiting the domain of variables λ_p^k. This MIP would be exactly the same as the one which is solved during the branch-and-cut phase of the aforementioned price-and-branch-and-cut approach. However, separating the work into two independent MIPs has one major advantage: the branch-and-cut phase does not need to be carried out by SCIP. Instead, we can use any of the commercial out-of-the-box solvers like CPLEX, Gurobi or FICO Xpress, which typically perform significantly better. Although this method does not use the fractional solution of the LP-relaxation, but only the variables created during solving it, we call this approach *enhanced RENS* and will evaluate its performance in Section 18.3.

11.3 Randomized Rounding

All methods proposed in this chapter represent heuristic algorithms. They do not guarantee solution quality or finding a feasible solution, if there exists any. From a mathematical point of view, this issue cannot be resolved. The applied transformations might lead to an infeasible model or the algorithms are just not able to find a solution. For the use of our algorithmic framework in practice, this is an unfortunate situation. As we already explained in Section 5.6, there is some value in providing any solution, even though it is not optimal. Nevertheless, we expect that it is possible in many cases, to gain the benefit of a simplified model with relaxed flow variables and handling above infeasibility issues from a business perspective.

The simplest solution would be to ignore all capacity constraints during flow rounding. This would lead to a multi-commodity flow that is properly balanced (so all demands are fulfilled, see definitions in Section 3.2), but violates the capacity of some arcs. Since there are typically very few fractional values within an LP solution, we wondered if these capacity violations are really a problem for business operations. If there is a reserved capacity on a train of 100 vehicles, do we have any problem when placing 101 vehicles on that transport? What is about 102 vehicles? In our experience, logistics planners are able to resolve such situations easily, just by doing a phone call. However, we should at least spend some effort to minimize the congestion of the arcs, and thereby simplify the resolution for the planners.

A probabilistic algorithm for this task has been proposed by Raghavan and Tompson (1987) and is called *randomized rounding*. It has been extended to the multi-commodity case by Srinivasan (2001). For a given fractional solution vector $\bar{\boldsymbol{\lambda}}$ the algorithm rounds up the fractional solution values with probability

$$\mathcal{P}\Big(\lambda_p^k \text{ is rounded up}\Big) = \bar{\lambda}_p^k - \Big\lfloor \bar{\lambda}_p^k \Big\rfloor. \tag{11.2}$$

We have implemented this algorithm and use it as a last resort to ensure an integral (but maybe infeasible) solution, if flow rounding failed to provide one.

12 | Simulated Annealing with Branch-and-Price

In previous chapters, we focused entirely on exact methods and how they can be sped up, potentially by combining them with heuristic approaches. However, there are already powerful heuristic algorithms available in literature that can be used to solve different types of network design problems. In the following, we are going to evaluate these algorithms in an extensive literature review and choose the best one to be applied to the PCIMCFND. Furthermore, we will explain all modifications to the algorithms that have been necessary.

12.1 Literature Review

There are lots of different heuristic approaches available in literature to tackle the MCFND, ranging from simple hill-climbing, over meta-heuristics like tabu-search, simulated annealing or evolutionary algorithms, to math-heuristics. On the basis of this multitude of available approaches, we have focused our literature review on this class of problems which is quite close to our problem domain.

The first proposed heuristics for the MCFND focused on adding or dropping arcs (with fixed-costs) from / to the network and afterwards calculating the resulting commodity flow. Powell (1986) considered the marginal costs of an arc to decide if it should be part of the network. In a similar approach Farvolden and Powell (1994) focused on the marginal costs of paths in the network. However, both approaches did not attempt to explore the solutions space past the first local optimum. Crainic, Gendreau, and Farvolden (2000) tried to address that limitation with a tabu-search based method (Glover and Laguna 1998) using pivot-like moves to create neighboring solutions in the space of path-flow variables. A cooperative parallelized variant of this algorithm has been described by Crainic and Gendreau (2002).

Ghamlouche, Crainic, and Gendreau (2003) introduced a new type of neighborhood by searching for arcs holding flow that can be detoured on alternative paths. Compared to the simpler approaches that focused on adding or dropping single arcs or detouring flow of single commodities, the developed tabu-search heuristic showed superior and robust performance. Ghamlouche, Crainic, and Gendreau (2004) further improved that approach by introducing a path re-linking method (Glover, Laguna, and Martí 2000). Additional machine learning mechanisms have been introduced by Ghamlouche et al. (2011). Performing the neighborhood search in a multilevel cooperative parallelized fashion has been proposed by Crainic, Li, and Toulouse (2006). The special case of undirected networks, where flows share the capacity of an edge regardless of the flow's direction, has been studied by Alvarez, González-Velarde, and De-Alba (2005). They used scatter search (Glover, Laguna, and Martí 2000), a population-based search method where a thoroughly selected subset of all known solutions is combined to create new ones. The same meta-heuristic framework has been chosen by Crainic and Gendreau (2007). A multi-objective approach taking the costs and the length of each commodity path into account has been suggested by Kleeman et al. (2012) in form of a evolutionary algorithm.

Chabrier (2003) and Chabrier et al. (2004) were the first to propose combinations of heuristics and mathematical programming techniques (math-heuristics) to solve a special variant of the MCFND where all flow of a commodity must use a single path. Crainic, Gendron, and Hernu (2004) successfully combined a slope scaling heuristic with Lagrangian relaxation that allows to solve a sequence of LPs by adopting linear arc costs. Katayama, Chen, and Kubo (2009) tried to estimate the value of the arc activation variables y_a upfront using capacity scaling and solved the remaining MCF by column and row generation. The LP solution is used to iteratively adjust the estimate of the activation variable values. A similar approach has been described by Yaghini, Rahbar, and Karimi (2013), but they use a simulated annealing heuristic to fix the activation variables. Hewitt, Nemhauser, and Savelsbergh (2009) tried to solve the MIP directly with an approach similar to very large-scale neighborhood search (VLNS). Some variables of the MIP are fixed using a previously known solution and the other variables are re-optimized. Instead of fixing variables of the MIP explicitly, in local branching (Fischetti and Lodi 2003) some variables are fixed implicitly by introducing additional branching constraints. Rodríguez-Martín and José Salazar-González (2010) successfully applied such methodology to MCFND. The solution framework of Hewitt, Nemhauser, and Savelsbergh (2012) limits the

solution space in a similar manner. Chouman and Crainic (2010) leveraged a tabu-search method for the generation of feasible solutions within a branch-and-cut framework.

In recent years, there have been many papers published that extend or combine past approaches to more powerful heuristics. Katayama (2015) extended his own approach of capacity scaling with a local branching scheme and gains considerable improvements. An evolutionary algorithm combining scatter search with an iterated local search method using similar neighborhoods to the ones defined by Ghamlouche, Crainic, and Gendreau (2003) has been proposed by Paraskevopoulos et al. (2016). A variable fixing heuristic similar to the one from Hewitt, Nemhauser, and Savelsbergh (2009) that has been engineered to scale on distributed memory computing infrastructure and thus can explore many neighborhoods in parallel has been presented by Munguía et al. (2017). Finally, Gendron, Hanafi, and Todosijević (2018) combined iterative linear programming with the slope scaling technique of Crainic, Gendron, and Hernu (2004).

Choosing one algorithm from this multitude of heuristics is not a trivial task. There are extensive computational studies available comparing many of the above heuristics on a standard set of benchmark instances. These so called C and C+ instances have been used first by Crainic, Frangioni, and Gendron (2001) and are publicly available from the Operations Research Group of the University of Pisa (see "Data" at `http://groups.di.unipi.it/optimize/`). Using the benchmark results from the original publications, in the remainder of this chapter we are going to compare the performance of the following algorithms:

- **CTS**: Cycle-based Tabu Search, (Ghamlouche, Crainic, and Gendreau 2003)
- **PR**: Path Relinking, (Ghamlouche, Crainic, and Gendreau 2004)
- **MCA**: Multilevel Cooperative Algorithm, (Crainic, Li, and Toulouse 2006)
- **CSH**: Capacity Scaling Heuristic, (Katayama, Chen, and Kubo 2009)
- **IPS**: IP Search, (Hewitt, Nemhauser, and Savelsbergh 2009)
- **LB**: Local Branching, (Rodríguez-Martín and José Salazar-González 2010)
- **SACG1**: Simulated Annealing and Column Generation (600s time limit), (Yaghini, Rahbar, and Karimi 2013)

- **SACG2**: Simulated Annealing and Column Generation (18000s time limit), (Yaghini, Rahbar, and Karimi 2013)
- **CS-LB1**: Capacity Scaling with Local Branching (1000s time limit), (Katayama 2015)
- **CS-LB2**: Capacity Scaling with Local Branching (2000s time limit), (Katayama 2015)
- **CEA**: Cycle-based Evolutionary Algorithm, (Paraskevopoulos et al. 2016)
- **ParLS**: Parallel Local Search, (Munguía et al. 2017)
- **ILPH**: Iterative Linear Programming Heuristics, (Gendron, Hanafi, and Todosijević 2018)

Since these publications span almost two decades, just comparing the results is not an expedient approach. There have been considerable improvements in computing facilities and we want to compare the performance of the algorithms, and not the impact of the used hardware. Thus, we are going to normalize the reported computing times using PassMark CPU Scores (PassMark Software 2019). Table 12.1 lists the hardware used by the authors of the algorithms to perform their benchmarks, the CPU scores assigned by us and the resulting computing time scale factors. For any given heuristics H, normalized computing times $\bar{T}_H$ are calculated with the formula

$$\bar{T}_H = \frac{PCS(H)}{PCS(H_{ParLS})} T_H \tag{12.1}$$

where $PCS(H)$ and $PCS(H_{ParLS})$ are the PassMark CPU scores of heuristic H and the $ParLS$ heuristic, which will be used as reference, as it has the largest score value. If the type of the CPU could not be determined uniquely, we assumed the CPU matching all specifications reported by the authors with the lowest score. Unfortunately, there are no PassMark CPU Scores available for Sun processors. Hence, we used the LINPACK benchmark results provided by Dongarra (2014) to select an equivalent Intel processor, the Intel Pentium III 933S. When calculating the CPU scores, we assumed that the computational power is proportional to the number of CPUs and cores used. That is especially important when compute clusters have been used in the benchmarks or when only some of a CPU´s cores have been utilized.

Table 12.1: Details on used hardware in benchmarks and assigned CPU scores for the majority of published heuristics to tackle the MCFND

	CPU Typ	GHz	CPUs	Cores per CPU	Passmark Score	Scale factor	Solver
CTS	Sun UltraSparc II[a]	0.3	1	1	238	0.00204	CPLEX 6.5
PR	Sun UltraSparc II[a]	0.4	1	1	238	0.00204	CPLEX 6.5
MCA	Sun UltraSparc II[a]	0.4	64	1	$64 \cdot 238 = 15232$	0.13039	CPLEX 7.5
CSH	Intel Pentium E5800[b]	3.2	1	2 of 2	1889	0.01617	CPLEX 9.0
IPS	Intel Xeon	2.66	8	1	$8 \cdot 435 = 3480$	0.02979	CPLEX 9.1
LB	Intel Core 2 Duo E4600[b]	2.4	1	2 of 2	1384	0.01185	CPLEX 11.1
SACG1	Intel Core 2 Duo E6850[b]	3	1	2 of 2	1945	0.01665	-
SACG2	Intel Core 2 Duo E6850[b]	3	1	2 of 2	1945	0.01665	-
CS-LB1	Intel Core i7-2600[b]	3.4	1	4 of 4	8186	0.07008	CPLEX 12.6
CS-LB2	Intel Core i7-2600[b]	3.4	1	4 of 4	8186	0.07008	CPLEX 12.6
CEA	Intel Xeon E5507	2.27	1	1 of 4	$3125/4 = 781$	0.00669	CPLEX 12.6
ParLS	Intel Xeon X5650	3.1	16	6 of 6	$16 \cdot 7301 = 116816$	1	CPLEX 12.4
ILPH	Intel i7-4900MQ	2.8	1	4 of 4	8982	0.07689	CPLEX 12.6

[a] No PassMark CPU score available. Equivalent processor Intel Pentium III 933S used instead.
[b] Assumed from author's specification.

Using the calculated CPU scores, we are able to compare all the algorithms regarding solution quality and computing time. Table 12.2 gives an overview of the average performance of the algorithms on the 43 C and C+ instances. It shows the average relative gap of the best found solutions to an optimal solution or to the best known lower bound. Furthermore, the average times to find the best solution are reported, both the original times and the normalized ones. All computing times are reported in seconds. Finally, the table lists how many of the 43 instances have been solved to optimality and for how many of them the best known solution has been found (optimal or sub-optimal). For a detailed listing of the metrics for each of the instances the reader is referred to Chapter 23 in the appendix. When studying the performance metrics in detail, the average relative gap and the amount of optimal and best solutions found are indicators for the quality of the solutions that are found by a heuristic. Regarding that factor the algorithms SACG, CS-LB, CEA, ParLS and ILPH seem to be competitive and find

Table 12.2: Performance metrics for comparing the majority of available heuristics for the MCFND. Besides the average optimality gap and the amount of found best and optimal solutions, also the average computing time T_H reported in the original papers and the normalized computing time $\bar{T}_H$ calculated by us is presented.

	Gap	$\mathbf{T_H}$	$\mathbf{\bar{T}_H}$	**Optimal**	**Best**
CTS	4.5%	18312.3	37.3	4	4
PR	4.1%	8123.7	16.6	5	5
MCA	3.5%	4061.9	529.6	3	3
CSH	1.4%	463.1	7.5	6	6
IPS	1.9%	408.1	12.2	6	6
LB	2.9%	554.3	6.6	14	14
SACG1	2.1%	428.2	7.1	13	13
SACG2	0.7%	3956.7	65.9	16	17
CS-LB1	0.5%	3162.1	221.6	20	23
CS-LB2	0.4%	6728.1	471.5	19	26
CEA	1.0%	4553.5	30.4	14	14
ParLS	0.6%	152.7	152.7	15	17
ILPH	0.7%	927.6	71.3	24	24

near-optimal solutions for almost all instances. However, there is a considerable deviation regarding normalized computing times. While SACG, CEA and ILPH find the best solutions in almost 1 minute or less, the other algorithms require 2.5 up to 8 minutes. We decided to implement SACG, since it showed a reasonable balance between solution quality and computing time. Furthermore, in the original paper, this algorithm utilizes column generation to solve a sub-problem and we expect that this part can be easily replaced with our branch-and-price-and-cut solver. Hence, all work that we spent in improving this approach would be available within this new heuristic algorithm.

We want to close this review with a final note on the selection process described above. We are aware of the fact that scaling computing times using CPU scores is not an exact science, since there are so many factors that cannot be taken into account. First, we do not have any guarantee that the PassMark CPU scores have been computed in a laboratory environment and can be considered as unbiased. Second, we were forced to make estimates and assumptions when selecting a score for a heuristic's hardware. Supposing a linear correlation between the score and the amount of used CPU cores might be another source of errors. Furthermore, in the past decades there have been enormous improvements regarding out-of-the-box solvers, that are used by most heuristics internally. How the overall heuristic's performance is influenced by the performance of the used solvers remains ambiguous. The same holds for used compilers, compiler flags and external libraries which have not been reported in the publications at all. If all calculations were to be repeated in a laboratory environment asserting equal characteristics for all measurements, we would expect Table 12.2 to report different metrics. However, we believe that the magnitude of the metrics would remain unchanged, so that the SACG heuristic still is a valid choice for our requirements.

12.2 SACG: Simulated Annealing and Column Generation

The SACG heuristic presented by Yaghini, Rahbar, and Karimi (2013) combines simulated annealing with column generation to solve the MCFND. The idea behind simulated annealing is to move from a given solution to a neighboring solution if this improves the solution value. However, this neighbor solution is accepted also in the case where the solution value is getting worse, but only with an progressively decreasing probability. The probability is "cooling down" similar to the annealing of metals, which explains the origin of that method's name.

Figure 12.1 gives a high-level overview of the algorithm developed by Yaghini, Rahbar, and Karimi (2013). In the following, we will only give a shallow summary of their work, for a detailed description, we refer to the original publication. The algorithm starts with the construction of an initial solution by relaxing all activation variables and solving the remaining LP using column generation. Afterwards, all arcs not used within the LP solution are discarded from the original MIP and the remaining simplified program is solved to optimality. This initial solution is used to initialize the simulated annealing.

Starting from a given solution, the algorithm finds neighboring solutions by closing or opening arcs and solving the remaining LP using column generation. Please note that all arc activation variables are fixed and not available any more in

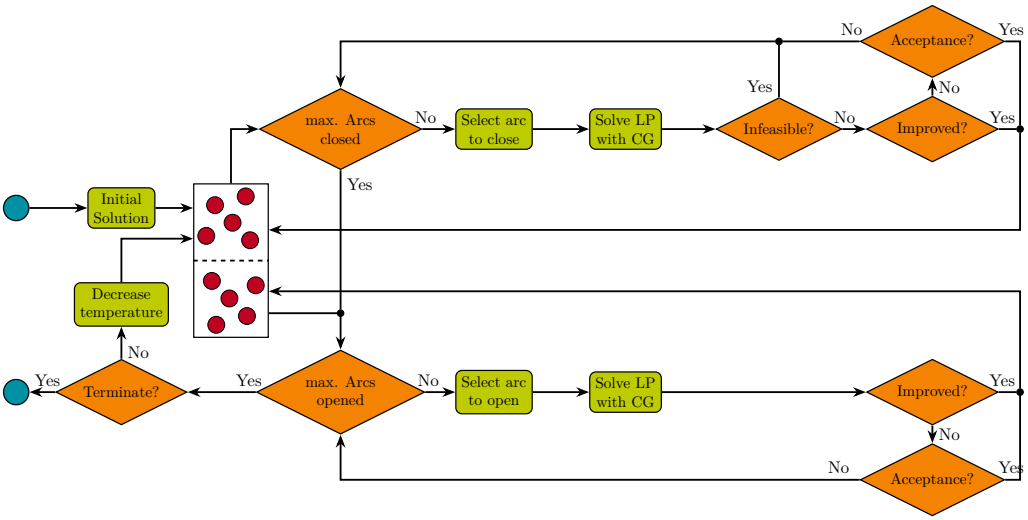


Figure 12.1: Flow chart of an algorithm combining simulated annealing and column generation.

the LP. Thus, the solution of the MCFND is split into two phases. First it is decided which arcs will be available in the network (setting variable $y_a = 0$ or $y_a = 1$). Second, the multi-commodity flow using that network is determined by solving the remaining MCF. The selection of an arc to close or open during an iteration of the algorithm is guided by two different scoring functions that are picked randomly and either focus on the arc utilization or the arc usage costs. The SACG implementation switches repeatedly between a few iterations that only close arcs ($y_a = 0$) and a few iterations that only open arcs ($y_a = 1$). If the LP becomes infeasible after closing an arc, this arc will be reopened again for the next iteration. If the neighbor solution improves the solution value it will become the given solution for the upcoming iteration. In case there is no improvement, the neighbor solution will still be chosen, but only with a probability of

$$\mathcal{P}(\text{Choose solution in iteration } i) = \exp\left(\frac{f_{i-1} - f_i}{T_i}\right). \tag{12.2}$$

Here $f_i \in \mathbb{R}_{\geq 0}$ is the LP solution value of the i-th iteration and $T_i \in \mathbb{R}_{\geq 0}$ is the *temperature*. Please note that $f_{i-1} - f_i \leq 0$, if there is no improvement of the solution value during iteration i, which ensures that the probability will be in the interval $(0, 1]$. Furthermore, choosing T_0 properly is an important factor for the success of simulated annealing. There exist publications in literature facing this single topic only (see Ben-Ameur 2004). We followed the suggestion of Kirkpatrick, Gelatt, and Vecchi (1983) and chose T_0 to be equal to the maximal expected cost increase between two neighboring solutions (see details in Section 18.4). Please note that determination of the initial temperature is one major disadvantage of this solution method and complicates its use in practice, as we expect that the procedure for calculating T_0 must be adopted in each and every usage scenario.

After a sequence of consecutive close and open movements the temperature will be decreased by the *cooling rate* Δ using the formula $T_i = T_{i-1} - \Delta$. It must be ensured that T_0 and Δ are chosen appropriately to avoid a negative temperature value. If no termination criterion is met, the algorithm starts over with the next iteration. As soon as the maximum amount of iterations is reached, the computing time restriction has been exceeded or the limit of consecutive non-improving solutions has been attained the algorithm will terminate.

Obviously, SACG cannot be considered for solving instances of the PCIMCFND without modifications. During column generation, it is required to use the path

search algorithms described in Chapter 6 when pricing new columns, instead of using one of the classical shortest path algorithms. Furthermore, the PCIMCFND asks for integral flows, so whenever an LP is solved within the original algorithm, we are going to solve a MIP instead. Both requirements can be fulfilled by replacing the column generation part of SACG with our branch-and-price-and-cut solver. Of course, this modification could considerably increase overall computing times. Therefore, we used the optimized configuration of our solver (without cut separation, see Section 16.7) and further accelerated it by using heuristic approaches. We distinguished between solving the MIP during the initialization phase of SACG and all subsequent solver runs. For the former, we relaxed all flow and arc activation variables and used the flow rounding techniques presented in Chapter 11 to find an integral initial solution. For the latter, we additionally activated heuristic pricing as it has been introduced in Chapter 10. Both techniques might limit the increase of computing times and should have only a minor impact on solution quality (see Sections 18.2 and 18.3).

Finally, during preliminary testing we observed one major drawback of the original algorithm. Whenever closing an arc leads to an infeasible MCF, this arc will be reopened again and the algorithm continues with the next iteration. However, in the original paper there is no hint how to avoid that the next iteration will not close the same arc again. Although there are two different arc selection strategies in use and the algorithm switches randomly between them, we observed longer sequences during our experiments, where the same arc is chosen repeatedly. The same happened for iterations of opening or closing arcs which did not lead to an update of the current solution. To overcome this inefficiency, we modified the algorithm and black-listed all arcs which had already been touched (opened or closed) in the current solution without success. The blacklist is cleared as soon as the current solution changes. We will evaluate how this modified variant of SACG performs on our benchmark instances in Section 18.4. To better reflect the ingredients of this algorithm, we renamed it to *simulated annealing with branch-and-price (SABP)*.

Part IV

Computational Study & Results

13 Preliminaries

In the following, we will evaluate all presented solution approaches in an extensive computational study. First, the used benchmark instances and computing environment are introduced. Afterwards, the computational properties of each solution approach are evaluated and compared to other approaches from the literature where possible. Furthermore, we will give some hints on parameter tuning.

13.1 Benchmark Instances

Evaluating the effectiveness of a given solution approach requires a set of well selected benchmark instances. These instances should support the comparison of different approaches and they should be as close to real-world instances as possible. The latter is important for ensuring that the solution method will be able to tackle the problems in practice and is not limited to a laboratory environment only. Therefore, we have developed an instance generator that is able to create randomized instances of different sizes. However, creating instances that are similar to real-world instances is not a trivial task and requires some insights into the structure and characteristics of real transport networks and vehicle flows. In the following, we will explain our assumptions that are used when generating such instances regarding factors like the

- amount of assembly plants including locations and production capacities,
- volume of vehicles being produced or located at intermediate locations,
- length of the planning horizon,
- sales volumes of different geographic regions with their delivery locations,
- available means of transport with costs, frequency and traversal time,
- other nodes of the network like compounds or sea ports,

- routes within the network connecting plants to delivery locations and
- additional services required for the vehicles.

The production of finished vehicles is carried out in assembly plants that are distributed all over the world (see Figure 1.3). Looking at statistics regarding the annual production capacity of the assembly plants of the manufacturers Toyota, Volkswagen, Hyundai, General Motors and Ford reveals, that the production capacity of the plants approximately follows a gamma-distribution. The corresponding probability density function with shape parameter α and scale parameter β is

$$f_\Gamma(x;\alpha,\beta) = \frac{x^{\alpha-1}e^{\frac{-x}{\beta}}}{\beta^\alpha \Gamma(\alpha)} \tag{13.1}$$

where $\Gamma(\alpha)$ is the gamma function $\Gamma(\alpha) = (\alpha-1)!$. Figure 13.1 shows a histogram of the annual production capacities (in thousands) collected from the plants of the aforementioned manufacturers and a fitting probability density function with $\alpha = 1.2$ and $\beta = 190$. We expect the overall amount of assembled vehicles that need to be considered within a production planning horizon of D days to be input

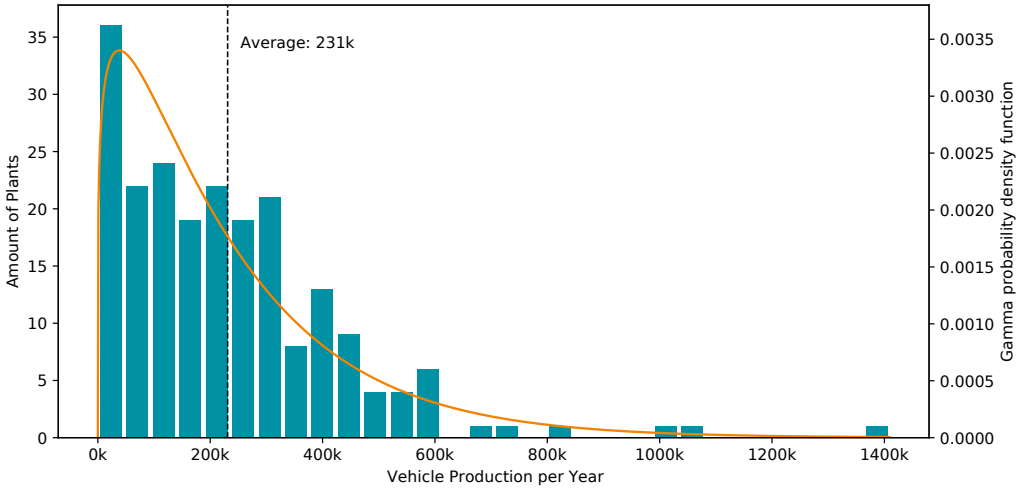


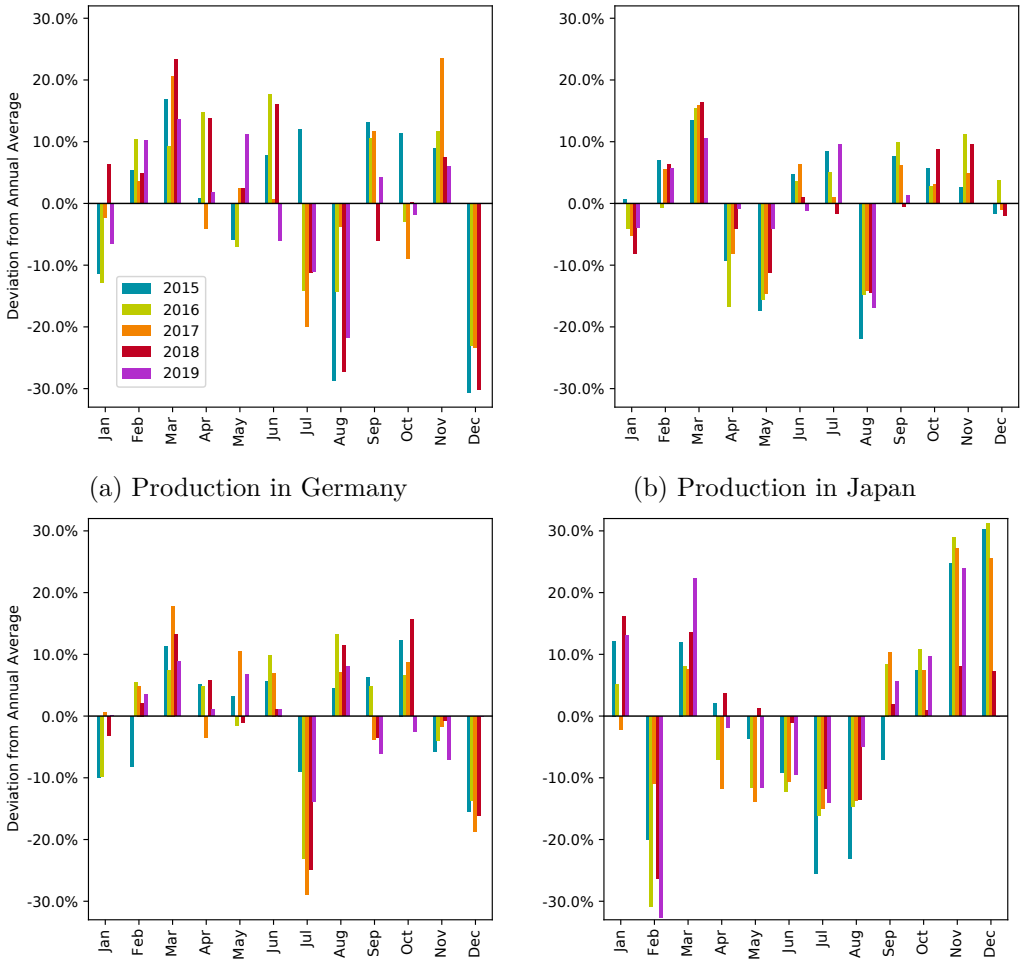
Figure 13.1: Chart showing a histogram of the production capacity of assembly plants in 2019 and an approximating gamma probability density function with $\alpha = 1.2$ and $\beta = 190$. (Source: Volkswagen Group 2019; MAN SE 2019; Toyota Motor Corp. 2018; Daihatsu Motor Co. 2019; Hino Motors 2019; Hyundai Group 2019; Kia Motors Corp. 2019; General Motors 2019; Ford Motor Company 2019)

parameter C of the instance generator. However, assuming that all vehicles will be produced in an assembly plant during the production planning horizon would be inaccurate. Certainly, there will already be vehicles traveling through the transport network that have been produced earlier. Thus, we split parameter $C = C_P + C_N$ into the vehicles C_P that will be produced and the vehicles C_N that are located at a random compound within the network. We expect $C_N \approx 0.2C$, so 20 % of the overall vehicle volumes will start at a compound. Considering the production amount C_P and assuming 250 work days within a year the amount of assembly plants P for a generated instance can be calculated by

$$P = \left\lceil \frac{C_P}{1000\alpha\beta} \cdot \frac{250}{D} \right\rceil \tag{13.2}$$

with $1000\alpha\beta$ being the average annual production capacity when considering a gamma distribution. During instance generation, P assembly plants of any of the aforementioned manufacturers will be selected randomly (with uniform probability) and to each of the selected plants a gamma-distributed annual production capacity will be assigned. Afterwards, these capacities are scaled down linearly to the production planning horizon so that finally C_P vehicles will be produced in D days at P assembly plants. To avoid degenerated instances with plants producing only a few vehicles (which is possible when using a gamma distribution), we ensure that each plant will at least have a share of 5 % of the overall annual production volume. Please mind that D is not equal to T_{end} that has been introduced in Section 5.2 as the maximum index of the time discretization. Typically, it holds that $D \ll T_{end}$ since T_{end} will be chosen in a way that allows all vehicles to reach their destinations while D only determines when no further vehicles will start at their sources.

The production of vehicles exhibit seasonality throughout the year. So assuming a uniform distribution of the annual production capacity to the days of the planning horizon would excessively simplify reality. Looking at the monthly production statistics for the world's largest manufacturing countries Germany, Japan, China and USA, supports that assumption. Figure 13.2 shows the deviation from the annual average production for the years 2015 to 2019. We aggregated these statistics and used the deviations shown in Figure 13.3 to distribute the annual production capacity to the months of the planning horizon. Within the same month a uniform distribution is used to derive the capacity per day. Although the production of the aforementioned countries might not necessarily be transferable to other countries in the world, it is more important that there is a realistic non-uniform distribution



(a) Production in Germany

(b) Production in Japan

(c) Production in USA

(d) Production in China

Figure 13.2: Charts showing the relative deviation of the monthly production volumes for the years 2015 to 2019 from the annual average volumes for Germany, Japan, China and USA. (Source: VDA 2019; JAMA 2019; BEA 2019; CAAM 2019)

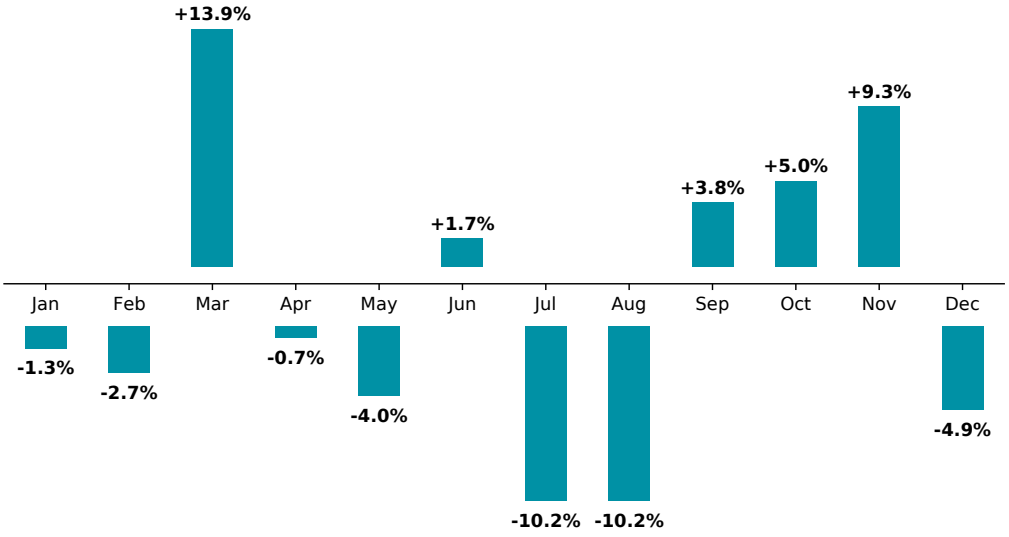


Figure 13.3: Relative deviations of annual average production volumes that are used to derive monthly from annual capacities.

in use than the actual numbers of that distribution. We assumed that the planning horizon starts at a random day of the year and that there will be no production on Saturdays or Sundays.

While the assembly plants are one end of our outbound supply chain, the sales markets form the opposite end. There are many factors that drive the manifestation of sales volumes like population, quality of the transport infrastructure, degree of urbanization, environmental awareness or the mindset of the people regarding possession of a vehicle. Instead of modeling all of them, we decided to use the real annual sales volumes obtained in 2017[1] that are provided publicly by OICA (2019b) (same data is presented as heat-map in Figure 1.4). These sales figures are scaled linearly to match the length of the production planning horizon D and afterwards a set of markets is selected randomly (with uniform probability) that just exceeds the desired vehicle volume C. Subsequently, the sales figures of the selected markets are scaled down linearly to be exactly equal to C. Similarly to production capacities, the data reveals a seasonality of the sales volumes throughout the year. We will not reflect the sales' seasonality explicitly but assume that this has already been incorporated into the fluctuation of the production volumes. Thus, we will distribute the sales volumes to the days of the planning horizon by estimating the

[1]When we generated our benchmark instances in April 2019 the most recent dataset published by OICA has been from 2017.

transport times from the assembly plants to the markets. We will further discuss this point later.

Since the data provided by OICA (2019b) has a granularity of countries, besides determining the sales volumes of the markets, it is further required to generate some random final destination locations within these markets: the dealers / customers and VHCs. While the latter will be combined with some distribution compounds (see below), the first will be generated randomly within the largest cities of the markets. Datasets like the one provided by GeoNames (2019) can be used to identify these cities and provide the required geo-spatial information. In many markets, it is a common practice to buy vehicles from stock while other markets have a strong need for customization and want to order vehicles with a personalized feature-set instead. To reflect that within our instance generator, we will randomly decide per market which proportion between 20 % to 80 % of the vehicles will be bought from stock (uniformly distributed). These vehicles will reside in a VHC before being delivered to a dealer. The remaining market volume will be distributed to the largest cities. The population of each city will be used to determine its probability to be chosen by the instance generator. However, using each city as an individual destination location is an unrealistic assumption as typically in strategic

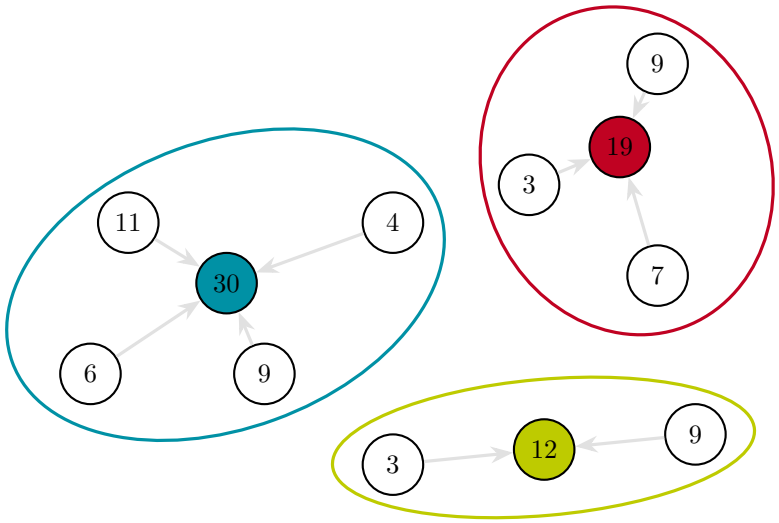


Figure 13.4: Example of aggregating destination locations to delivery zones by clustering. The numbers within the nodes correspond to the assigned sales volumes. Only the colored nodes will be added to the benchmark instance.

and tactical planning delivery regions are used instead. To reduce the degree of detail, we clustered the selected cities to larger regions using an agglomerative hierarchical clustering algorithm similar to the one presented by Eppstein (2000). The algorithm used a complete-linkage strategy with a distance threshold of 300 km. In the benchmark instances each cluster has been replaced by its centroid and its demand will be summed up over all sales volumes in that cluster. Figure 13.4 shows an example of such a clustering.

After production and sales volumes have been determined, they can be combined for generating vehicles (or commodities) with an assembly plant as origin and a dealership as destination. Both, plants and dealerships will be selected randomly using a uniform distribution. The special cases of vehicles that will reside in a VHC or that are not produced but start within a compound are ignored for now. A proportion of 10 % of all generated vehicles will be flagged to require a modification within a VMC. Obviously, all of these vehicles require a transport network to be able to reach their destination from their origin. We assume that this network must be able to support three different delivery scenarios:

1. **Inter-continental delivery**: Vehicles are produced and sold in two different continents. It is required to transport these vehicles on a deep-sea vessel from an export to an import port and finally distribute them after reaching the destination market.

2. **Intra-continental delivery**: Assembly plants and markets are located within the same continent but in different countries. There might be large distances between plant and final distribution compound which have to be overcome mainly by using trains or trucks. However, coastal or short-sea vessels might be another option here.

3. **Domestic delivery**: Vehicles are sold in their production country and the distances from plant to final distribution compound are rather short. The dominant means of transport will be the truck in that case, however there might be as well trains in use for some cases.

Obviously, these scenarios do not reflect reality well in all cases. There are countries like the USA, Russia, India or China which have on their own a similar dimension like a continent and there might be other reasons to focus on any means of transport like a poor street infrastructure or larger areas of mountainous terrain. However,

choosing only these three simple scenarios supports the understanding of the benchmark instances and still produces a mixture of different delivery strategies observed in reality.

Starting with the inter-continental scenario, we first need to generate import and export ports for each exporting plant and importing market. Datasets like the "World Port Index" provided by NGA (2020) can be used to determine the location and size of the world's sea ports. For export, we randomly chose two ports for each plant from the five closest ones. Distances are always measured as great-circle distance using the Haversine formula (Robusto 1957). For imports, we determine the two closest ports for each delivery zone that will receive imported vehicles. We randomly keep only one-quarter of these ports for each market but at least two of them. Whenever randomly selecting a port, the size of that port (measured by the classification of NGA (2020)) corresponds to its probability to be chosen. Finally, we expect all vehicles to require a service right before and after being shipped on a vessel. These services will be offered at each port (modeled as predecessors and successors of the vessel relation).

Besides ports, also compounds are an essential part of a transport network, especially for the final distribution of vehicles. We will generate such compounds within the sales markets by clustering the centroids of the delivery zones like we did before with the dealers, but use a distance threshold of 1000 km in that case. These compounds can be used in all three delivery scenarios. Furthermore, plants can also be used as a distribution compound. For each delivery zone that receives vehicles flagged for modification, we chose a random compound of the three closest to offer such modification. If a plant is among that group, no additional compound will be chosen, since that plant will handle all modifications. Finally, for each market, we select randomly one or two compounds that might be used as a VHC. The aforementioned amount of vehicles that will be build-to-stock can be generated using these compounds as final destination.

Connecting all the nodes generated above will complete our transport network. For the inter-continental case, we connect each plant to the closest two sea ports and distribute the produced vehicles uniformly to them. If there is a compound close to a plant, 10 % of the vehicles will first be routed to that compound and to the closest port afterwards but only if the overall travel distance to the port does not increase more than 20 %. For the import market, we need to distinguish between vehicles with or without modifications. The latter will be transported

directly from the importing port to the delivery zone unless there is a compound close by. In this case, all vehicles to this specific zone will visit that compound first. In case of a required modification, it is necessary to visit a VMC prior to delivery. For each zone we chose the VMC minimizing the total distance to the port. Finally, import and export sea ports are connected. The connections supporting the intra-continental case are quite similar. The plants will be connected to the closest port and compound, while 30 % of all vehicles will be routed to the port. From the importing port again there will be direct transports to the delivery zone or VHC and deliveries using a distribution compound. If a vehicle has been routed to a compound close to the plant it will be forwarded to a VHC, a distribution compound or a VMC, depending on the requirements. For domestic scenarios, the transport network will be simpler. There will be a direct transport from the plant to the delivery zone or the VHC or alternatively an intermediate compound will be used. The vehicles are distributed uniformly to these two options. In case a modification is required, there will always be an intermediate compound. Having all of these relations in place, we can now uniformly distribute the remaining vehicle volume C_N to the ports and compounds used on the routes to the desired markets.

Although we described which links will exists within a transport network of a benchmark instance, we did not specify any characteristics of that links (except start and end nodes). Obviously, all links connecting sea ports will be operated by RoRo vessels. Regardless if it is a deep-sea or short-sea vessel, we assume an average speed S_{Vessel} uniformly distributed from 15 kn to 23 kn ($\approx$ 28 km/h to 43 km/h). Using the great-circle distance d_a (in meters) between start and end node the traversal time function for the corresponding arc a can be calculated to

$$\tau_{\text{Vessel}}(a,t) = \left\lceil \frac{1.5 d_a}{44\,448 S_{\text{Vessel}}} \right\rceil \tag{13.3}$$

with 44 448 being the conversion factor for kn to m/d. Furthermore, the distance d_a is increased by 50 % to incorporate the typical non-linear travel path of a vessel along the coast-lines. Similar formulas can be derived for truck and train relations. However, especially for trucks not the distance to travel but regulations on driving times are the dominant factor. We assume for a truck to be able to travel uniformly distributed between $S_{\text{Truck}} = 400$ km/d to 700 km/d. Also here the distance will be increased to take non-linear paths into account, but we expect a factor of 25 % to be sufficient in that case. Multiplying S_{Truck} with a factor of 1000 to convert

its unit from km/d to m/d, the resulting formula has the form

$$\tau_{\text{Truck}}(a,t) = \left\lceil \frac{1.25 d_a}{1000 S_{\text{Truck}}} \right\rceil \tag{13.4}$$

Typically, freight trains are able to travel with a similar speed to trucks, so we assume $S_{\text{Train}} = S_{\text{Truck}}$. However, rail tracks will be closer to a straight connection, so we assume a factor of only 10 % in the formula

$$\tau_{\text{Train}}(a,t) = \left\lceil \frac{1.1 d_a}{1000 S_{\text{Train}}} \right\rceil \tag{13.5}$$

The final transport to a delivery zone will always be operated by a truck carrier. Connections between compounds, ports and plants might be served either by truck, by train or both. With increasing distance between start and end node, we linearly increase the probability for a train connection and decrease the probability for trucks. We assume that all relations longer than 2000 km will be operated by train only. While trucks can be scheduled each day of the week, this is not possible for trains or vessels. Typically, these means of transport follow a fixed schedule. We assume trains to be randomly available 2 to 3 days per week, while a vessel will depart only 4 to 5 times per month. Transports will never start on Sundays and will be enlarged by one day for each Sunday that is contained in the traversal time (except for vessels). Finally, not only the time a vehicle travels but also the time spent in any compound or port needs to be considered. We assume that a vehicle resides within each compound or port for at least one day before moving to the next location. If modifications are required, we will randomly add additional 1 to 3 days at the VMC compound. Services received at the importing or exporting port will always take one additional day. All that information can be aggregated to carry out an estimation of T_{end} considering an inter-continental delivery including export and import compounds, required modifications and worst-case distances and schedules for all means of transport.

Required transport capacities can be derived by calculating the expected usage when moving the generated vehicles along their predetermined routes. Of course, traversal times and schedules must be taken into account during that step. We will neglect compound storage capacities within all benchmark instances because our (preliminary) experiments indicated that they are not the limiting factor of such a network. In contrast, capacities for modifications and services within a compound or port will be considered. Furthermore, to generate more realistic instances,

we will limit all capacities to a random maximum amount per day. For trucks, that limit will be 200 to 300 units per day, for train 600 to 1200 and for vessel transports we consider a maximum of 4000 to 6000 vehicles. Similarly, the capacity for modifications will be randomly restricted between 50 to 100 vehicles per day, for port services it will be 400 to 600. Since we assume the plant's production to vary over time, it is likely that hot-spots will occur within the transport network with a high vehicle load. Typically, distribution planners are aware of such fluctuations and adopt capacities accordingly. Imitating this behavior, we will double the capacity limits of the 20 % transports, modifications and port services with the highest load. Simulating the vehicle flow while tracking the usage of the network and considering that usage as the capacities of a benchmark instance will lead to tightly constraint instances. Early experiments revealed a strong correlation between capacity scarcity and the ability of our solution methods to find any feasible solution. Since we aimed at generating benchmark instances with diverse levels of difficulty, we allowed to relax that capacity limitations by introducing parameter $R \in \mathbb{R} \geq 1$ of the instance generator that will be used as a scale factor for all transport, VMC and service capacities. Although we assumed that a vehicle will travel on its predetermined route, in the benchmark instances we will allow all routes that might take a vehicle from its start to its end node by defining the arcs set A^k accordingly. This might offer an opportunity to the solution methods to optimize the vehicle flow by consolidating vehicles.

Besides traversal times, schedules and capacities, there are also different kinds of costs that need to be specified. Typically, transport costs vary by the distance traveled, the transported amount of vehicles and by their size. Thus, we randomly classify all generated vehicles to be of small, medium or large size. All per-unit costs described in the following will be used only for medium-sized vehicles, while small and large vehicles receive a 30 % discount or surcharge. For truck transports, we expect distance independent costs of 30 to 40 Euros[2] for each vehicle and additional costs of 0.20 to 0.25 Euros per kilometer per vehicle. In contrast, for vessel relations the costs will not only depend on the distance but also on the costs for (un)loading in the ports and the customs clearance process. We assume the costs per vehicle to linearly scale with the distance from 400 to 800 Euros for short-sea and from 800 to 1600 Euros for deep-sea connections. While the cost structure for trucks and vessel is rather simple, it will be more complex for train relations. Typically,

[2]We use Euro as currency here, however from a mathematical point of view the currency is negligible as long as all costs are specified in the same currency.

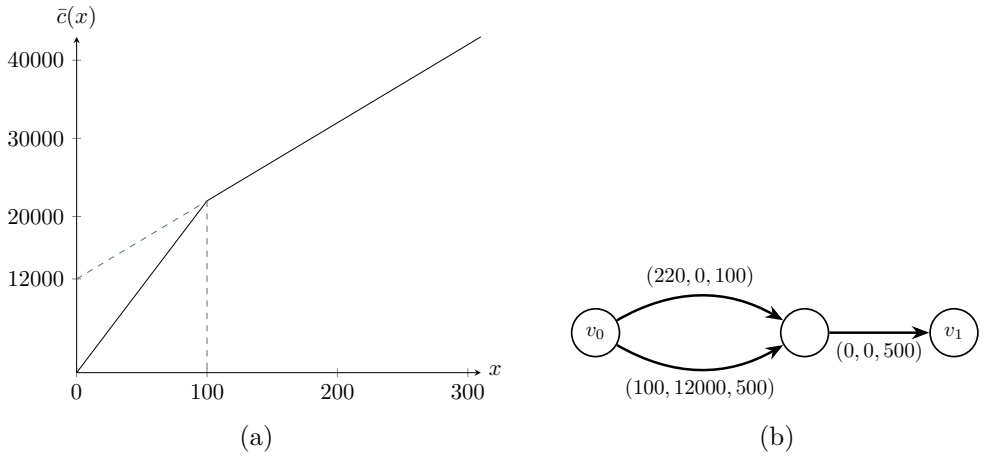


Figure 13.5: Example of a piecewise-linear concave cost function (a) and its corresponding sub-graph (b) showing the overall transport costs of train relations. Shown is a relation with length of 1000 km, distance independent costs of 120 Euros and distance dependent costs of 0.10 Euros per kilometer. Thus, costs per vehicle will be $120 + 1000 \cdot 0.10 = 220$ Euros. Fixed-costs for a train will be chosen to 12 000 Euros and the maximum capacity is 500 units. The arcs are labeled with $(c(a), \tilde{c}(a), u(a))$.

for larger vehicle volumes that need to be transported (which is not unusual for compound-to-compound relations), train transports will be ordered with a fixed amount of wagons or as a full train. These costs will be fixed-costs that occur whenever a relation is used independent of the transported volume. We expect the break-even-point to be reached for 100 vehicles. For fewer vehicles, a price model similar to trucks with 120 to 150 Euros per vehicle independent of the traveled distance and 0.10 to 0.15 Euros per kilometer per vehicle will be used. As soon as more than 100 vehicles will be transported on a day, an entire train will be ordered and only the distance dependent costs must be paid per vehicle. The fixed-costs that will be charged for the train depend on the aforementioned per-unit costs. The overall cost structure should result in a piecewise-linear concave cost function. An example of such a cost function and its representation as a sub-graph following the explanation from Section 5.1 is shown in Figure 13.5.

Additionally, not only transports but also handling within a compound, port or VMC will be charged. While storage within a compound is assumed to be cheap with 1 to 2 Euros per vehicle per day, it is typically more expensive at ports with 5 to 6 Euros per vehicle per day. Furthermore, for port services an additional amount

of 20 to 40 Euros must be paid. Modifications within a VMC are expected to be charged with 100 to 150 Euros. Finally, next to these monetary costs, there are also lateness costs within our problem model that will be paid in form of reputation. From the calculation of the traversal times, it is trivial to derive a TDD for each vehicle. However, this is only reasonable for vehicles that need to be delivered to a dealership. Vehicles that are destined for a VHC will not have a TDD at all. For each day a vehicle is delivered late or early, we will charge at least additional 100 to 200 Euros. However, we want to be able to generate instances where customer satisfaction is weighted differently in relation to transport costs. Therefore, we introduced scale parameter $L \in \mathbb{R}_{\geq 0}$ of the instance generator that can be used to adopt the lateness costs of all vehicles accordingly.

Although we mentioned some details before, the internal structure of plants, compounds or ports has not been described so far. Figure 13.6 shows the structure used by the instance generator that is able to support all the use cases explained above. Vehicles will always start at the dedicated source node v_s and end at node v_v in case of a VHC delivery. Furthermore, there are explicit input and output nodes v_i and v_o that can be used to enter or exit the compound but also for waiting at the compound's storage area. Since there is an additional storage arc from node v_m to v_o, it is guaranteed that a vehicle will stay at a compound for

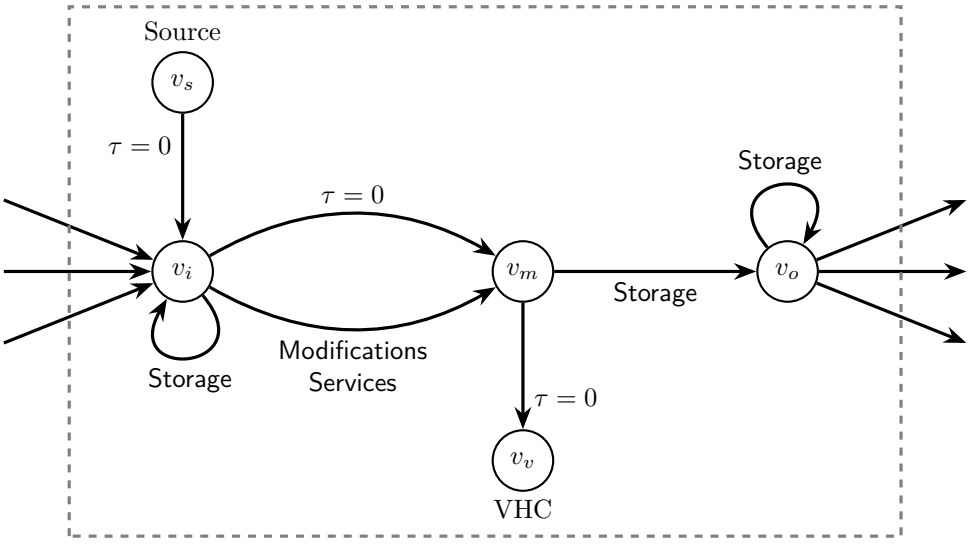


Figure 13.6: Graph modeling the internal structure of all plants, compounds and ports within our randomly generated instances.

at least one day. Vehicles requiring modifications or port services may use the distinguished arc from node v_i to v_m. All arcs labeled with $\tau = 0$ can be used without consuming any time. The structure can be simplified for some nodes, e.g., if no modifications or services are offered by a compound the nodes v_i and v_m can be merged. Furthermore, source node v_s and VHC node v_v are optional if no vehicle starts or ends at that compound. The instance generator will always create the simplest structure that matches all use cases of a node.

With the instance generator at hand, we will use it to generate benchmark instances for the computational study presented in the remainder of this chapter. Generating instances of different sizes has been one of the key points that motivated the implementation of the generator. The amount of commodities C and the length of the production planning horizon D are the main drivers for the size of an instance. Thus, we will vary that input parameters accordingly from 1000 commodities produced within 5 days up to 500 000 commodities produced within 30 days, which is considered to be close to the size of a real-life scenario. During our first computations, we figured out that some heuristic approaches are facing difficulties in finding an initial feasible solution when the capacities of the arcs are tight. For this, the parameter R has been introduced to relax these capacity limitations. We will increase this parameter from tight instances with $R = 1.0$ to relaxed ones with $R = 1.2$, which results in 20 % excess capacities. Finally, we observed that the lateness costs have a serious impact on the overall computation times of some solution approaches. As a consequence, we will vary parameter L from having no lateness costs at all with $L = 0$, over balanced costs with $L = 1$ to weighting lateness costs more important than transport costs with $L = 10$. Table 13.1 lists all 72 parameter combinations we explored with its assigned class name. For each combination, we generated 5 benchmark instances by varying the initial randomness seed. Some figures regarding these instances can be found in the table as well.

Table 13.1: Listing of all explored random instance classes. The name indicates the size and configuration of capacity tightness and lateness costs. The first ranging from tightest (TT) and tight (T) to relaxed (R) and most relaxed (RR), the second from having no lateness costs to balanced (L) and dominant (LL) costs. The amount of arcs and nodes is related to the non-time-expanded graph. For each class the minimum and maximum values over the 5 generated instances are listed as details.

	Parameters				**Instance details**					
Class name	C	D	R	L	$\|V\|$	$\|A\|$	T_{end}	Zones	Comp.	Ports
C1D5TT	1000	5	1.0	0	59–160	141–360	71–101	9–31	2–10	4–12
C1D5TTL				1	48–185	112–436	65–95	3–25	2–10	6–22
C1D5TTLL				10	85–170	204–395	78–113	13–29	4–10	8–16
C1D5T			1.01	0	54–134	132–310	73–85	5–22	2–8	4–16
C1D5TL				1	90–251	213–553	87–129	14–60	3–18	8–19
C1D5TLL				10	100–156	226–365	69–129	15–27	4–8	11–13
C1D5R			1.05	0	97–246	245–551	65–124	16–61	3–16	7–12
C1D5RL				1	27–155	63–364	70–156	1–29	1–11	4–14
C1D5RLL				10	42–189	102–447	66–117	4–43	2–13	4–12
C1D5RR			1.2	0	60–150	150–363	53–168	7–23	1–11	4–10
C1D5RRL				1	70–193	167–424	74–101	10–49	3–11	6–14
C1D5RRLL				10	85–137	198–350	68–92	7–32	2–8	6–12
C5D5TT	5000	5	1.0	0	96–303	247–747	76–144	17–69	6–14	4–25
C5D5TTL				1	292–578	687–1303	107–156	60–125	16–34	14–42
C5D5TTLL				10	111–410	272–958	77–167	16–102	4–26	10–28
C5D5T			1.01	0	125–380	307–839	73–131	20–125	4–23	5–26

	C	D	R	L	$\lvert V\rvert$	$\lvert A\rvert$	T_{end}	Zones	Comp.	Ports
C5D5TL				1	122–406	297–951	80–129	15–73	5–22	12–31
C5D5TLL				10	88–494	232–1169	93–143	17–89	5–32	4–34
C5D5R			1.05	0	79–290	187–674	69–177	15–58	2–18	5–18
C5D5RL				1	265–365	651–873	85–123	43–81	10–25	10–23
C5D5RLL				10	201–315	483–721	88–138	37–80	11–16	10–22
C5D5RR			1.2	0	107–430	282–971	71–114	19–95	4–23	6–27
C5D5RRL				1	96–391	213–958	85–112	18–69	3–23	10–29
C5D5RRLL				10	286–483	688–1124	99–164	66–113	17–29	6–36
C10D10TT	10000	10	1.0	0	195–501	479–1227	111–151	38–133	11–24	10–24
C10D10TTL				1	127–518	313–1248	91–129	21–147	5–24	4–18
C10D10TTLL				10	161–436	403–1069	84–165	41–103	7–26	5–27
C10D10T			1.01	0	167–428	408–1042	101–167	27–115	8–25	8–29
C10D10TL				1	104–753	252–1761	76–161	16–171	4–45	7–47
C10D10TLL				10	107–421	260–1016	83–157	13–117	5–22	11–28
C10D10R			1.05	0	122–420	305–963	76–132	22–102	3–24	9–29
C10D10RL				1	146–559	349–1352	86–181	24–128	5–31	12–40
C10D10RLL				10	156–605	386–1404	80–140	31–154	7–36	8–24
C10D10RR			1.2	0	279–449	658–1064	99–122	69–112	16–22	10–25
C10D10RRL				1	173–728	434–1686	103–155	36–182	10–33	8–42
C10D10RRLL				10	256–582	591–1444	88–161	63–156	13–36	8–25
C50D10TT	50000	10	1.0	0	572–2409	1553–6728	119–157	112–390	20–83	21–113

	C	D	R	L	$\|V\|$	$\|A\|$	T_{end}	Zones	Comp.	Ports
C50D10TTL				1	491–2080	1452–5693	115–191	93–399	19–80	19–97
C50D10TTLL				10	1101–1827	2917–5028	154–171	204–349	43–73	53–85
C50D10T			1.01	0	1118–2278	3035–6338	162–174	208–390	35–91	44–93
C50D10TL				1	848–1837	2224–5109	138–210	170–336	29–66	36–83
C50D10TLL				10	805–1705	2271–4859	157–201	151–291	30–61	33–73
C50D10R			1.05	0	930–2344	2841–6588	130–182	136–400	27–75	38–108
C50D10RL				1	597–1925	1842–5450	141–183	97–350	17–64	12–74
C50D10RLL				10	1309–2096	3597–5692	163–177	195–374	39–66	57–105
C50D10RR			1.2	0	1079–2135	2861–5874	135–184	189–420	41–84	41–92
C50D10RRL				1	1425–2301	4033–6524	139–168	272–400	40–88	51–101
C50D10RRLL				10	499–2156	1415–6160	138–179	99–367	18–70	19–101
C100D30TT	100000	30	1.0	0	538–1158	1565–3160	135–191	111–214	19–47	17–56
C100D30TTL				1	1294–1904	3389–5178	154–196	282–405	44–80	70–103
C100D30TTLL				10	487–1485	1316–3909	125–211	108–316	19–59	20–64
C100D30T			1.01	0	481–1756	1357–4828	125–193	64–350	15–72	31–77
C100D30TL				1	402–1901	1106–5164	174–194	60–406	12–72	26–83
C100D30TLL				10	502–1372	1479–3623	177–197	76–319	16–62	30–63
C100D30R			1.05	0	382–1149	930–3106	140–179	96–239	17–46	15–57
C100D30RL				1	1140–2293	3076–6093	138–198	193–497	45–95	53–87
C100D30RLL				10	600–1628	1727–4468	173–195	117–316	26–60	22–98
C100D30RR			1.2	0	690–1248	1871–3399	139–189	142–250	22–53	36–64
C100D30RRL				1	628–2011	1656–5362	145–186	126–429	22–78	30–86

	C	D	R	L	$\|V\|$	$\|A\|$	T_{end}	Zones	Comp.	Ports
C100D30RRLL				10	566–1793	1541–4730	142–193	105–430	18–75	24–55
C500D30TT	500000	30	1.0	0	3896–8376	13108–27733	203–224	460–1081	90–170	107–186
C500D30TTL				1	4077–9126	13397–29906	196–222	491–1140	81–174	102–200
C500D30TTLL				10	6361–8574	20361–27437	206–229	767–1140	130–186	154–191
C500D30T			1.01	0	4560–8922	15707–29029	212–226	496–1148	81–193	106–200
C500D30TL				1	3791–6992	12079–23165	204–238	541–859	96–141	86–163
C500D30TLL				10	5467–8525	18336–27979	205–226	566–1131	89–175	140–195
C500D30R			1.05	0	6707–9434	21711–30775	205–232	863–1219	132–192	154–216
C500D30RL				1	4023–7728	13080–24992	203–216	473–982	79–151	118–178
C500D30RLL				10	4754–7542	15551–24800	190–213	583–914	91–151	112–175
C500D30RR			1.2	0	5595–7554	18387–24914	209–234	737–1024	127–172	126–199
C500D30RRL				1	4099–6062	13417–20444	192–236	484–688	86–119	114–141
C500D30RRLL				10	2669–6407	9052–21073	193–214	264–780	51–120	86–146

13.2 Hardware and Software

The computational study has been carried out on the CLAIX 2018 high-performance computing cluster of RWTH Aachen University. Each node of that cluster is composed of an Intel HNS2600BPB compute module and two CPUs of type Intel Skylake Platinum 8160 (2.1 GHz). This CPUs are equipped with 24 cores each (so 48 cores per node). The main memory is limited to 192 GB per node.

The cluster is operated by CentOS Linux (v7.7.1908, Kernel 3.10.0-1062.12.1.el7, 64-Bit) and Slurm Workload Manager (v18.08.7), a job scheduling and computing resource management utility. As all algorithms were implemented in Java, we used OpenJDK (v1.8.0_242-b08, 64-Bit) as runtime environment. The code has been compiled using Oracle's HotSpot JDK (v1.8.0_144-b01, 64-Bit). Whenever an out-of-the-box solver has been required, we utilized Gurobi Optimizer (v9.0.0, 64-Bit) for that task. Our branch-and-price-and-cut approach has been build on SCIP (v7.0.0, 64-bit), using Gurobi as LP-solver. SCIP and all plugins that have been implemented by us (e.g., pricer, branching rule, separator) has been compiled with cmake (v3.10.1) and gcc (v7.3.0).

All experiments measuring computing times have been carried out using Java Microbenchmark Harness (JMH, v1.23). That framework forks a dedicated Java virtual machine (JVM) for each experiment and performs a few warm-up iterations followed by several measurement iterations. We will specify in detail the amount of performed iterations in the explanation of the individual experiments. Please note that all reported computing times are the average of these measurement iterations. This should sufficiently ensure, that the experiments are not distorted by external factors like the system load (we shared the computing cluster with other researchers).

Within our Java implementations, we relied on a few libraries that we want to list in the following:

- Collection data structures like sets, lists, queues, maps or heaps have been taken from fastutils v8.3.1 (`http://fastutil.di.unimi.it/`).
- For multi-maps and collection-valued maps we utilized commons-collections v4.4 (`https://commons.apache.org/proper/commons-collections/`).

- Reading from and writing to files has been supported by commons-io v2.6 (`https://commons.apache.org/proper/commons-io/`).
- StringUtils and various data structures for pairs and tuples have been used from commons-lang v3.9 (`https://commons.apache.org/proper/commons-lang/`).
- FastMath, random number generators and probability distributions haven been imported from commons-math v3.6.1 (`https://commons.apache.org/proper/commons-math/`).

Not all of these libraries are used within performance critical regions of the implementation but a few of them are, so there might be some performance impact that cannot be neglected.

Our computational study as a whole required more than 210 000 computing hour equivalents (core hours), which would have been impossible to acquire using a usual workstation. Fortunately, we were allowed to use the "RWTH Compute Cluster" for that task. All experiments were performed with computing resources granted by RWTH Aachen University under project "rwth0560". We would like to thank all involved persons for that opportunity and the great support.

13.3 Computing Limitations in Practice

Although we used a high-performance computing cluster to carry out our computational study, we do not assume that such hardware resources are required to apply our algorithms to practice. For the largest instances, we simultaneously used at most 8 cores and 96 GB of main memory. While that CPU performance is given in most modern workstations, the amount of main memory typically is not. However, in our experience it does not pose a problem for the IT department of an automobile manufacturer to provide such hardware resources. Alternatively, all of the commercially available cloud-computing platforms, like Amazon Web Services, Microsoft Azure or Google Cloud Platform, can provide computing services of that scale. Hence, we do not expect hardware resources to be a limiting factor regarding the applicability of our solution approaches.

The situation is different in regard to computing times. We used various time limits during our experiments with a maximum of 12 hours which we expect to be

a typical value applied in practice. This allows for two optimization runs per day, which in our experience is a good balance between solution quality and actuality of the data used for optimization. Nevertheless, this parameter highly depends on the practical use case. It might be sufficient to perform one optimization run per week. In contrast, if the data used for optimization is changing quickly, more than two runs per day might be required, to allow operations to carry out the plan. Hence, the configuration of that parameter must be a trade-off between solution quality and practicability within a business context.

13.4 Performance Metrics

When evaluating or comparing the performance of different algorithms, a metric for measuring the performance is required. Depending on the experiment, we will measure either the quality of the solutions found by an algorithms or the computational effort spent to find that solution (or both). While the first typically is measured by some form of distance to a proven optimal solution or a known lower bound, the latter is given as amount of computing time or iterations. We will give some more details on the used metrics in the following.

Optimality and MIP gap Given is an optimal solution $\overset{*}{x} \in X$ and another solution $x \in X$ for an arbitrary optimization problem with solution space X. The objective function $f : X \to \mathbb{R}$ calculates the value of any given solution. The *(relative) optimality gap* is defined as

$$\sigma_{OPT}(x) = \frac{|f(x) - f(\overset{*}{x})|}{|f(\overset{*}{x})|} \tag{13.6}$$

Please note that $\sigma_{OPT}(\overset{*}{x}) = 0$ always holds. In contrast, there is also the notion of a *MIP gap*. Given is a (mixed) integer program defined on polyhedron $X_I \subseteq \mathbb{Z}^n_{\geq 0}$ and its corresponding LP-relaxation with polyhedron $X_L \subseteq \mathbb{R}^n_{\geq 0}$. The objective function of these programs is $f : \mathbb{R}^n_{\geq 0} \to \mathbb{R}$. For any given feasible integer solution $x \in X_I$ the MIP gap is defined as

$$\sigma_{MIP}(x) = \frac{|f(x) - f(\overset{*}{x})|}{|f(\overset{*}{x})|} \tag{13.7}$$

with $\overset{*}{x} \in X_L$ being an optimal solution of the LP-relaxation.

If $X_L = \text{conv}(X_I)$ is the integer hull of X_I, the MIP gap will be $\sigma_{MIP}(\overset{*}{x}_I) = 0$ for any optimal integer solution $\overset{*}{x}_I \in X_I$. Please note that this formula can be used for any lower bound other than an optimal solution of the LP-relaxation, too. Although both definitions of $\sigma_{OPT}(x)$ and $\sigma_{MIP}(x)$ look quite similar, there is a huge difference when interpreting the results of our computational study using these definitions. The first compares a solution against an optimal solution while the second uses a lower bound of the optimal solution value without any assumptions on the quality of that lower bound. Whenever it is not clear which of the Equations (13.6) or (13.7) is used in a certain context, we will specify that explicitly.

Computing Time and Iterations There have been different definitions of *computing time* used in the literature. Most commonly used are real time (wall clock time) passed from the start of a measurement to its end and CPU time, counting all the CPU cycles performed during the measurement (Barr et al. 1995). The latter is usually split into user and system CPU time, distinguishing the CPU cycles spent by the user program from calls to the kernel of the operating system. While CPU time only measures times when the CPU is actively processing, the real time measurement will also contain times when the CPU has been idle, e.g., waiting for a resource like an I/O device. In the following, computing time will always be equivalent to the real time passed. However, I/O intensive tasks, like reading and parsing a problem instance from hard-disk will not be contained in this time. Our measurements always start after the problem instances have been loaded to main memory. Nevertheless, all preprocessing steps will be part of the measurement unless otherwise stated.

Instead of reporting the real time passed, in some charts we will present the amount of *iterations* spent by an algorithm. In contrast to time measurements, counting iterations has the major benefit that it is independent of the computing environment and the system load. A deterministic algorithm will always find the same solution to a problem instance with the same amount of iterations. All our implementations are deterministic unless otherwise stated. Even if there are randomized steps within an algorithm, these will be performed by a pseudo-random-number-generator that is seeded with information from the problem instances. Thus, for the same instance the algorithm will be deterministic. The definition of a single iteration depends on the algorithm, but typically it is related to the most outer loop. If it is ambiguous, we will give the required details in the following.

13.5 Visualizing Performance Measurements

Typically, the performance metrics discussed above are gathered from large benchmark datasets leading to an even larger set of measurement values. For presenting these values, they are usually aggregated to averages, cumulative totals or medians and quartiles. But there are also other options to visualize such sets of measured values, which will be explained in the following.

Performance Profiles Dolan and Moré (2002) suggested to use *performance profiles* for visualization when comparing algorithms. We will shortly recap their ideas and definitions. Given is a set $\mathcal{B}$ of benchmark instances and a set $\mathcal{S}$ of solvers. For each instance b and each solver s, we define $t_{b,s}$ as the computing time required to solve instance b by solver s. Comparing the performance of solver s on instance b with the best performance of all solvers leads to the definition of the *performance ratio*

$$r_{b,s} = \frac{t_{b,s}}{\min\{\, t_{b,\tilde{s}} \mid \tilde{s} \in \mathcal{S} \,\}} \tag{13.8}$$

with $r_{b,s} \in [1, r_{max}]$ and r_{max} being the performance ratio if the solver was unable to solve the instance. Calculating the cumulative distribution function for the performance ratio finally results in the definition of the performance profile

$$\rho_s(\xi) = \frac{|\{\, b \in \mathcal{B} \mid r_{b,s} \leq \xi \,\}|}{|\mathcal{B}|}. \tag{13.9}$$

This function specifies the probability for solver $s \in \mathcal{S}$ that its performance ratio $r_{b,s}$ is within a factor $\xi \in \mathbb{R}$ of the best ratio. Please note that $\rho_s(1)$ is the probability that solver s will be the best for a certain instance, while $\lim_{\xi \to r_{max}} \rho_s(\xi)$ gives the probability that the solver can successfully solve an instance. We will use performance profiles in the remainder of this work whenever the performance of different algorithms or variations of the same algorithm should be evaluated and compared to each other.

Box Plots Besides performance profiles, we also use box plots to visualize the statistical distribution of measured values. Whenever we use these plots the boxes are aligned to the 25th and 75th percentile of the values and contain both the median (solid) and average value (dashed). Unless specified differently, the whiskers represent all measurements within a distance from the upper / lower

quartile of 1.5 times the inter-quartile range (IQR). All outliers are visualized explicitly. Figure 13.7 shows an example of such a box plot.

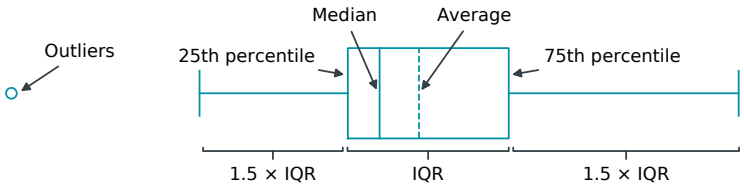


Figure 13.7: Example of a box plot visualizing all relevant parameters of that type of chart, like the IQR, median or average values.

14 | Path Search Algorithms

In the following, we are going to evaluate and compare the performance of the path search algorithms presented in Chapter 6. We start our study of the resource-constraint case with the adapted algorithm of Feillet, continue with the adapted Boost algorithm and end with the Delayed Dominance algorithm. For all approaches, we examine both a plain implementation and further extensions for node reachability calculation, path cost estimation and A*-like node or label sorting. We compare the best variants of all implementations to select an algorithm that will be used for path search during the remainder of our computational study. Furthermore, we give insights into the impact of negative-cost cycles on the overall performance. Our study ends with the evaluation of the non-resource-constraint case for that we will utilize the A* algorithm.

We used all small to medium size instances of our randomly generated benchmark set for the evaluation (classes C1D5 to C50D10). Having 5 instances for each class results in overall 240 instances. As mentioned earlier, the path search algorithms cannot be used to solve the PCIMCFND directly, but they are a utility that will be embedded in other solution approaches. Hence, trying to solve our benchmark instances would not make sense. Instead, during a path search experiment we select (pseudo) randomly 500 commodities from an instance and calculate the shortest path from origin to destination for all of them. We assume that this approach will be similar to the expected application of the algorithms.

The computing time of each experiment consists of the time required for preprocessing and the accumulated time of all 500 path searches. Each experiment will be measured 5 times by JMH after performing 3 warm-up iterations within the same JVM. The average of these measurement iterations will be reported as the resulting computing time. During all runs the heap memory of the JVM has been limited to a maximum of 8 GB, since that has been sufficient for all experiments. In cases where the convergence of an algorithm to the shortest path has been examined, we were only interested in the amount of required iterations of an implementation

and not in computing time. Thus, we did not use JMH in that case. Furthermore, all convergence measurements have been performed using the largest instances of class C500D30, since convergence required the most iterations here which resulted in more appealing charts. Please note that similar behavior has been observed in all classes of instances, albeit we do not present all the details.

When reporting performance results in the following, we will use a naming scheme to easily identify the variant of an algorithm that has been evaluated. While the plain implementations will be called *Feillet*, *Boost* and *Delay*, we append the following suffixes to identify extensions made to the plain implementation:

- **GR**: In preprocessing a depth-first search algorithm will calculate the connectivity of all pairs of nodes $u, v \in V$ using arcs A of the entire graph $\mathcal{D} = (V, A)$ of the benchmark instance. Connectivity will be calculated only once for all commodities. During path search, these results will be used to discard labels that cannot fulfill arc-requirements or precedence relations any longer. Furthermore, arcs that will not lead to the destination node are filtered too.

- **R**: This extension is similar to GR but instead of determining connectivity only once the calculation will be performed separately on the sub-graph $\mathcal{D}^k = (V^k, A^k)$ for each commodity. This might increase preprocessing times but will be more accurate.

- **C**: While pruning by bounds will only use a label's current path costs r_c in all plain implementations, this extension strengthens the pruning step by using path cost estimates from a label's resident node to the destination node. The estimates are calculated by an enhanced Bellman-Ford algorithm (single-sink shortest path) presented by Bannister and Eppstein (2012), incorporating the "walk to the root" cycle-detection introduced by Cherkassky and Goldberg (1999). The algorithm runs on the support graph deduced in Section 6.9. If the estimated total path costs of a label from its origin via its resident node to its destination exceed the current upper bound, the label will be discarded.

- **CX**: Instead of estimating the remaining path costs using the estimated costs from a label's resident node to the destination node only, this extension will additionally consider all unfulfilled arc-requirements and successor relations by calculating the minimum estimated path costs of all their permutations

(see Section 6.9 for details). Please note that in contrast to extension C, this requires an all-pairs cost estimate. We use the classical Floyd-Warshal algorithm for this task (Cormen et al. 2013, pp. 693–700), as it can be easily enhanced to detect negative-cost cycles. Nevertheless, if there are no negative arc costs available in the network, we will instead use Dijkstra's algorithm utilizing a Fibonacci heap for each node $v \in V^k$. That will be considerably faster on sparse graphs (Cormen et al. 2013, pp. 658–664).

- **CR / CXR**: During a shortest path calculation the connectivity of nodes will be determined implicitly. Thus, we can easily combine extensions C and CX with the node reachability feature described in R without any further preprocessing. Please note that this requires an all-pairs shortest path computation even for the CR case.
- *****: Whenever path cost estimates have been collected in a preprocessing step, we can use that estimates to sort the queue of labels or nodes in an A*-like fashion. We will integrate that ordering scheme in variants C, CX, CR and CXR. Please note that the ordering scheme will use the enhanced cost estimates for CX and CXR and the simpler ones for C and CR.

After having introduced above naming scheme we are going to evaluate and compare all variants of the different algorithms in the following.

14.1 Adapted Algorithm of Feillet

We started the computational study on our path search algorithms by evaluating the impact of extensions R and GR on Feillet's algorithm. Figure 14.1a shows the performance profile of these two extensions compared to the plain implementation. Both extensions show similar performance, with FeilletR being slightly better. Furthermore, both enhancements outperform Feillet, which fits to our expectation that R and GR implementations discard considerable more labels, as they ignore (most) paths that will not lead to the destination node. We hoped preprocessing times not to be dominant compared to the overall computing times, and that could be confirmed as Figure 14.2 indicates. Preprocessing requires up to 3.4 % of the overall computing time with an average of 0.8 % for R and 0.4 % for GR. Surprisingly, FeilletR performs better, although it requires more time for the majority of node reachability calculations (not only in average). Being more

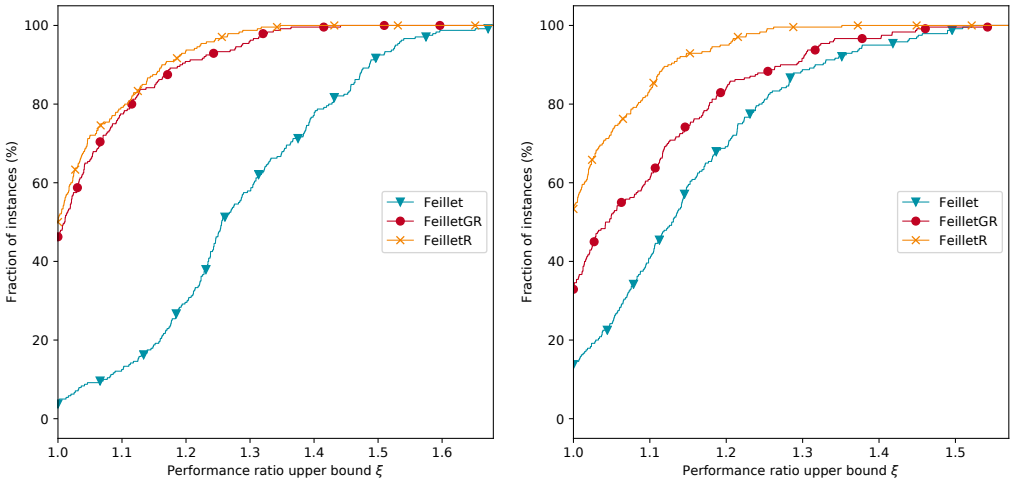


Figure 14.1: Performance profiles of Feillet's algorithm comparing the implementations that add node reachability calculations. While (a) shows the benchmark with arbitrary vehicles, in (b) only VMC vehicles have been used.

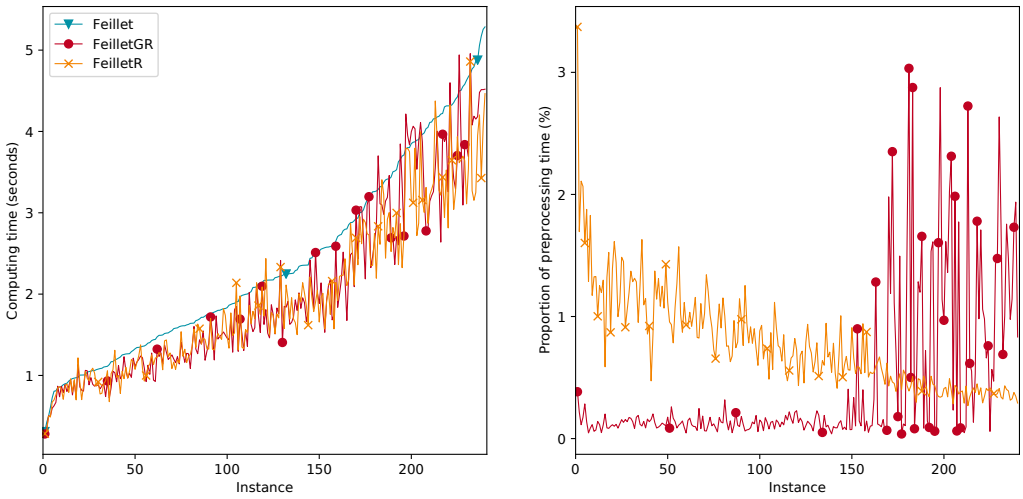
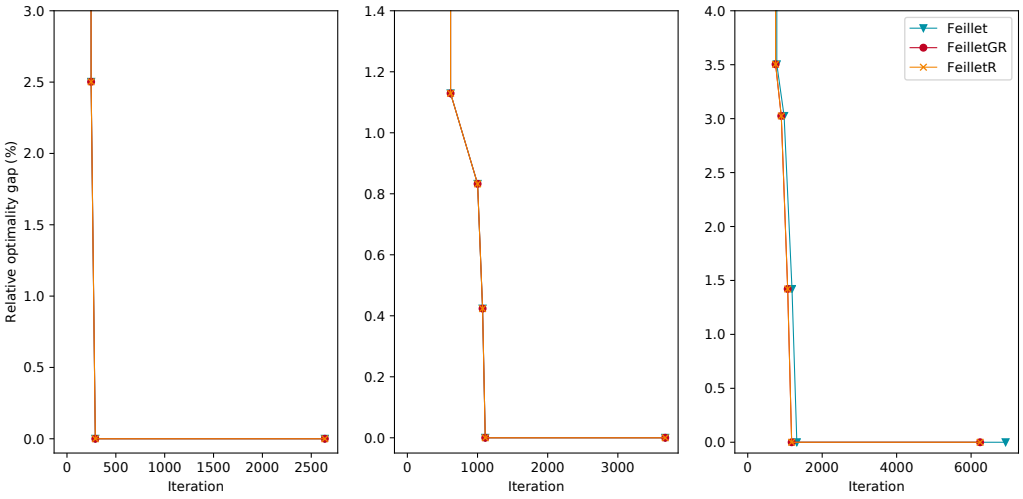
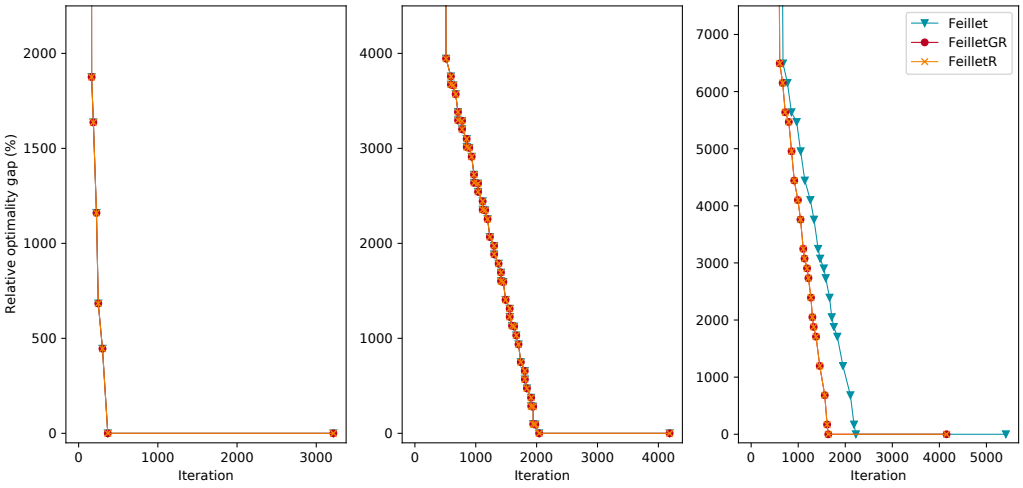


Figure 14.2: Comparison of computing times for Feillet's algorithm and its extensions for node reachability. While the left chart shows the overall computing time required for each benchmark instance, the right chart shows the proportion of preprocessing of the total computing time. Instances are sorted by the computing time of the plain implementation.



(a) Vehicles of instance C500D30TT-1 without lateness costs



(b) Vehicles of instance C500D30TTLL-1 with dominant lateness costs

Figure 14.3: Convergence of Feillet's algorithm to the shortest path for three randomly selected vehicles of instances (a) C500D30TT-1 and (b) C500D30TTLL-1. The charts show the optimality gap of all found feasible paths and the iteration when that path has been found while using or ignoring the node reachability extensions. From left column to right the vehicles required domestic delivery, inter-continental delivery and inter-continental delivery after modification in a VMC.

accurate here seems to improve the overall performance. This observation does not even change when doing considerable more than 500 path searches, where preprocessing might be negligible for FeilletGR.

Additionally, when looking at the convergence of the implementations to the optimal solution in Figure 14.3, we see that node reachability extensions entail minor decreases in the number of iterations required to find the shortest path and to terminate, at least for vehicles that require modifications. However, these vehicles account for only 10 % of the total vehicles in our benchmark instances, so this might not be the only reason for the superiority of R and GR. We worked out that the enhanced implementations still outperform the plain implementation if the path search is done only for such VMC vehicles (see Figure 14.1b). Nevertheless, the GR extension performs noticeable worse in favor of the plain implementation, while the performance of FeilletR seems to be unchanged. As a result, using FeilletR from the three presented alternatives is highly recommended for all kinds of vehicles. The question remains, why FeilletGR is not working well for VMC vehicles and if there are other characteristics that support the superiority of both extensions. Since the enhancements R and GR are outperformed by all other extensions (see below), we did not investigate this further.

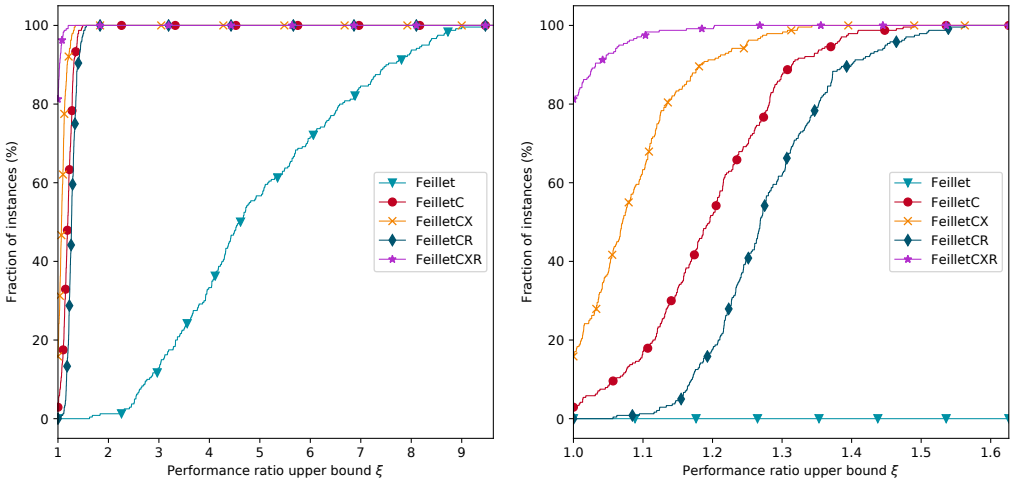


Figure 14.4: Performance profiles of Feillet's algorithm comparing the implementations that add path cost estimates. While (a) shows the full profiles, (b) restricts the view to the range $\xi \in [1, 1.65]$

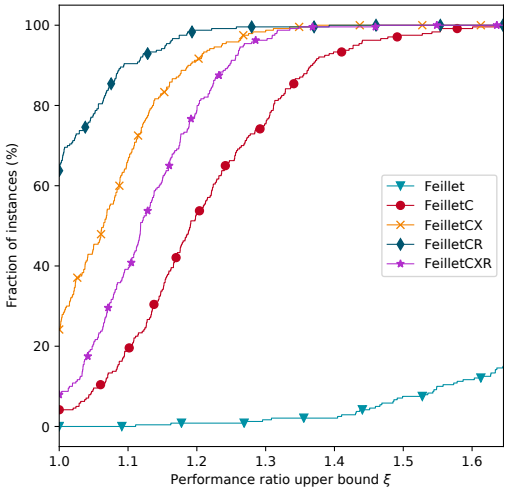
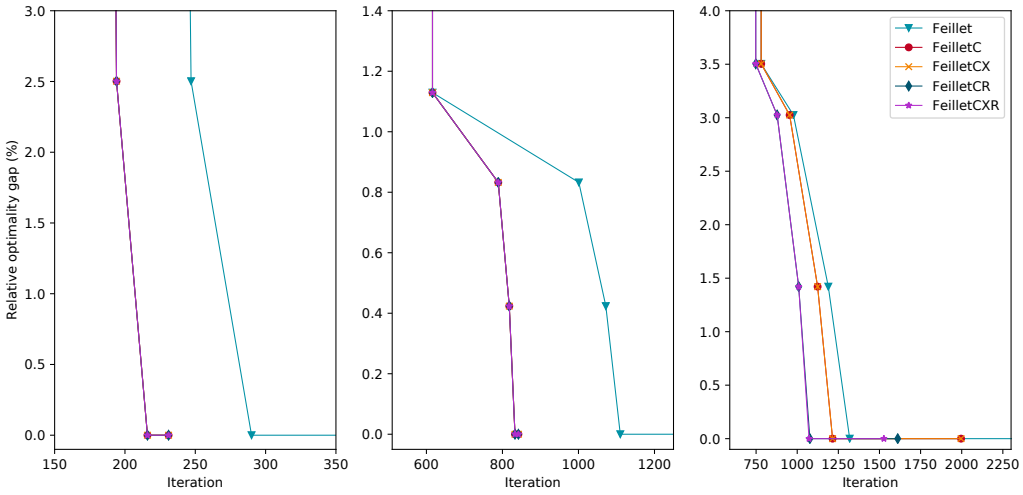
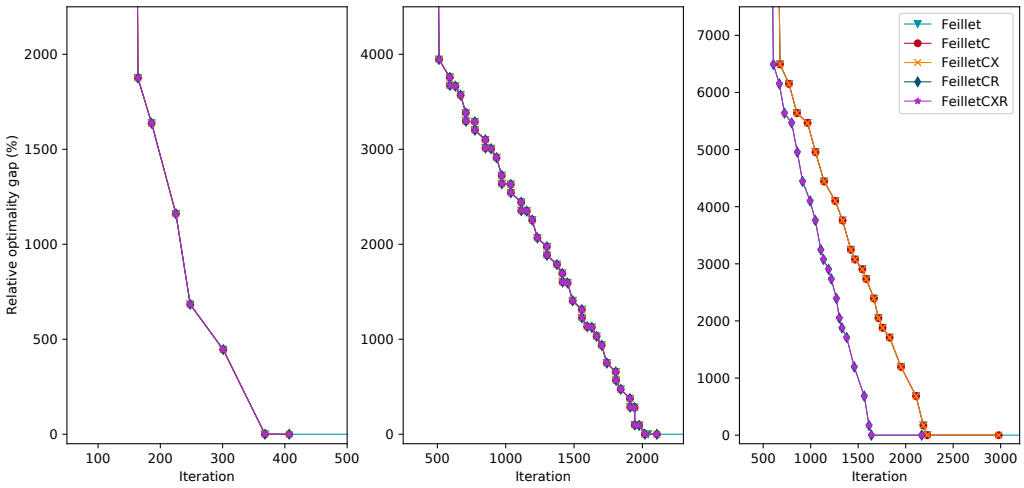


Figure 14.5: Performance profiles of Feillet's algorithm comparing the implementations that add path cost estimates. The benchmarks used only vehicles that require modifications. To improve readability, the view has been restricted to the range $\xi \in [1, 1.65]$.

We continued our study by enhancing Feillet's algorithm with path cost estimates. Figure 14.4 shows the corresponding performance profiles. FeilletCXR clearly outperforms all other implementations, with CX being the second best. In contrast, the plain implementation is not competitive at all. Obviously, enhanced cost estimates, that consider all permutations of required arcs, are worth the effort. Nevertheless, this changes as soon as we focus our study only on vehicles that require modifications. The performance profiles in Figure 14.5 indicate, that FeilletCXR is dominated by other extended implementations in this case. Both the CR and the CX extensions improve the path search for VMC vehicles. We were wondering why CX performs better, even though we cannot observe any advantage when looking at the convergence of the different implementations (see Figure 14.6). While there seems to be no difference for arbitrary vehicles, there is a considerable improvement for modification vehicles, at least regarding the iterations required to terminate. For extensions CR and CXR the convergence to the shortest path requires the fewest iterations. Investing additional CPU cycles to perform an all-pairs shortest path computation during preprocessing compared to a single-sink shortest path problem that is solved for FeilletC, seems to pay off in any case. However, we did not investigate if this changes with an increasing amount of nodes.



(a) Vehicles of instance C500D30TT-1 without lateness costs



(b) Vehicles of instance C500D30TTLL-1 with dominant lateness costs

Figure 14.6: Convergence of Feillet's algorithm to the shortest path for three randomly selected vehicles of instances (a) C500D30TT-1 and (b) C500D30TTLL-1. The charts show the optimality gap of all found feasible paths and the iteration when that path has been found while using or ignoring path cost estimates. From left column to right the vehicles required domestic delivery, inter-continental delivery and inter-continental delivery after modification in a VMC.

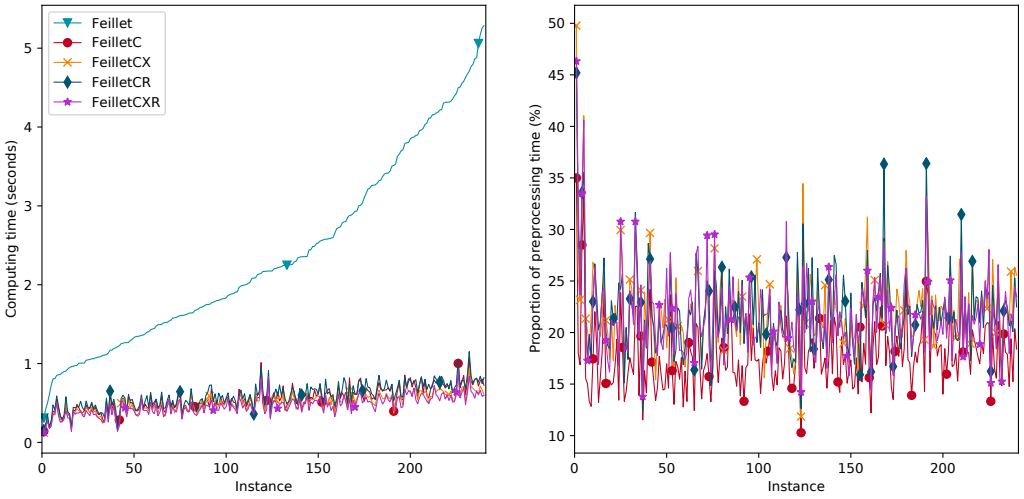
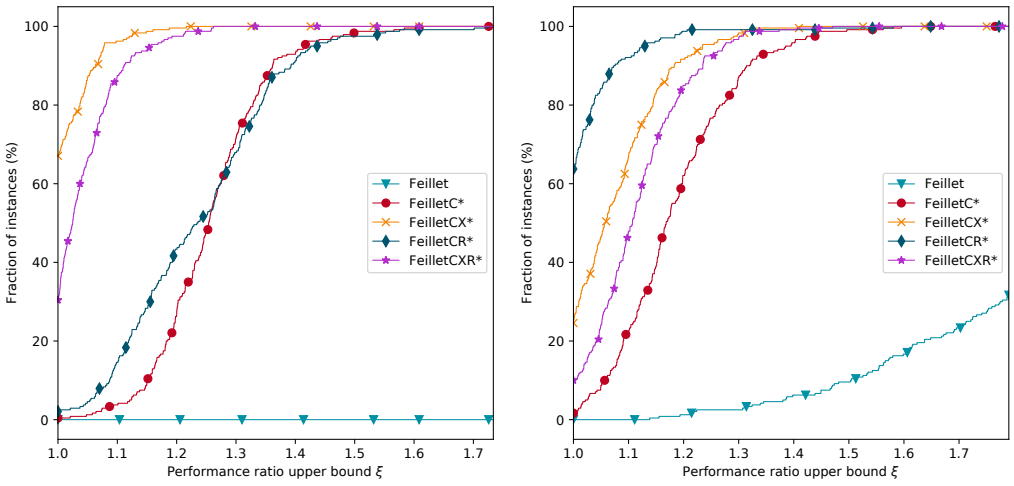


Figure 14.7: Comparison of computing times for Feillet's algorithm and its extensions adding path cost estimates. While the left chart shows the overall computing time required for each benchmark instance, the right chart shows the proportion of preprocessing of the total computing time. Instances are sorted by the computing time of the plain implementation.

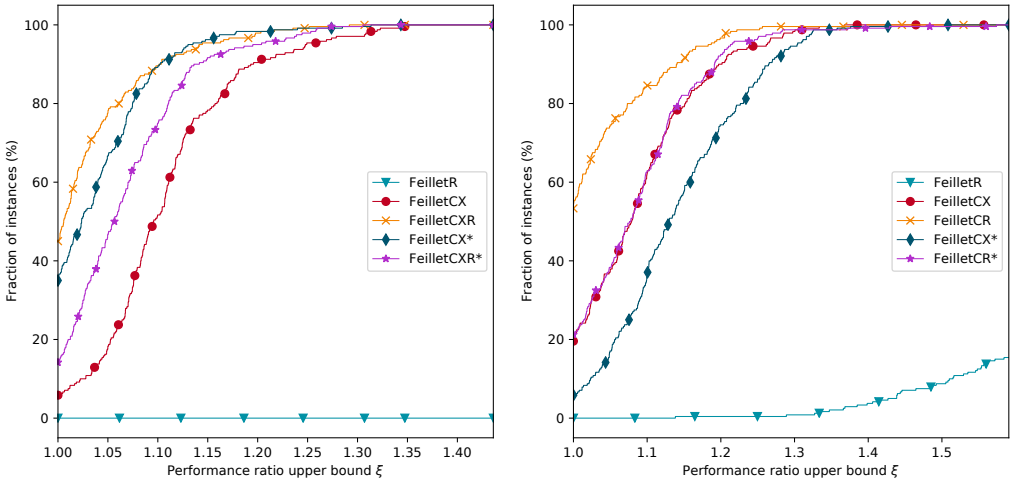
Figure 14.7 shows the computing times required for preprocessing, ranging from 10 % to 50 % of the overall computing times. While for FeilletC this proportion is 18 % in average, the other implementations required 22 %. The additional effort spent for determining the minimum cost estimate under all permutations seems to be beneficial, at least for our benchmark instances. We did not investigate further, if this advantage increases, when there are more arc-requirements or precedence relations. Nevertheless, we were wondering why FeilletCR outperforms both enhanced path cost estimation strategies for VMC vehicles. One possible explanation might be that the detour heading towards the VMC is the dominant factor of the path search and is well-supported by FeilletCR, that will discard all labels not leading there. Thus, the enhanced cost estimates might not be able to prune more labels in that instances, but consume additional CPU cycles. However, the measurements indicate that on our benchmark set FeilletCR should be used for all vehicles requiring modifications, while FeilletCXR is the best option for the other vehicles.

The relative performance of the cost-based enhancements does not change significantly when using the cost estimates to perform an A*-like prioritization of



(a) (b)

Figure 14.8: Performance profiles of Feillet's algorithm comparing the implementations with A*-like node sorting. While (a) shows the benchmark with arbitrary vehicles, in (b) only VMC vehicles have been used. To improve readability, the view has been restricted to the range $\xi \in [1, 1.8]$.



(a) (b)

Figure 14.9: Performance profiles of best implementations of Feillet's algorithm. While (a) shows the benchmark with arbitrary vehicles, in (b) only VMC vehicles have been used. To improve readability, the view has been restricted to the range $\xi \in [1, 1.6]$.

the pending nodes. Same holds for their convergence behavior. However, for non-modification vehicles FeilletCX* now outperforms FeilletCXR*. Figure 14.8 shows the performance profiles of the corresponding implementations. Again, the dominance of the CX* extension remains an open question. Nevertheless, maintaining the minimum path cost estimate from all labels at each node requires additional computational effort. Hence, it is no surprise that the performance decreases compared to the simple node ordering strategy. Figure 14.9 contrasts the best implementations of Feillet's algorithm. The previously stated suggestion still holds: FeilletCR is the best choice for VMC vehicles, for the others FeilletCXR should be used.

14.2 Adapted Boost Algorithm

The performance of the adapted Boost algorithm is quite similar to Feillet's algorithm when adding the node reachability extensions, at least for VMC vehicles. In contrast, the algorithms perform noticeable different for the others. Here the plain implementation clearly outperforms both extensions. This fits to its

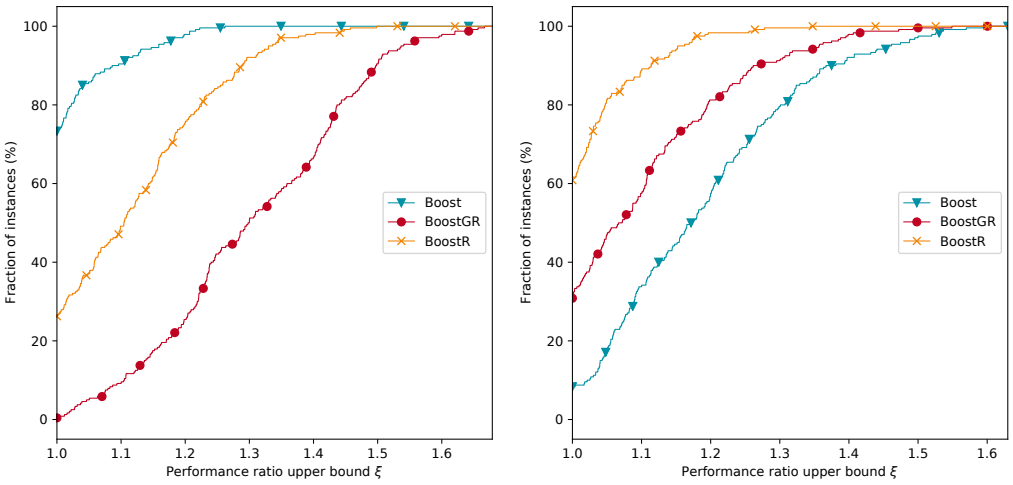
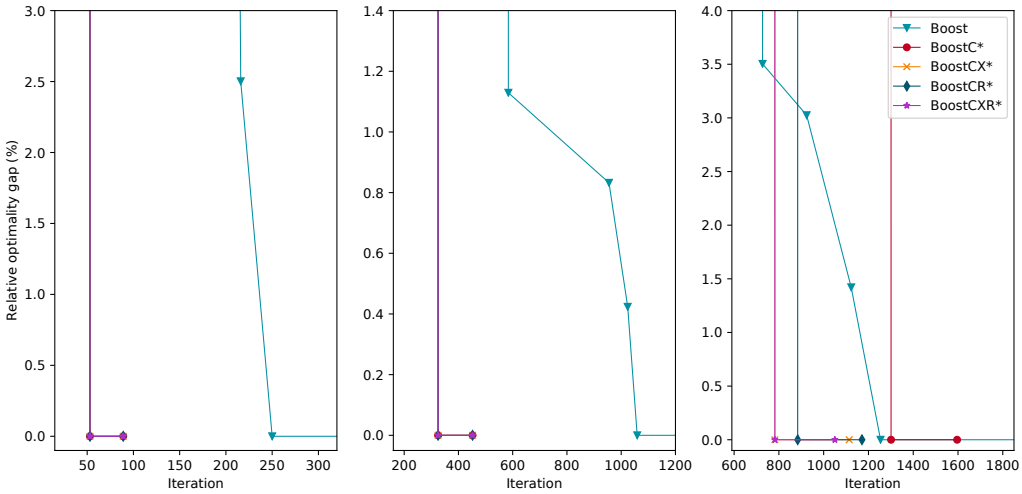
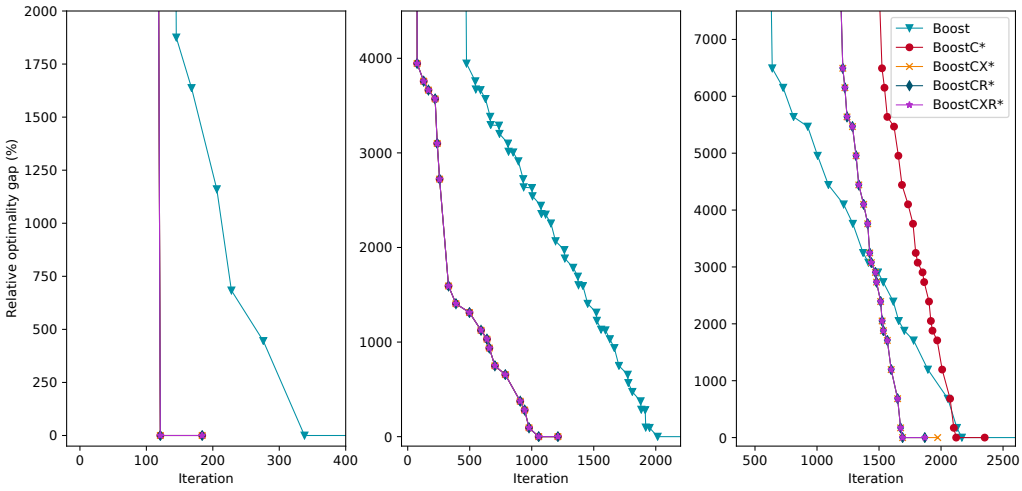


Figure 14.10: Performance profiles of Boost algorithm comparing the implementations that add node reachability calculations. While (a) shows the benchmark with arbitrary vehicles, in (b) only VMC vehicles have been used.

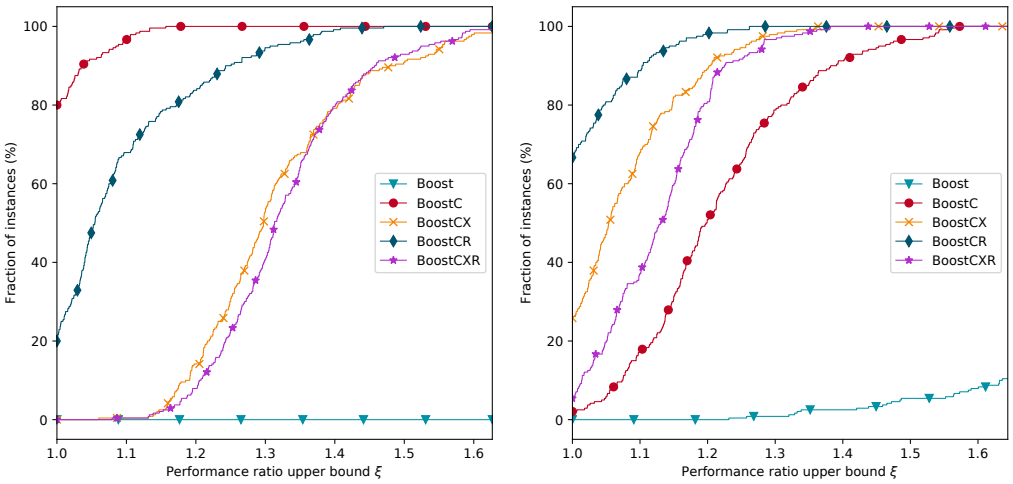


(a) Vehicles of instance C500D30TT-1 without lateness costs



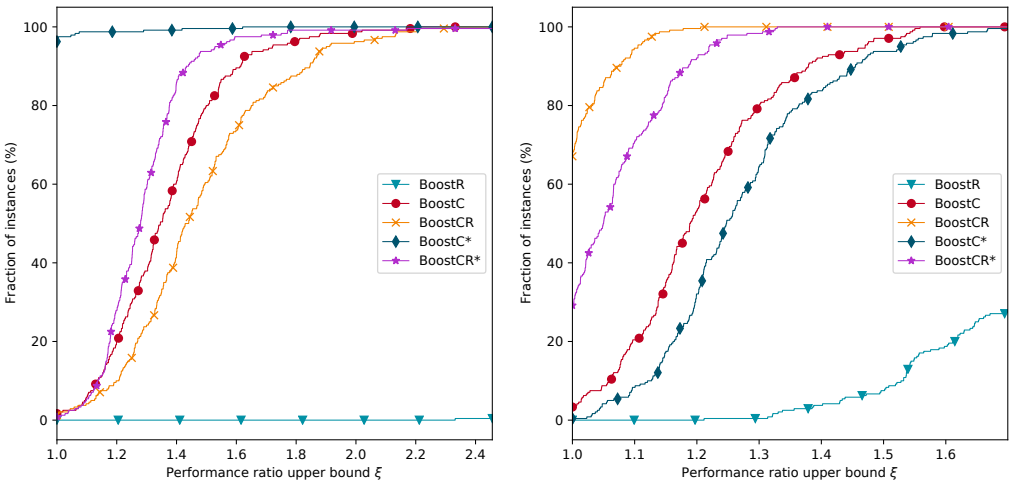
(b) Vehicles of instance C500D30TTLL-1 with dominant lateness costs

Figure 14.11: Convergence of Boost algorithm to the shortest path for three randomly selected vehicles of instances (a) C500D30TT-1 and (b) C500D30TTLL-1. The charts show the optimality gap of all found feasible paths and the iteration when that path has been found while using or ignoring path cost estimates and label sorting. From left column to right the vehicles required domestic delivery, inter-continental delivery and inter-continental delivery after modification in a VMC.



(a) (b)

Figure 14.12: Performance profiles of Boost algorithm comparing the implementations that add path cost estimates. While (a) shows the benchmark with arbitrary vehicles, in (b) only VMC vehicles have been used. To improve readability, the view has been restricted to the range $\xi \in [1, 1.65]$.



(a) (b)

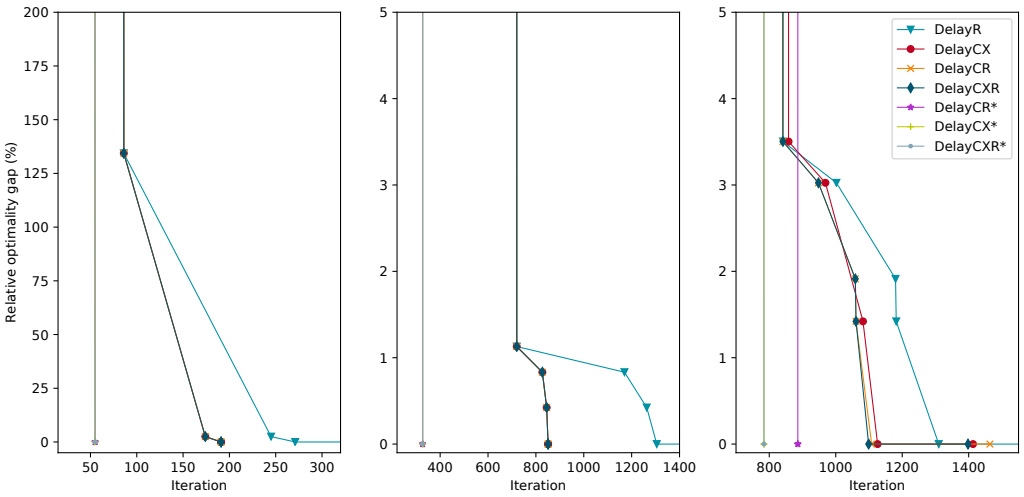
Figure 14.13: Performance profiles of best implementations of Boost algorithm. While (a) shows the benchmark with arbitrary vehicles, in (b) only VMC vehicles have been used. To improve readability, the view has been restricted to the ranges (a) $\xi \in [1, 2.7]$ and (b) $\xi \in [1, 1.7]$.

convergence behavior that looks similar to the one reported for Feillet's algorithm. There is no improvement regarding the amount of iterations, until we focus the view on vehicles that require modifications. Because the node reachability extensions are again outperformed by all other implementations, we keep the analysis short at this point and only show the corresponding performance profiles in Figure 14.10.

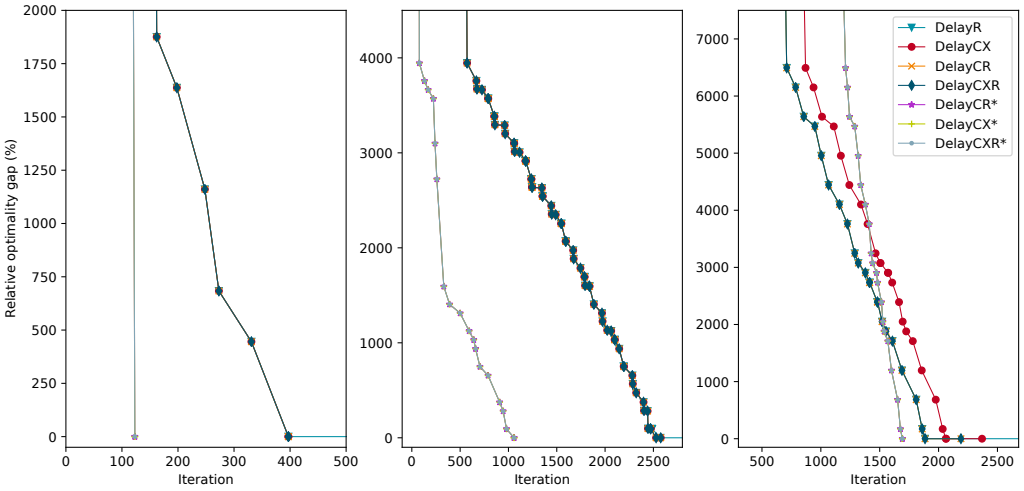
Furthermore, the relative performance looks identical for VMC vehicles when adding path cost estimates (see Figure 14.12), while it is considerable different for arbitrary vehicles. For the latter, BoostC shows noticeable better performance than all other extensions, while CR again is the best choice for VMC vehicles. The same results can be observed, when sorting the pending labels in an A*-like fashion and when looking at the convergence behaviour in Figure 14.11. It has been surprising that in most cases the first found valid path already is the shortest one. After having found the optimal solution, all enhanced implementations require only a few iterations to terminate. Again, extensions CR* and CXR* converge and terminate with the fewest iterations. Propagating the best available label in each iteration seems to quickly lead to good valid solutions and thus supports the pruning by upper bounds. Nevertheless, the additional effort for permutation-based cost estimates does not lead to any improvement. Figure 14.13 shows the final comparison of the best implementations of the Boost algorithm. For arbitrary vehicles the implementation C* performs noticeable better than all the other implementations. However, when looking only at VMC vehicles, again the CR extension outperforms the others.

14.3 Delayed Dominance Algorithm

When examining the Delayed Dominance algorithm, we were surprised that its performance and convergence is similar to Feillet's algorithm. We had assumed to see a behavior that is closer to the one of Boost, as this algorithm performs nearly the same steps (although in a different order). However, our expectation was that delaying the dominance checks and propagating the most promising labels first by using an A*-like sorting strategy can significantly increase performance, because many dominance checks are not required anymore. The algorithm might find a good feasible solution quickly and afterwards it is able to prune further labels excessively using the tight upper bound. We selected the same implementations as the best ones, that were used for Feillet's algorithm before. Figure 14.15 shows that

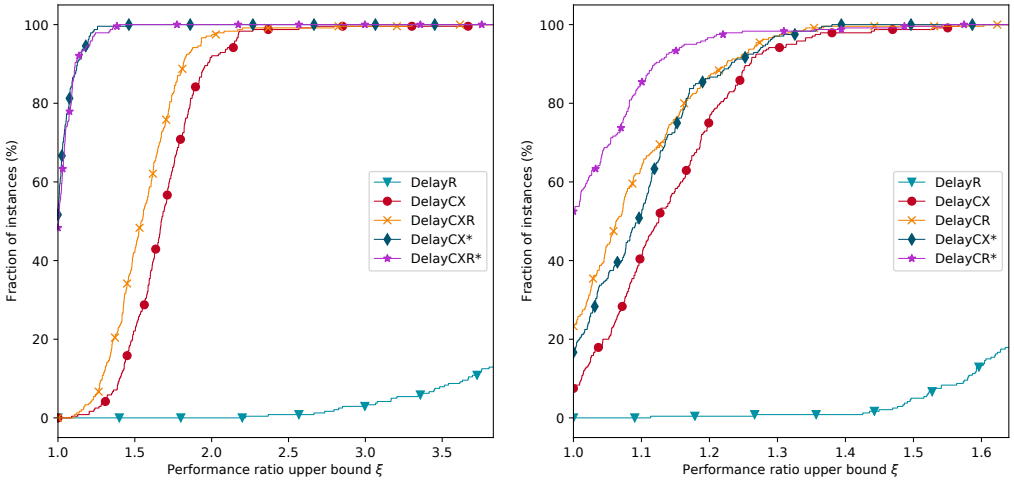


(a) Vehicles of instance C500D30TT-1 without lateness costs



(b) Vehicles of instance C500D30TTLL-1 with dominant lateness costs

Figure 14.14: Convergence of Delayed Dominance algorithm to the shortest path for three randomly selected vehicles of instances (a) C500D30TT-1 and (b) C500D30TTLL-1. The charts show the optimality gap of all found feasible paths and the iteration when that path has been found for the best performing implementations. From left column to right the vehicles required domestic delivery, inter-continental delivery and inter-continental delivery after modification in a VMC.



(a) (b)

Figure 14.15: Performance profiles of best implementations of Delayed Dominance algorithm. While (a) shows the benchmark with arbitrary vehicles, in (b) only VMC vehicles have been used. To improve readability, the view has been restricted to the ranges (a) $\xi \in [1, 3.85]$ and (b) $\xi \in [1, 1.65]$.

implementations CX* and CXR* perform best for arbitrary vehicles. Nevertheless, in case of VMC vehicles the CR* extension outperforms all the others, even the CR extension which has been the best for the other two algorithms. Looking at the convergence behavior of these implementations in Figure 14.14, that dominance of A*-like sorting becomes obvious and supports our assumption that these strategies help in finding the optimal solution quickly. Both DelayCX* and DelayCXR* terminate as soon as the first valid path has been found. DelayCR* converges and terminates fast in all presented examples, although it is not as quick as the other two implementations in every instance. Nevertheless, the best choice for vehicles requiring modifications is CR*, while for arbitrary vehicles we need to decide between CX* and CXR*, which show nearly identical performance. We are going to continue with both implementations and evaluate how they compare to the best of the other algorithms.

14.4 Selection of Best Path Search Algorithm

Now that we have selected the best performing implementations of each algorithm, we continue determining which algorithm should be used for the remainder of our computational study. For arbitrary vehicles, the choice of an algorithm is simple. Figure 14.16 shows that DelayCX* and DelayCXR* clearly outperforms the other implementations, both having nearly equal performance. We decided to choose the DelayCX*, as this is the less complex approach. However, for VMC vehicles the decision is not that trivial. FeilletCR, BoostCR and DelayCR* show similar performance with a minor preference on FeilletCR. However, since an advantage of any of these variants can hardly be measured, we decided for the application of DelayCR*. That choice entails the benefit of using the same algorithm in all situations (although their configurations are slightly different) which felt like a practical approach that avoids the implementation and maintenance of two different algorithms. Thus, in the remainder of this computational study, we will use DelayCX* and DelayCR*, depending on the modification requirements of a vehicle.

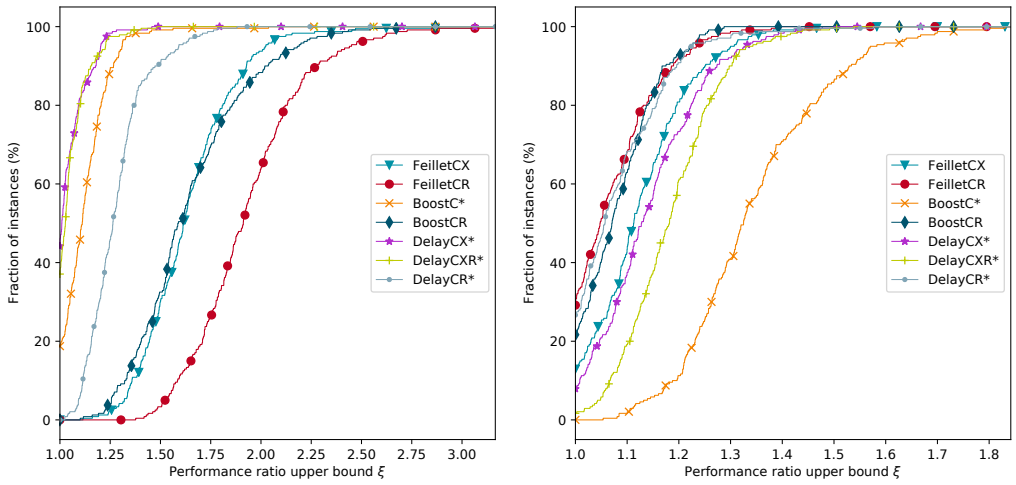


Figure 14.16: Performance profiles of best implementations of all algorithms. While (a) shows the benchmark with arbitrary vehicles, in (b) only VMC vehicles have been used.

14.5 Impact of Negative Cost Cycles

All benchmark instances considered so far only arcs with non-negative costs and acyclic graphs. In the following, we will investigate the impact of adding negative-cost cycles to our benchmark instances on the overall algorithmic performance, although we expect such cycles to rarely occur in real-life instances. For that, the graph $\mathcal{D} = (V, A)$ of each instance of class C10D10TT has been modified. For each arc $a_1 = (v, u) \in A$ with $\tau(a_1) = 0$ an anti-parallel arc $a_2 = (u, v)$ with $\tau(a_2) = 0$ is introduced. We add a_2 to each set A^k with $k \in K$ that already contained a_1. Furthermore, the cost function is extended to $c(a_2, k) = c(a_1, k) + 1$ for all $k \in K$, which allows the existence of negative-cost 2-cycles. Afterwards, we randomly select a proportion of arcs from the extended graph and negate its cost function values for all commodities. That proportion is varied during the benchmark runs, which again calculate the resource-constraint shortest path for 500 randomly selected commodities. The upper bound for the artificial cycle detection resource r_L has been fixed to $M = 50$ for all measurements, since we assumed that it is unlikely that the shortest path contains more arcs.

We expect the performance of the algorithms to be noticeably worse compared to the benchmark results presented before. There are several effects that will have a negative impact on the amount of created labels and required iterations to terminate. First, cost estimates are likely not available any longer, as all the used single-sink and all-pairs shortest path algorithms are incapable of handling negative-cost cycles. The existence of negative-cost arcs within the graph (even if there are no cycles) prohibits the pruning by upper bounds, if the cost estimates are not calculated exactly. Furthermore, the usage of the artificial resource r_L that is required for cycle detection (see Section 6.3), will influence the dominance relation of the labels. There will be more labels that are not dominated. Finally, all algorithms will require several augmentation iterations until the state-space contains all nodes required to find an elementary shortest path. We expect this to considerably increase the overall computing time.

When starting our experiments, we quickly worked out that path cost estimation during preprocessing is not useful any longer, regardless of the usage of all-pairs or single-sink shortest path algorithms. Obviously, the support graph that is utilized for the calculation of the estimates contains a noticeable amount of cycles, which is no surprise, since we eliminated the time-dimension from that graph. So

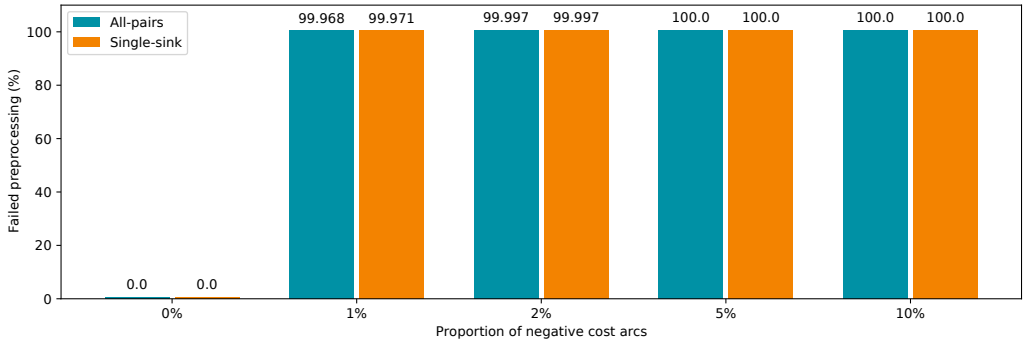
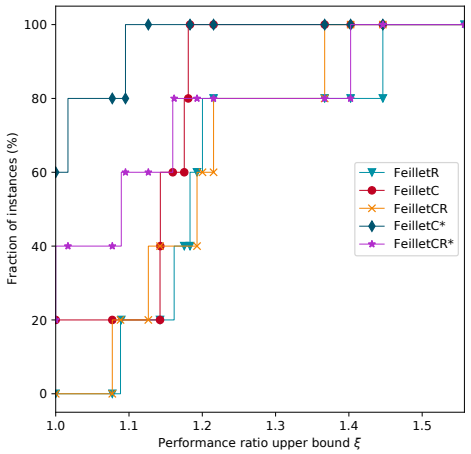


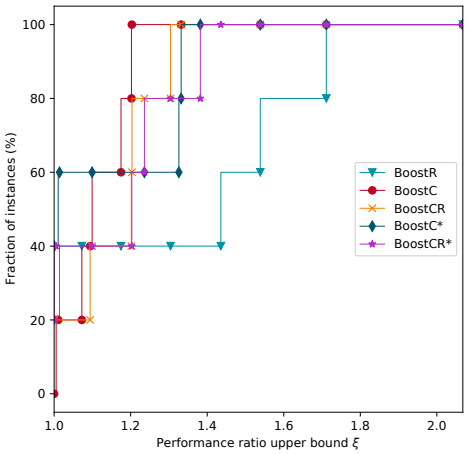
Figure 14.17: Chart showing the proportion of failed path cost estimates when negative-cost cycles exist. When the proportion of negative-cost arcs increases, the amount of failed estimates increases as well, regardless of the usage of all-pairs or single-sink shortest path algorithms.

even if the time-expanded graph is acyclic, the corresponding support graph may not be. As a result, randomly choosing arcs and negating their costs will easily lead to negative-cost cycles within the support graph. Figure 14.17 shows the proportion of failed path cost estimate calculations when such a negative cost cycle has been detected. The calculation fails for nearly all benchmark runs. Hence, it is no surprise, that the performance profiles of the path search algorithms in Figure 14.18 change significantly. Please note that we did not explicitly measure the implementations with permutation-aware cost estimates, since the all-pairs shortest path preprocessing fails in nearly all cases and as a result the enhanced cost estimates of these implementations will be switched off. In that case CX will execute the estimation procedures used by C and the same holds for the pairs CXR / CR, CX* / C* and CXR* / CR*. Thus, the measurements will be identical and can be ignored in the following.

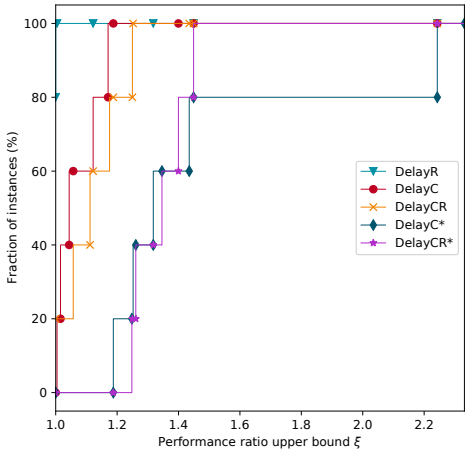
The implementations FeilletC* and FeilletCR* show the best performance over all benchmark instances after negating the costs of 5 % of all arcs. Feillet's algorithm seems to be a good choice for instances containing negative-cost cycles, also the Delayed Dominance algorithm performs well when only conducting node reachability calculations. Looking at the impact of the proportion of negative-cost arcs on the distribution of computing times in Figure 14.19 shows, that there seems to be a plateau for each proportion that is independent of the used algorithm. We assume these plateaus to be related to the amount of required state-space augmentation iterations (see Figure 14.20). Furthermore, the variance of the measurements



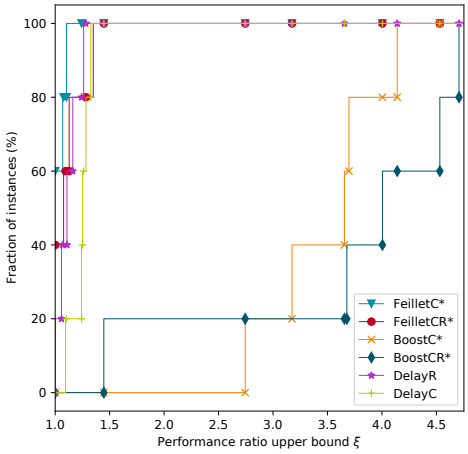
(a) Feillet's algorithm



(b) Boost algorithm

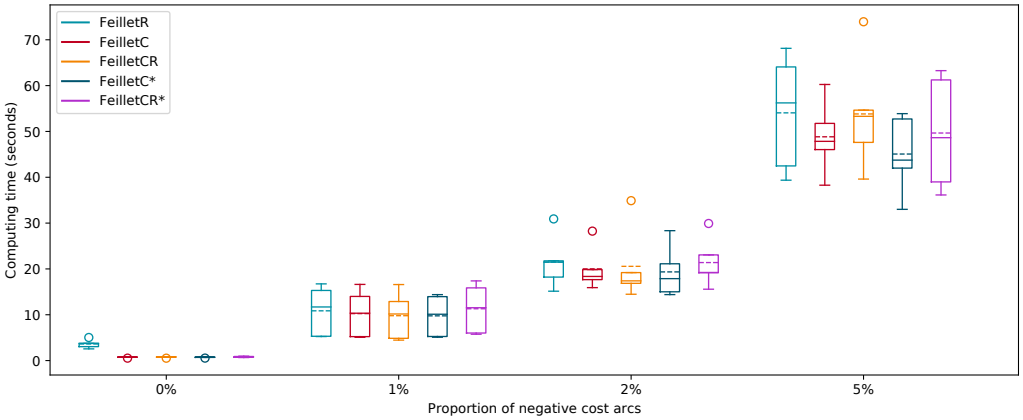


(c) Delayed Dominance algorithm

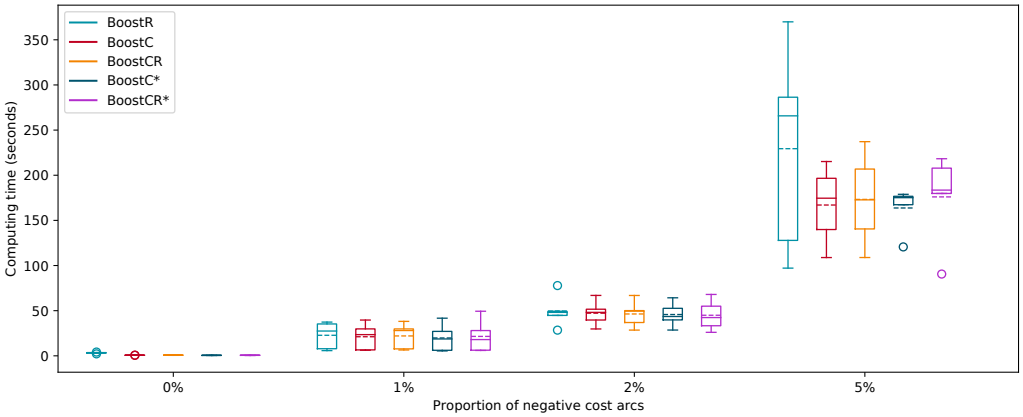


(d) Best implementations

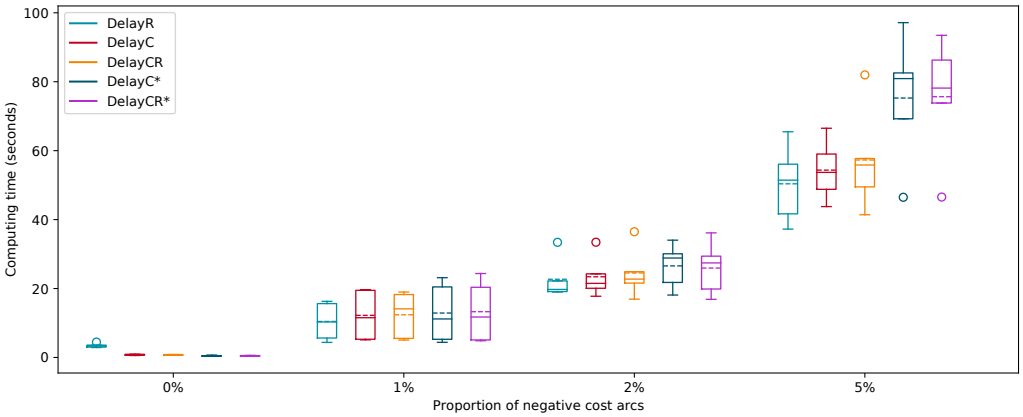
Figure 14.18: Performance profiles of all algorithms when adding negative-cost cycles to the benchmark instances. The benchmarks used instances with 5 % of the arcs having negative costs.



(a) Feillet's algorithm



(b) Boost algorithm



(c) Delayed Dominance algorithm

Figure 14.19: Box plots showing the impact of negative-cost arcs on the computing times of all examined algorithms.

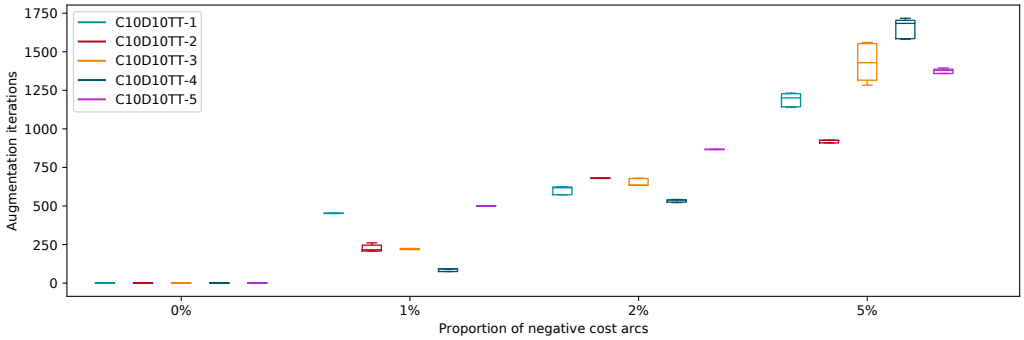


Figure 14.20: Box plots showing the amount of state-space augmentation iterations for each benchmark instance. The boxes are aligned to the 25th and 75th percentile. The whiskers represent the minimum and maximum values.

increases with the amount of negative-cost arcs. Since not all implementations find the same shortest path (the shortest path is not unique) they require different amounts of augmentation iterations. We expect these paths to vary more widely when the amount of negative-cost cycles increases. That entails the larger variance of the measurements. Above results indicate, that using DelayCX* and DelayCR* are not the best choices, if the network contains negative-cost cycles. Hence, for such cases, which can only occur due to negative reduced costs within a column generation approach, we will use FeilletC* in the remainder of this work.

14.6 A* Algorithm

We complete our computational study on path search algorithms by evaluating the performance of the A* algorithm. As mentioned before, this algorithm can only be used for vehicles that have neither arc-requirements nor precedence relations and if all arc costs are non-negative. Hence, in the following a “normal” shortest path problem without any resource-constraints must be solved. Again, the shortest path of 500 randomly selected vehicles from each benchmark instance will be calculated. If one of the chosen vehicles has arc-requirements or precedence relations, these will be ignored during path search.

Figure 14.21 shows the comparison of Dijkstra’s algorithm (which is equal to A* without path cost estimates) and our A* implementation, both enhanced with node

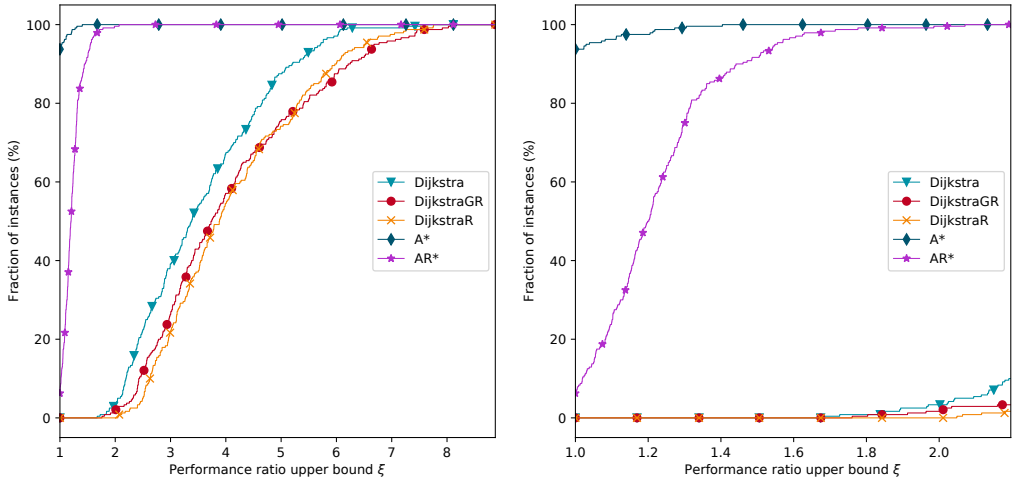


Figure 14.21: Performance profiles of A* algorithm comparing all different implementations. While (a) shows the full profile, (b) restricts the view to the range $\xi \in [1, 2.2]$

reachability calculations. All implementations used an array-based binary heap as queue. The results are unambiguous: A* (using path cost estimates, but no node reachability) outperforms all other implementations. In 90 % of all benchmark instances this implementation solved the problem fastest and is able to solve all instances within a performance ratio of 1.4. These results match our expectation of A* being dominant and there is no doubt that we should use this implementation during the remainder of our computational study.

15 | Branch-and-Cut

When evaluating the performance of a branch-and-cut approach, there are many adjustments that might need to be considered. Starting with the method for solving LP-relaxations, over the chosen branching and node selection strategies, to the aggressiveness of pre-solving and heuristics. Although, all of these offer a high potential for improving the overall solution process (Achterberg and Wunderling 2013), in the remainder of this section, we will focus entirely on the performance and impact of the separation of cuts. We will show that our Branch-and-Cut approach is unable to solve medium-sized benchmark instances and we did not see any chance that one of the other adjustments might be able to improve that situation significantly. Hence, we did not investigate these adjustments further. Please note that we are not going to evaluate the separation of the lazy constraints explained in Section 7.2, since our benchmark instances do not contain any cycles. We assume that cyclic graphs are an edge case in our problem domain and hence leave that task open for further research.

For our implementation, we utilized Gurobi and its Java API to solve MIP (5.10) presented in Chapter 5. We focused our evaluation on the performance of cuts and left all other parameters on Gurobi's default settings. For the aggregation of commodities, we chose the multiple-source-single-sink approach, which we supposed to perform best, since the introduction of the super-sink entailed that there are considerable fewer sinks than sources. Although, recent research showed that single-source-single-sink aggregations may perform better, especially for instances with many commodities (Chouman, Crainic, and Gendron 2016), in preliminary testing, we faced noticeable more unsolved instances because of out-of-memory issues. Please note that even for multiple-source-single-sink aggregation 10 instances could not be solved in our experiments. We figured out that instances containing lateness costs are especially vulnerable to such issues, which is no surprise taking into account that these instances require more data in computer memory and the existence of lateness costs avoids aggregation of commodities to some degree.

MIP solvers typically use heuristics to find good feasible solutions and generate cuts mainly to close the gap between best known solution value and lower bound. As we wanted to focus on cut generation, we decided to perform all experiments using warm-start solutions that have been created by our instance generator. These warm-start solutions showed an average MIP gap of 9.6 % after solving the root node LP-relaxation (see Figure 7.1 or Table 21.1). For the same reason, Gurobi has been configured to separate all cuts most aggressively (e.g., by setting parameter `CliqueCuts = 2`).

While the MIP that needs to be solved has been described in detail in Chapter 5, there is still one question unanswered: How to choose the constants M^k? These constants are used as costs for the shortcut arcs from origin to destination nodes. Obviously, these constants need to be chosen sufficiently large, to ensure that a shortcut arc is only used in case of an infeasible instance. In contrast, we cannot assign them arbitrarily large values, or we might run into rounding errors due to floating-point precision of the used computer hardware. Fortunately, the matrix of our MIP does not contain any of these constants, thus they will not lead to any instability issues. For our benchmark instances, we assumed that no feasible solution will contain a path that consists of more than 30 arcs. We multiplied this value with the largest arc costs available within our network to obtain a single constant M that will be used for all commodities.

$$M^k = M = 30 \max_{k \in K, a \in A^k} \left\{ c_a^k + \tilde{c}_a \right\} \tag{15.1}$$

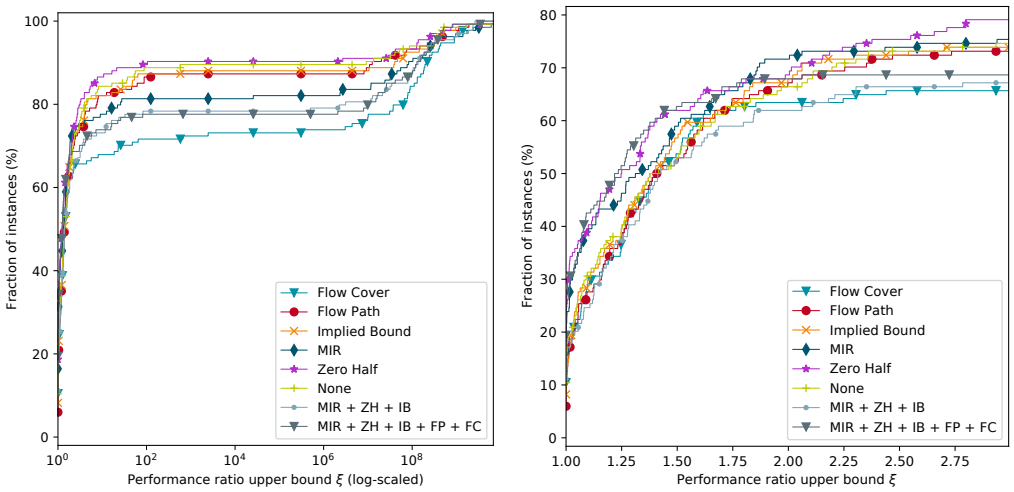
The experiments have been run on all small and medium size instances (classes C1D5 to C50D10), while we used the largest ones only for the final benchmark of the best solver configuration and the others only during parameter tuning. The available hardware resources have been varied depending on the size of the MIPs that have been created from the instance. Table 21.1 in the appendix lists all details regarding used cores and memory for each instance. Since all experiments have been performed using JMH and Gurobi's Java API, the available memory needed to be split into memory available to the JVM as heap and native memory used by Gurobi directly. In preliminary testing, we figured out that a distribution of 75 % for the JVM to 25 % for Gurobi works well. We configured JMH to fork a single JVM for each benchmark instance and run a single measurement iteration. As all instances required a noticeable amount of computing time, we assumed that

further measurement or warm-up iterations are not necessary to gather usable benchmark results. All reported computing times include only the time spend by Gurobi to solve the MIPs. Preprocessing steps or the creation of the MIPs have been excluded explicitly. Please note that we used performance profiles with and without log-scaled axes, which must be considered when comparing different profiles. Furthermore, we varied how performance is measured depending on the use-case (computing time, MIP gap, explored nodes or simplex iterations).

15.1 Built-in Cuts

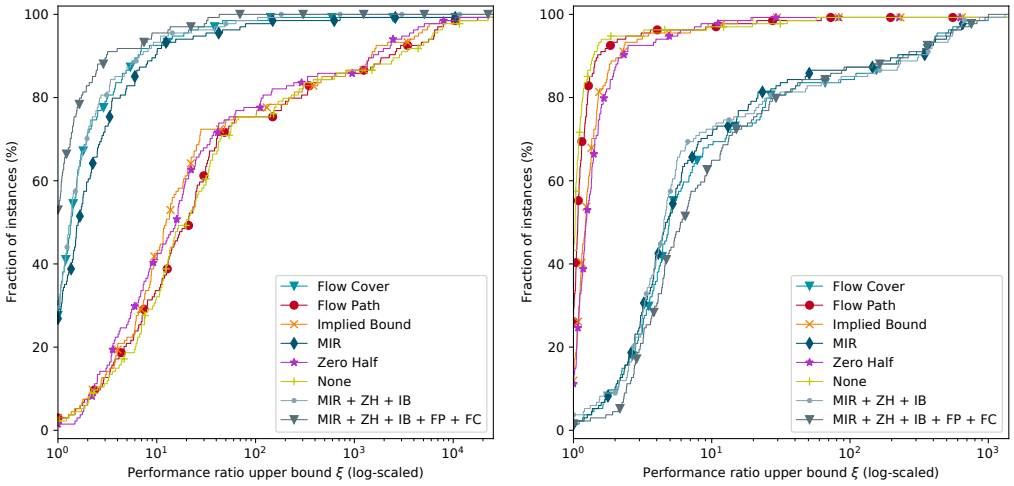
We started our computational study for the branch-and-cut solver by evaluating the impact of Gurobi's built-in cut separation procedures. The Gurobi version used in our experiments (v9.0.0) offered algorithms for 18 different cuts. First, we solved instances of classes C1D5 to C10D10 using each of these cuts in isolation and comparing the results to the case where no cuts are separated. Additionally, we combined the most promising cuts and solved the instances again, while generating all of these cuts simultaneously. Figure 15.1 shows the profiles of the best performing cuts and cut combinations. Since cut-generation did not improve the solution process for "easy" instances at all, we focus the following figures only on those instances that could not be solved to optimality within a time limit of 2 hours in any experiment (which we call "hard" instances). Performance is measured by the remaining MIP gap.

Only MIR and Zero Half cuts show an apparent improvement compared to the case where no cuts are generated. Furthermore, the combination of MIR, Zero Half, Implied Bound, Flow Path and Flow Cover cuts performs quite well. All other cuts perform equal to or worse than the no-cuts case, especially those cuts that have not been listed here, but are available within Gurobi. We were wondering what causes these unfavorable performance results for most of the cuts. Typically, the generation of cuts will improve the bounds of the node's LP-relaxation and thereby support cutting off branches from the branch-and-bound tree. As a consequence, less nodes will be explored until an optimal solution is found. In contrast, the size of the LP solved for each node will increase when cuts are added, which in turn will raise the amount of iterations required to solve one of these LP-relaxations. So separation of cuts should target a balance between decreased amount of nodes



(a) (b)

Figure 15.1: Performance profiles of best performing built-in cuts and cut combinations of Gurobi. While (a) shows the full profile, (b) restricts the view to the range $\xi \in [1, 3]$. Please note that the horizontal axes are scaled differently, to simplify visualization.



(a) (b)

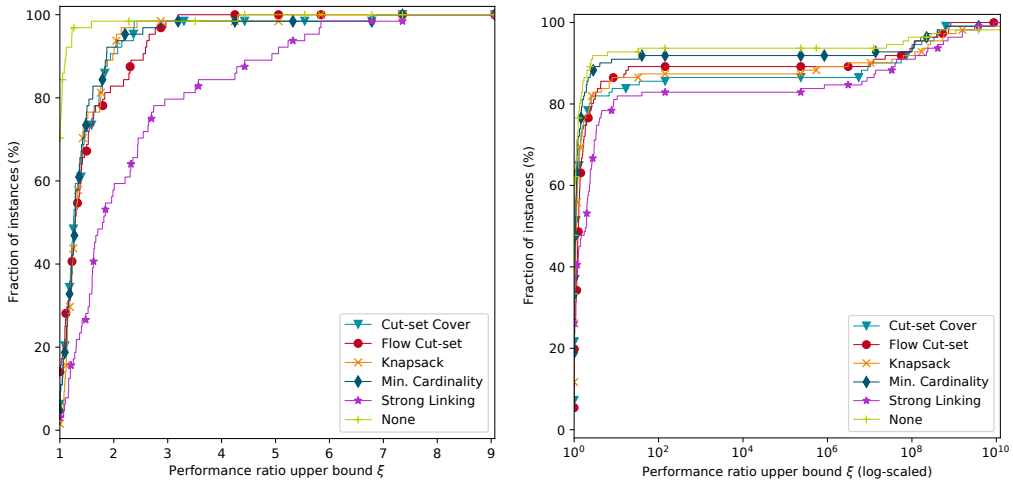
Figure 15.2: Performance profiles of Gurobi's cut separation algorithms derived from (a) the amount of explored nodes and (b) the average simplex iterations per node.

in the branch-and-bound tree and the time required to solve the LP-relaxation of each of these nodes to optimality.

Figure 15.2a shows a performance profile that is based on the amount of explored nodes. In comparison Figure 15.2b shows profiles where performance is measured by the average simplex iterations per node. Obviously, there are a few cuts that are overly successful in reducing the amount of explored nodes, but significantly increase the required simplex iterations. MIR, Zero Half and the cut combination mentioned above seem to have the best balance between these two influencing factors.

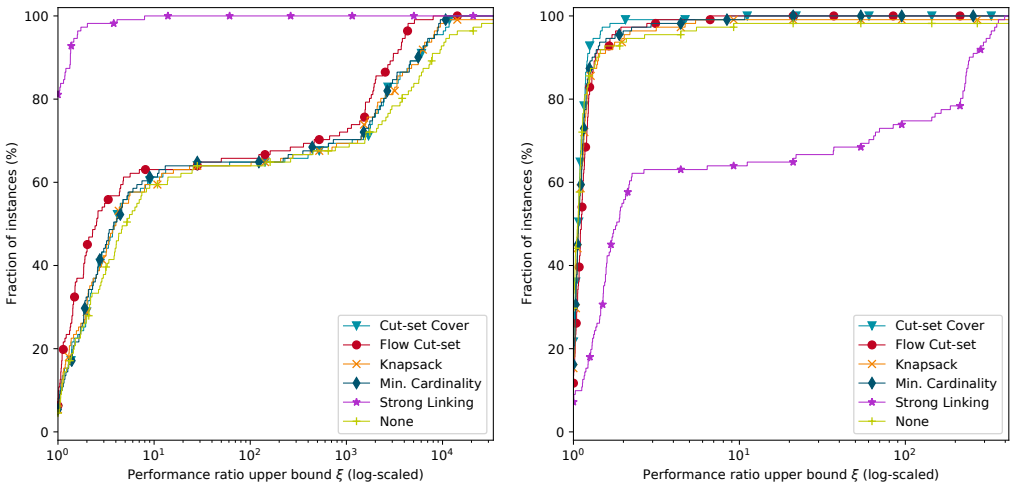
15.2 Custom Cuts

We continued our computational study by evaluating the custom cuts that have been explained in Section 7.4. Again, we solved instances of classes C1D5 to C10D10 within a time limit of 2 hours, while generating each of the cuts in isolation. Figure 15.3 shows the performance of the cut-generation on our easy and hard instances. While the performance is measured by the remaining MIP



(a) (b)

Figure 15.3: Performance profiles for generation of custom cuts on (a) easy and (b) hard instances. Please note that the horizontal axes are scaled differently and that performance is measured in solving time for easy instances and is based on the remaining MIP gap for the hard ones.



(a) (b)

Figure 15.4: Performance profiles of custom cut separation algorithms derived from (a) the amount of explored nodes and (b) the average simplex iterations per node (comparing only hard instances).

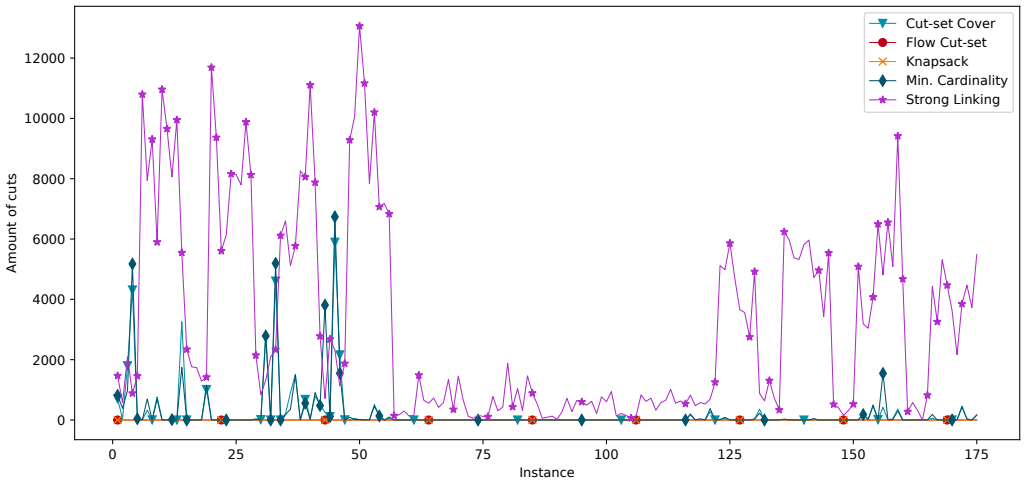


Figure 15.5: Chart showing the amount of generated custom cuts for each benchmark instance.

gap for the latter, we used the overall solving time for the easy instances (which have been all solved to optimality within the time limit).

Obviously, none of the cuts is able to improve the overall solution process. For easy instances the no-cuts case clearly outperforms all cut-generating approaches and while it shows the best performance for the hard instances as well, it is at least not that dominant. Looking at the amount of explored nodes in Figure 15.4a and the required simplex iterations in Figure 15.4b, only strong linking cuts seem to have a considerable impact on the overall solution process. However, they do not improve it.

This result can easily be explained when investigating the amount of cuts that have been generated. For most instances, only strong linking cuts are generated, except for a few instances with cut-set cover and minimum cardinality cuts (see Figure 15.5). All other types of cuts do not seem to be violated or discovered by the implemented algorithms. Hence, it is no surprise, that neither the amount of explored nodes nor the required simplex iterations are affected. However, all separation algorithms require some computing time, which explains that the cut-set-based inequalities perform worse compared to the no-cuts case. Although strong linking cuts were identified as crucial factor for solving network design problems by other researchers (see Katayama, Chen, and Kubo 2009; Andersen et al. 2010; Hewitt, Nemhauser, and Savelsbergh 2012; Gendron and Larose 2014), our experiments did not confirm this. Besides the reported results, we tried different algorithmic approaches to reduce the amount of separated strong linking cuts, targeting a better balance between decrease of explored nodes and increase of required simplex iterations. However, this was not successful. Furthermore, we could not determine with certainty whether the unfavorable performance of the cut-set-based cuts is related to our implementation or is caused by the structure of our benchmark instances.

15.3 Best Configuration

After having evaluated Gurobi's built-in and our custom cuts, we configured Gurobi to use the best cut generation algorithms and tried to solve all instances of larger class C50D10. Since none of the custom cuts showed favorable performance, we decided to no longer use them. Looking at the remaining MIP gap when using Gurobi's cuts, there is only a marginal difference between the best three options. Thus, we decided to focus our computational study on Zero Half cuts only, which nearly did not increase the amount of simplex iterations but managed to cut off a decent amount of nodes. We assumed that this selection will be beneficial for even larger instances, when solving the LP-relaxation is already computationally expensive.

Table 15.1 shows some details regarding the created MIPs and the remaining MIP gap when solving all instances of class C50D10 using Zero Half cuts only. Solving time has been limited to 10 hours and for each experiment we used 8 cores and 96 GB of main memory. These experiments demonstrate the major weakness of our branch-and-cut approach: it does not scale very well. Although we significantly increased the amount of available main memory, most of the benchmarks failed due to memory issues. These 40 instances are not listed in the table. Analyzing our log files in detail showed, that for most instances our algorithm was unable to create a MIP because it required too much memory. Some instances failed during Gurobi's pre-solving step, which seems to require a significant amount of memory as well. Furthermore, even when Gurobi started to solve one of the MIPs, the quality of the best solution after 10 hours is not always satisfactory.

From the given results of our experiments, we see no option how this branch-and-cut approach can be improved to solve even larger real-world instances. Although these results are not surprising and have been anticipated to some degree, we would have expected to run into such issues only when using larger instances from class C100D30 or C500D30.

Table 15.1: Table showing the size of the MIPs generated for each benchmark instance, the amount of explored nodes and simplex iterations and the quality of best found solutions measured by their MIP gap. All instances that failed during creation of the MIP are not listed.

Instance	**Rows**	**Columns**	**Explored nodes**	**Iterations per node**	**Gap (%)**
C50D10R-1	13 658 376	17 335 326	5778	3371	1.7
C50D10R-2	30 738 108	38 770 347	*Out of memory in pre-solving*		
C50D10R-3	23 326 096	29 488 396	7428	2436	1.3
C50D10R-4	29 786 931	37 539 837	6964	2589	10.8
C50D10R-5	32 424 168	40 615 200	*Out of memory in pre-solving*		
C50D10RR-1	29 334 101	36 740 935	7246	2102	2.0
C50D10RR-2	23 276 170	29 241 357	6988	1428	1.9
C50D10RR-3	16 598 376	21 158 130	5836	2690	0.9
C50D10RR-4	26 917 315	33 724 624	6065	2216	1.8
C50D10RR-5	21 809 063	27 453 700	5423	3477	1.8
C50D10T-1	25 577 765	32 155 063	7470	1072	3.5
C50D10T-2	21 532 491	27 142 076	2165	2232	2.2
C50D10T-3	24 865 670	31 509 935	7224	840	7.8
C50D10T-4	25 822 140	32 858 879	2609	2096	2.8
C50D10T-5	29 364 679	36 968 388	*Out of memory in pre-solving*		
C50D10TT-1	19 436 190	24 481 452	7436	672	3.1
C50D10TT-2	20 734 496	26 234 674	4825	2571	1.8
C50D10TT-3	21 890 404	27 624 220	7481	417	3.9
C50D10TT-4	15 037 952	19 343 530	2301	746	2.6
C50D10TT-5	20 682 913	26 158 409	7473	669	3.2

16 | Branch-and-Price-and-Cut

In the following, we are going to evaluate our branch-and-price-and-cut approach that has been described in Chapter 8. As stated earlier, our solver has been build on the framework SCIP (Achterberg 2009b) by providing customized plugins for variable pricing, branching and cut separation. Since most of our implementation has been done in Java, we forked and extended the JSCIPopt Java API (`https://github.com/scipopt/JSCIPOpt`) for integrating with SCIP.

SCIP ships with a multitude of plugins for different purposes. While we used the built-in heuristics for finding feasible solutions, we deactivated all plugins concerned with pre-solving, cut separation or domain propagation, since they cannot be used when doing column generation without additional effort. For solving LPs, we linked SCIP against Gurobi as LP-solver. Furthermore, we configured SCIP (or Gurobi) to always use a primal simplex algorithm, since that showed the best convergence behavior (see Figure 8.1).

In contrast to our branch-and-cut experiments, we followed the suggestion of Chouman, Crainic, and Gendron (2016) and aggregated all commodities by the single-source-single-sink approach. We assumed that the benefits of the path-based formulation introduced in Section 8.1 outweigh the higher memory consumption implied by a larger MIP (see Section 18.1). This assumption has been supported by other researchers (Alvelos 2005; Chouman, Crainic, and Gendron 2016). For path search, we used the algorithms and configurations that have been proven to be superior in Chapter 14

Similar to the experiments that have been carried-out for the branch-and-cut approach, we always used a warm-start solution that has been created by our instance generator. Furthermore, the costs M^k for the shortcut arcs have been calculated equally using the formula introduced in Equation (15.1). All experiments have been executed with JMH by forking a dedicated JVM for each experiment and performing a single measurement iteration. Again, we split the available memory

into 75 % for the JVM as heap and 25 % native memory used by SCIP. Please note that all SCIP plugins implemented in Java will use the JVM's memory to manage internal state. The available memory has been configured depending on the instance used in the experiment. Please see Table 22.1 in the appendix for more details concerning this configuration. Finally, most of the parameters that are offered to configure SCIP have been left at their default value, except the parameter `limits/gap` which is a threshold for the MIP gap and used as stopping criterion. We decreased its value from 10^{-9} to 10^{-4}, which has been assumed to be a better balance between accuracy and computing time for a practical approach and is the value suggested by Gurobi that also has been used for the experiments in Chapter 15.

Whenever reporting results from our experiments, we used line charts, box plots or performance profiles for visualization. The performance of the latter has been measured differently depending on the context and may be given as computing time or remaining MIP gap. Computing times are always reported as the time required by SCIP to solve the given problem, so they will not contain times required for reading files and creating the initial master problem or times spent during preprocessing (except for the preprocessing done by SCIP internally).

Since the branch-and-cut approach was able to solve most small benchmark instances but failed for the medium-sized ones, we focused on the later to evaluate, if the branch-and-price-and-cut approach is more suitable in handling large instances. All experiments for tuning parameters or evaluating implementation options have been performed on instances of class C10D10 or C50D10. Finally, we examined if we are able to solve the largest instance classes C100D30 and C500D30.

16.1 Column Removal

We start our computational study with the evaluation of the column aging mechanism provided by SCIP. Since we assumed that a noticeable amount of generated variables will not occur within an optimal solution, there might be a benefit in removing variables from an LP that has been non-basic for some iterations. By removing such variables solving an LP might be accelerated, however removing variables and managing them afterwards (e.g., incorporating them again if required) will consume some computing time.

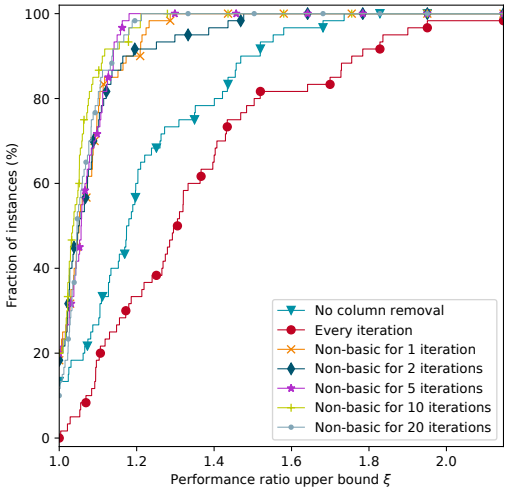


Figure 16.1: Performance profile comparing computing times for different settings of column aging. We measured the computing times required for solving the root LP-relaxation to optimality.

In our experiments, we solved the LP-relaxation of the root nodes of all C50D10 benchmark instances to optimality. For this, we performed column-and-row generation without any additional techniques like cyclic pricing or cut separation. In every pricing iteration, we performed a path search for each commodity and added a limited amount of the resulting columns to the LP. Here we relied on SCIPs default settings and selected (at most) 2000 columns by their reduced costs. An experiment ends with an optimal solution for the root nodes LP-relaxation, if no more columns could be found in an iteration.

Since we are interested in the impact of column aging for the overall solution process, we configured the amount of iterations a variable is allowed to stay in the LP without being basic. Within our experiment the configuration has been varied from never removing columns, to removing them after some iterations to removing all non-basic columns in each iteration. Figure 16.1 presents the result of this experiment. Obviously, removing columns in every iteration shows the worst performance. However, never removing any columns is also clearly dominated by all other runs that sometimes remove columns. The performance of these runs seem to be more or less comparable, with a slight disadvantage for the runs removing columns after one or two iterations. We wondered why there is no clearer deviation between these runs like the one visible when removing columns in each iteration. There might be a noticeable amount of columns that leave the basis only shortly

and must be incorporated again after they have been removed. Nevertheless, we did not investigate this any further, but keep it for future research. Instead, we decided to follow the suggestion of SCIP and continue with its default value of removing columns after 10 iterations. So whenever a column has not been contained in the basis of the LP solution in one of the last 10 iterations, it will be removed from the LP and moved to the column store. From there, it will be automatically reincorporated into the LP by SCIP as soon as it shows negative reduced costs. Such reincorporation takes place before the variable pricer will be called, hence it can be ensured that the same column will not be generated twice. We used this configuration for the remainder of our computational study.

16.2 Parallelized Pricing

In our next experiment, we examined the performance impact of parallelizing the variable pricer, following the algorithm described in Section 8.6. We solved the root node's LP-relaxation of all C10D10 instances to optimality and measured the computing time spent by the variable pricer, when varying the amount of used CPU cores. For solving, we only used column-and-row generation without cut separation. In each iteration a full pricing has been performed and all found variables are

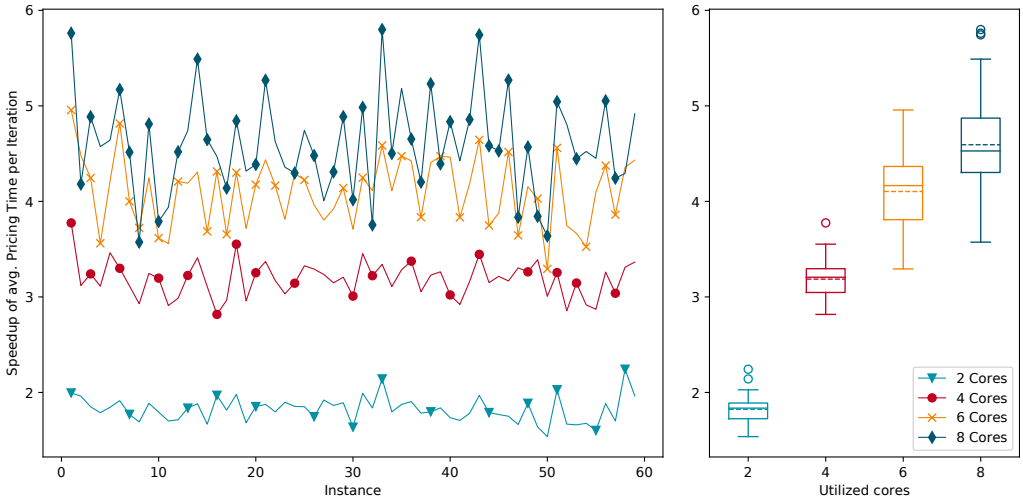


Figure 16.2: Chart showing the speed-up of pricing when varying the amount of used cores for solving the root node LP-relaxation on all C10D10 benchmark instances.

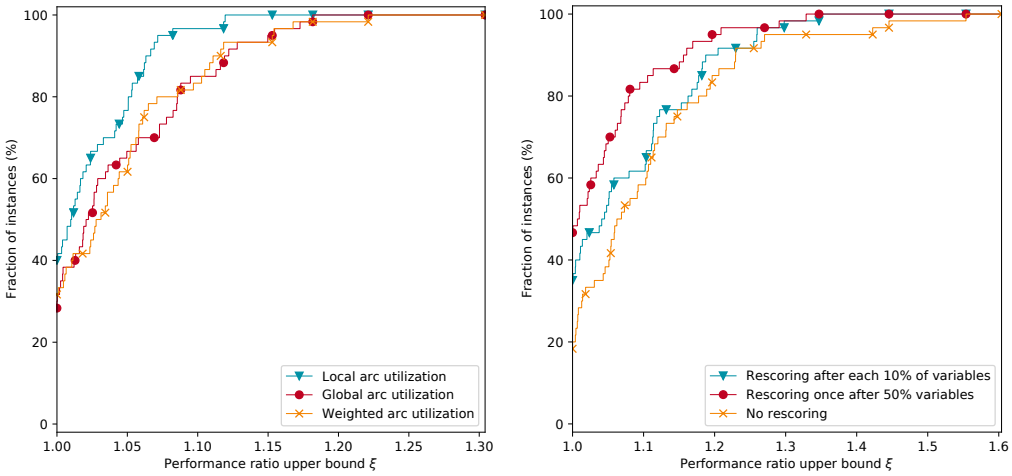
incorporated into the LP. This should lead to only a few time-consuming pricing iterations that are dominant within the overall solving process and thus are a good basis for comparing the time spent in pricing for different configurations of parallelization.

Figure 16.2 shows the speed-up when moving from a serial pricing approach to a multi-threaded approach utilizing up to 8 CPU cores (in all experiments computer memory has been limited to 32 GB). For each instance, we measured the average computing time spent for pricing variables and calculated the speed-up for the different configurations. Obviously, the speed-up is sub-linear in the number of cores, which is an expected result and typical for most parallelized algorithms in accordance with the law of Amdahl (1967). Furthermore, the variance of the speed-up between different instances increases with a growing amount of cores and it does not seem to be reasonable to use significantly more than 8 cores in parallel. In our view, the gained speed-up does not justify the additional hardware resources, when using more cores. For the remainder of our computational study, we used 8 cores as the upper limit during all experiments, for some smaller instances even less cores were used.

16.3 Scoring of Candidate Columns

In previous experiments, only reduced costs have been used to select the most promising candidates that will be incorporated into the LP from all priced variables. In the following, we are going to evaluate the alternative scoring strategies that have been presented in Section 8.6. For all experiments, the root node's LP-relaxation will be solved to optimality by using only column-and-row generation and performing full pricing iterations, that will lead to a limited amount of maximum 2000 variables that are added to the LP. We used all C50D10 instances for this experiment.

Figure 16.3a indicates that from the three alternative scoring approaches introduced earlier the local arc utilization score performs best. Furthermore, re-calculating the score once after half of the variables have been selected is favorable compared to not re-scoring at all and re-scoring more often (see Figure 16.3b). Nevertheless, when comparing these alternative scoring approaches to the classical score based on reduced costs in Figure 16.4, it is obvious that reduced costs are the best measure



(a) (b)

Figure 16.3: Performance profiles of computing times for different scoring strategies that are used to select new columns during pricing are presented in (a), while (b) compares the impact of re-calculating the score of the columns after a predefined amount of variables have been added to the LP.

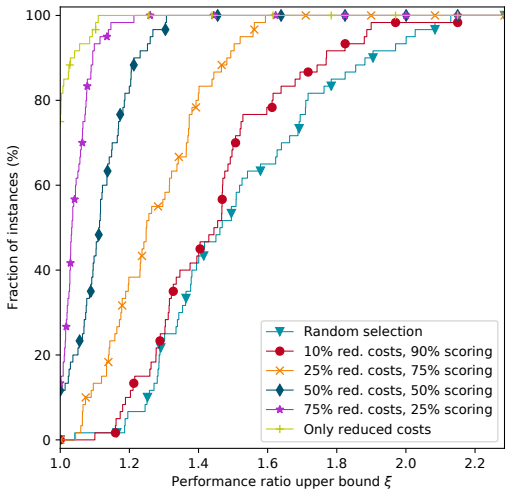


Figure 16.4: Performance profiles comparing the selection of new columns by scoring with the selection by reduced costs. Profiles are based on computation times for solving the root LP-relaxation to optimality. The performance of a random selection is presented as baseline.

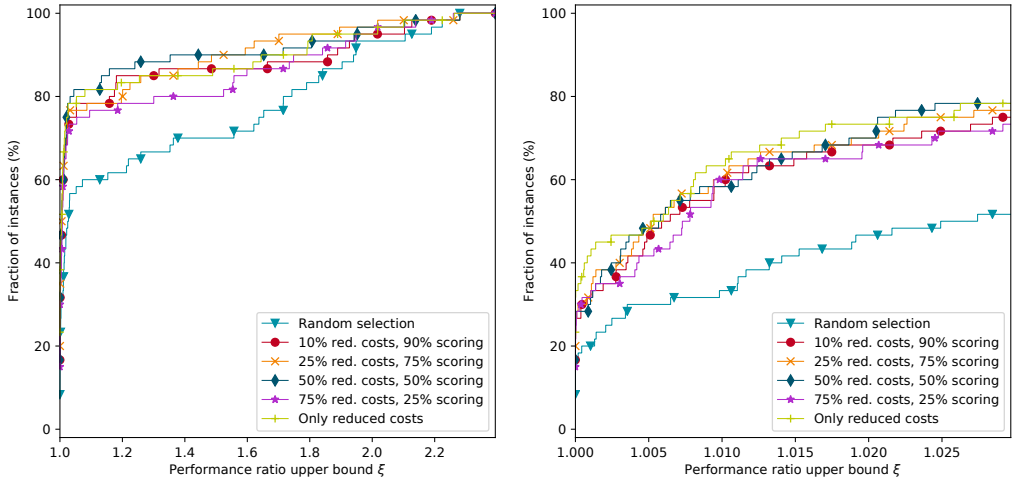


Figure 16.5: Performance profiles comparing the selection of new columns by scoring with the selection by reduced costs. Profiles are based on remaining MIP gaps after solving the MIP with a time limit of 4 hours. The performance of a random selection is presented as baseline. While (a) shows the full profile, (b) restricts the view to the range $\xi \in [1, 1.03]$

for selecting variables. The higher the amount of variables that have been selected by alternative scoring (regardless which of the three scores is used) the lower is the overall performance of the approach. When only 10 % of the variables are selected by reduced costs, the performance is comparable to a random selection. Hence, we see limited potential that one of our scoring approaches can have an advantage when solving LPs.

Nevertheless, as we are solving a MIP and not an LP, we repeated our last experiment and tried to solve the MIPs of all C50D10 instances with a time limit of 4 hours, instead of solving the root node's LP-relaxation only. Figure 16.5 shows the performance profiles comparing the remaining MIP gap of all approaches. Although the alternative scores were not able to beat the selection by reduced costs, they are at least competitive. We assume that the alternative approaches bring some diversity to the paths available for a commodity, which might help finding good feasible solutions. Even though solving the node's LP-relaxation requires more computing time, the overall process of finding an integer solution can be accelerated by such approaches. Unfortunately, we did not reach our goal of beating the classical reduced costs selection. However, there might be more promising

alternative approaches for variable selection. Hence, we keep that question open for further research and will use the reduced costs approach for the remainder of our study.

16.4 Cyclic Pricing

Until now, we performed only full pricing iterations to evaluate the performance of different implementation options. In the following, we are going to examine the impact of doing cyclic pricing and solving only a subset of the sub-problems in each iteration ($\varrho_{min} < |K|$). For this purpose, we varied the amount of new variables χ that can be incorporated into the LP in each iteration from 500 to 10 000 and chose $\varrho_{min} = \alpha \cdot \chi$ with α having a different value from set $\{1, 2, 3, 4\}$ in each of our experiments. Again, we solved the LP-relaxation of the root node of all benchmark instances from class C50D10 and compared the computing times required to find an optimal solution.

Figure 16.6 shows the performance profiles of the best performing configurations from these experiments. There seems to be no configuration that clearly dominates the others, however obviously it is beneficial to price more variables than those which will be added to the LP ($\alpha > 1$). Thus, having options to choose from

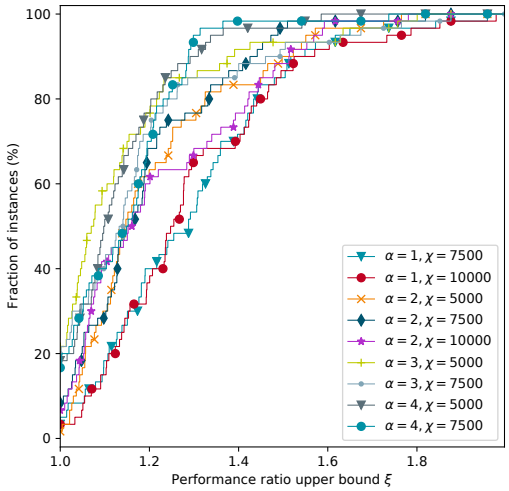


Figure 16.6: Performance profiles comparing the computation times when solving the root LP-relaxation for the best performing settings of parameters α and χ.

promising variables seems to be more important than saving computing time by reducing the amount of solved sub-problems. Furthermore, we worked out that it is highly specific for an instance if it is favorable to choose a larger or smaller value for χ. Unfortunately, we were unable to identify the characteristics of an instance that should drive this decision or develop a formula. Since the configuration $\alpha = 3$ and $\chi = 5000$ performed best within our experiments, we decided to accept this setting for all further experiments.

16.5 Generating Cuts

As we discovered within our branch-and-cut approach that separating any violated inequalities is hard except for strong linking cuts (compare Section 15.2), we first wanted to investigate if our separation algorithms are more successful when doing branch-and-price-and-cut. Hence, we solved the root node's LP-relaxation of all C50D10 instances with a time limit of two hours and counted all cuts that have been separated. Figure 16.7 shows the result of this experiment. Please note that we plotted all cuts and not only the ones that have been selected by SCIP to enter the LP. The results look quite similar to what has been examined within our branch-and-cut approach (see Figure 15.5). Obviously, separating strong linking

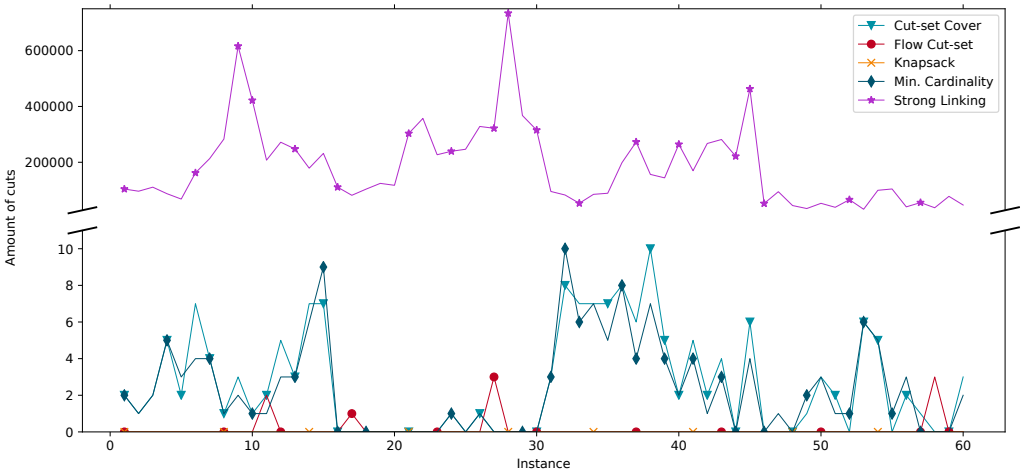


Figure 16.7: Chart showing the amount of different cuts that have been separated when solving the root node LP-relaxation of all C50D10 benchmark instances. Please note that the vertical axis has been split to reflect the huge difference between strong linking and all other cuts.

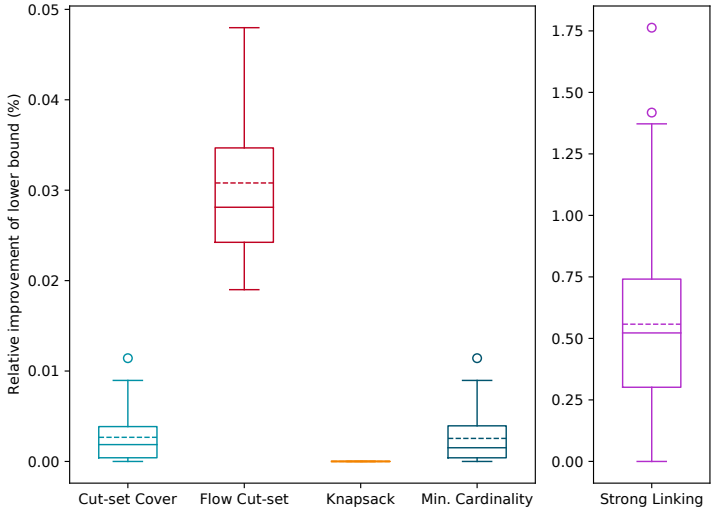
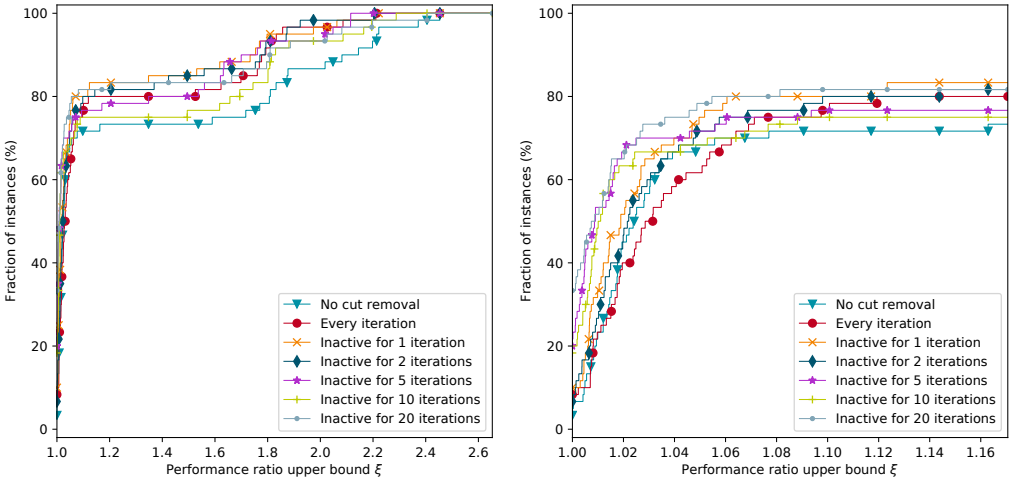


Figure 16.8: Box plots showing the relative improvement of the lower bound when separating cuts in the root node of all C50D10 benchmark instances.

cuts is easy and there are several hundreds of thousands for each instance. In contrast, for all other types of inequalities the algorithms did not find any or just a few violated ones. Even less have been selected by SCIP for being incorporated into the LP. Hence, it is no surprise that the impact of the cuts on the lower bound varies heavily. While strong linking cuts lead to a relative improvement of the lower bound of up to 1.75 %, all other cuts induced an improvement, which did not exceed 0.05 %. Further details are depicted in Figure 16.8. Since our separation algorithms are rarely successful in finding violated inequalities, but consume a considerable amount of computing time, and the corresponding cuts have a very limited impact on the lower bound, we decided to continue our study only with strong linking cuts. Compared to the other types of inequalities, they can be separated quickly and show an impact that might help in solving the MIPs to optimality.

Although SCIP is rigorous when selecting efficacious cuts that actually should enter the LP (typically from 50 000 cuts less than 1000 are selected), their efficacy might decrease in following pricing iterations or they might become redundant since other cuts that have been incorporated. In the end, there might be a considerable amount of cuts which have (nearly) no impact on the LP solution, but slow down the simplex algorithm. Removing such inequalities from the LP by performing row aging might speed up the overall solution process. Figure 16.9 compares different



(a) (b)

Figure 16.9: Performance profiles comparing remaining MIP gaps for different settings of row aging. We measured the remaining MIP gap after solving the root node's LP-relaxation to optimality. While (a) shows the full profiles, (b) restricts the view to the range $\xi \in [1, 1.2]$

settings of the allowed amount of iterations a cut might be inactive before it will be removed from the LP, ranging from never removing any cuts, to removing all inactive cuts in each iteration, to removing them after a few iterations of inactivity. The results are less clear than they have been for column aging (see Figure 16.1). Nevertheless, there seems to be a minor advantage in removing rows after 20 iterations of inactivity, so we will follow this approach for the remainder of this study.

When using branch-and-price-and-cut without limiting the amount of cuts or cut separating iterations per node, we examined that solving the LP-relaxation of a node can take a considerable amount of time. Since violated strong linking cuts can be found easily, the probability is high that cut separation will find cuts that enter the LP, which will trigger another round of pricing. This can lead to a long-running loop of pricing and cut separation. Figure 16.10 visualizes the results of such a loop, that has been recorded when solving instance C50D10RL-5 with a time limit of 24 hours. Obviously, there are numerous iterations that add priced variables or separated cuts to the LP. However, improvements of the lower bound are largest within the first few iterations. Furthermore, the time between consecutive pricing runs increases noticeable, which indicates that more and more time is consumed by

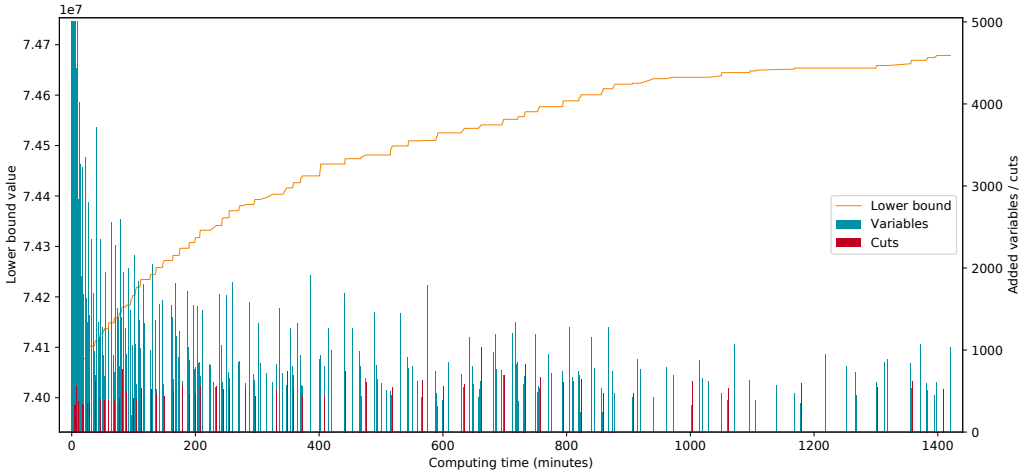


Figure 16.10: Chart showing the development of the lower bound when solving instance C50D10RL-5 with a time limit of 24 hours. The bars indicate the amount of priced variables and separated cuts in each iteration of the price-and-cut loop.

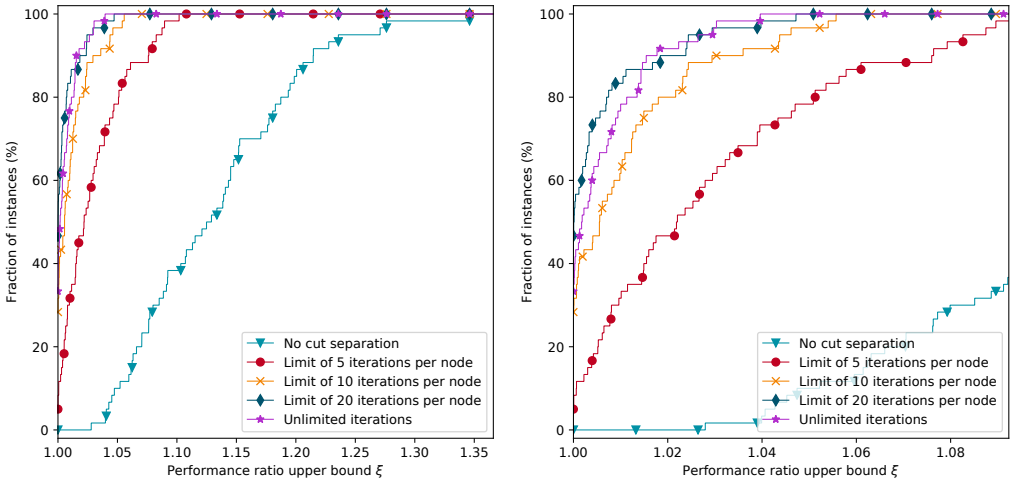


Figure 16.11: Performance profiles comparing the remaining MIP gap when solving all instances of class C50D10 with a time limit of 4 hours using different configurations limiting the amount of cut separating iterations per node. As baseline, we present also the case of separating no cuts at all. While (a) shows the full profiles, (b) restricts the view to the range $\xi \in [1, 1.1]$

the simplex algorithm to solve the reduced master problem. In the end, not even the root node could have been solved within the given time limit. We observed a similar behavior for other instances that have been randomly selected from class C50D10.

Finally, we have evaluated the impact of different limits for the amount of cut separating iterations on the overall performance. Within that experiment we configured SCIP to use its default values of separating maximum 2000 cuts per iteration in the root node and 100 cuts in all other nodes, to limit the growth of the LPs. Figure 16.11 shows the performance profiles of the different configurations when solving all instances of class C50D10 with a time limit of 4 hours. These profiles indicate that it is beneficial to separate cuts and to impose no limit on the amount of iterations. As for most instances, there have been less than 20 iterations within the time limit, it is no surprise that this configuration performed similar to the no limit case. Overall, this observation supports our assumption that there should be no limit and we will follow this approach in the following studies.

16.6 Early Branching

Besides limiting the amount of cut separating iterations, another opportunity for breaking out of an long-running price-and-cut-loop is to stop the pricing of further variables. However, this technique (called early branching) requires the calculation of a lower bound to the optimal solution value of the current LP. Otherwise, SCIP will not be able to determine the bound given by the current node of the branch-and-bound tree. The Lagrangian bound can be used as such lower bound, albeit its calculation requires a full pricing iteration. Hence, in the next experiment we configured our pricer to solve all sub-problems every 10th iteration (full pricing) and to perform early branching, if the gap between Lagrangian bound and current LP solution values is below 1 % or 2 %. Please note that we consider early branching only within the root node, since in all previous experiments we noticed that in other nodes there has been a very limited amount of pricing iterations anyway.

Figure 16.12 compares the performance of these approaches to an alternative without early branching. Avoiding early branching leads to the best improvements in the lower bound of most benchmark instances. Combining this observation with the results concerning cut separation presented in Figure 16.11, we assume that joint

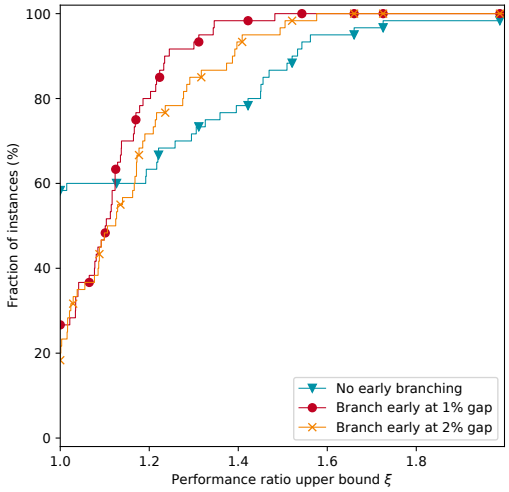


Figure 16.12: Performance profiles comparing the remaining MIP gap when solving all instances of class C50D10 with a time limit of 4 hours using different configurations for early branching.

pricing and cut separation has a larger impact on the bound than branching does. Nevertheless, the performance profiles indicate that early branching (regardless of the used gap) performs quite well on at least some of the instances. Unfortunately, we did not figure out what classifies an instance to benefit from early branching. Hence, we will not use it for the remainder of this study.

16.7 Best Configuration

In Sections 16.1 to 16.6, we have performed numerous experiments for evaluating implementation alternatives and for tuning parameters of our branch-and-price-and-cut solver. In our last experiment for this approach, we are going to examine how our implementation performs on the largest instances of our benchmark set (classes C100D30 and C500D30). Since we expect that we will not be able to solve them to optimality, we are mainly interested in the remaining MIP gap and its applicability to practice. Furthermore, we wondered if we will face similar out-of-memory issues we struggled with in experiments for our branch-and-cut approach. We used the configuration that has proven to perform best in our tuning experiments above, which can be summarized as follows:

- In each iteration at most 15 000 new variables are priced using cyclic pricing and only the 5000 most promising will be incorporated into the LP.
- The selection of these promising variables is based on their reduced costs only, hence no scoring approach is used.
- Pricing is performed in parallel on an instance-specific amount of cores.
- Column aging is activated and removes non-basic columns from the LP after 10 iterations.
- In each iteration at most 2000 strong linking cuts are separated.
- Row aging is activated but only for cuts and removes them from the LP after 20 iterations of inactivity.
- Early branching is not used at all and the amount of pricing or cut separating iterations is not limited.
- Above amounts of priced variables and separated cuts are used only at the root node but are reduced to 500 variables and 100 cuts for the other nodes.

Table 16.1 lists the results of our computations for all large instances with a time limit of 12 hours (which we assume to be the maximum computing time for usage in practice). Although we were able to initiate the solving process of the root node within our branch-and-price-and-cut solver in all experiments, we faced similar out-of-memory issues as described in Chapter 15 for the majority of C500D30 instances. For some instances there could not be a lower bound determined within 12 hours of computation (marked with "LB") or the MIP gap is excessively large (more than 20 %). There have been no cuts separated in any of the experiments and for some of them even no pricing run has been completed (see instance C500D30TT-1). In contrast, solving instances of class C100D30 worked well without any computational issues, nevertheless, the determined gaps are not tight. The best one has been 3 % for instance C100D30RRLL-5, while the worst has been 31.3 % for instance C100D30TT-5.

Table 16.1: Table showing the final size of the MIPs generated for each benchmark instance, the amount of priced columns and separated cuts and the quality of the best found solutions. In case a benchmark failed, we list the initial size of the MIP instead. Gaps have been calculated within a time limit of 12 hours. Instances for which no gap could be reported, due to a missing lower bound, are marked with "LB".

Instance	**Rows**	**Columns**	**Priced columns**	**Cuts**	**Gap (%)**
C100D30R-1	39 350	513 024	457 180	2662	7.0
C100D30R-2	63 629	471 743	392 570	2570	9.7
C100D30R-3	65 651	382 843	310 520	3905	4.8
C100D30R-4	65 500	454 374	389 911	4539	7.6
C100D30R-5	62 320	397 484	326 657	2600	9.4
C100D30RR-1	74 488	338 675	270 510	4364	6.7
C100D30RR-2	58 042	319 339	258 579	2982	11.6
C100D30RR-3	55 659	374 373	320 485	5446	12.8
C100D30RR-4	52 020	313 139	261 913	4720	9.1
C100D30RR-5	56 326	386 685	305 888	3172	10.3
C100D30RL-1	102 490	466 956	354 115	4042	10.6
C100D30RL-2	151 667	440 522	285 797	720	4.6
C100D30RL-3	98 154	509 070	398 247	3971	7.8
C100D30RL-4	114 710	559 041	420 129	4202	6.7
C100D30RL-5	82 184	500 090	401 582	3768	9.0
C100D30RLL-1	93 040	450 132	316 896	5203	6.7
C100D30RLL-2	107 903	626 801	489 164	3414	4.6
C100D30RLL-3	124 316	497 514	352 021	2016	7.6
C100D30RLL-4	83 719	591 116	471 365	3931	14.3
C100D30RLL-5	113 359	452 954	326 758	2332	4.8
C100D30RRL-1	111 969	500 343	356 247	5017	6.2
C100D30RRL-2	88 984	610 506	479 238	10 645	5.1
C100D30RRL-3	125 907	472 732	335 641	5495	12.5
C100D30RRL-4	70 597	538 268	447 789	7282	3.8
C100D30RRL-5	104 910	375 201	252 834	5764	7.5
C100D30RRLL-1	121 680	493 818	366 556	4615	14.3
C100D30RRLL-2	99 247	542 076	419 767	6181	5.2
C100D30RRLL-3	96 765	441 372	320 907	4649	5.4
C100D30RRLL-4	89 414	538 657	400 305	2510	3.2
C100D30RRLL-5	82 087	501 377	367 962	4780	3.0

Instance	Rows	Columns	Priced columns	Cuts	Gap (%)
C100D30T-1	55 469	416 115	354 415	2645	7.8
C100D30T-2	57 184	476 710	404 764	0	8.4
C100D30T-3	80 927	363 606	280 959	1660	12.8
C100D30T-4	37 023	420 456	369 865	4949	9.9
C100D30T-5	88 035	370 617	292 870	1391	10.7
C100D30TT-1	62 066	384 539	315 127	1048	7.9
C100D30TT-2	56 880	452 977	396 698	2847	7.9
C100D30TT-3	42 637	392 324	343 639	1631	9.4
C100D30TT-4	61 618	312 614	241 910	806	11.5
C100D30TT-5	52 587	293 124	230 000	0	31.3
C100D30TL-1	97 451	511 576	394 541	3126	5.0
C100D30TL-2	89 565	646 640	526 241	2637	7.4
C100D30TL-3	128 623	531 630	381 282	2492	9.4
C100D30TL-4	101 550	476 827	351 458	2697	5.2
C100D30TL-5	80 192	520 704	386 881	2904	4.0
C100D30TLL-1	113 967	453 098	300 927	759	6.1
C100D30TLL-2	81 166	592 401	508 214	4330	9.1
C100D30TLL-3	83 820	562 409	475 294	2455	9.2
C100D30TLL-4	86 028	506 033	382 614	3439	5.0
C100D30TLL-5	105 797	507 959	365 819	3111	7.9
C100D30TTL-1	108 344	433 731	307 495	813	9.1
C100D30TTL-2	121 984	465 170	324 311	825	9.2
C100D30TTL-3	125 215	512 472	373 051	843	8.7
C100D30TTL-4	96 662	426 001	321 917	620	3.7
C100D30TTL-5	92 152	458 127	367 917	858	9.2
C100D30TTLL-1	107 145	495 142	339 722	1289	7.2
C100D30TTLL-2	94 804	682 157	529 858	1600	4.9
C100D30TTLL-3	90 317	562 911	437 174	647	4.3
C100D30TTLL-4	84 456	429 839	330 674	687	4.7
C100D30TTLL-5	107 575	430 464	296 653	686	9.0
C500D30R-1	477 654	591 177	90 000	0	*LB*
C500D30R-2	459 632	550 128	60 000	0	*LB*
C500D30R-3	420 978	549 227	60 000	0	*LB*
C500D30R-4	401 153	555 767	85 000	0	*LB*
C500D30R-5	412 882	551 927	70 000	0	*LB*
C500D30RR-1	399 316	660 141	155 000	0	19.7

Instance	Rows	Columns	Priced columns	Cuts	Gap (%)
C500D30RR-2	402 846	510 989	*Out of memory during solving*		
C500D30RR-3	342 352	620 063	160 000	0	22.3
C500D30RR-4	395 933	485 428	*Out of memory during solving*		
C500D30RR-5	412 442	633 791	150 000	0	31.1
C500D30RL-1	523 382	797 460	*Out of memory during solving*		
C500D30RL-2	533 032	719 818	*Out of memory during solving*		
C500D30RL-3	408 243	626 389	*Out of memory during solving*		
C500D30RL-4	433 377	534 066	*Out of memory during solving*		
C500D30RL-5	525 271	723 998	*Out of memory during solving*		
C500D30RLL-1	383 751	507 977	*Out of memory during solving*		
C500D30RLL-2	481 945	621 173	*Out of memory during solving*		
C500D30RLL-3	433 780	645 107	*Out of memory during solving*		
C500D30RLL-4	496 397	668 873	*Out of memory during solving*		
C500D30RLL-5	531 327	787 180	*Out of memory during solving*		
C500D30RRL-1	385 835	514 332	*Out of memory during solving*		
C500D30RRL-2	434 801	666 784	*Out of memory during solving*		
C500D30RRL-3	531 780	759 078	*Out of memory during solving*		
C500D30RRL-4	485 518	767 749	*Out of memory during solving*		
C500D30RRL-5	495 384	796 627	*Out of memory during solving*		
C500D30RRLL-1	466 085	796 152	*Out of memory during solving*		
C500D30RRLL-2	391 889	551 072	*Out of memory during solving*		
C500D30RRLL-3	387 384	526 946	*Out of memory during solving*		
C500D30RRLL-4	480 217	765 092	*Out of memory during solving*		
C500D30RRLL-5	443 005	613 854	*Out of memory during solving*		
C500D30T-1	467 324	535 061	30 000	0	*LB*
C500D30T-2	412 882	460 160	35 000	0	*LB*
C500D30T-3	307 174	473 869	60 000	0	*LB*
C500D30T-4	359 595	491 749	55 000	0	*LB*
C500D30T-5	467 220	557 429	35 000	0	*LB*
C500D30TT-1	364 877	466 235	0	0	*LB*
C500D30TT-2	378 778	489 508	10 000	0	*LB*
C500D30TT-3	440 402	543 687	5000	0	*LB*
C500D30TT-4	286 800	457 697	15 000	0	*LB*
C500D30TT-5	406 676	510 243	10 000	0	*LB*
C500D30TL-1	460 184	748 489	*Out of memory during solving*		
C500D30TL-2	505 801	683 875	*Out of memory during solving*		

Instance	Rows	Columns	Priced columns	Cuts	Gap (%)
C500D30TL-3	423 015	523 119	*Out of memory during solving*		
C500D30TL-4	499 390	654 670	*Out of memory during solving*		
C500D30TL-5	451 451	703 450	*Out of memory during solving*		
C500D30TLL-1	499 073	608 256	*Out of memory during solving*		
C500D30TLL-2	464 499	664 457	*Out of memory during solving*		
C500D30TLL-3	549 020	682 605	*Out of memory during solving*		
C500D30TLL-4	477 336	643 853	*Out of memory during solving*		
C500D30TLL-5	470 516	663 066	*Out of memory during solving*		
C500D30TTL-1	513 611	745 157	*Out of memory during solving*		
C500D30TTL-2	350 151	574 098	*Out of memory during solving*		
C500D30TTL-3	594 424	775 387	*Out of memory during solving*		
C500D30TTL-4	529 038	712 071	*Out of memory during solving*		
C500D30TTL-5	429 230	683 318	*Out of memory during solving*		
C500D30TTLL-1	462 634	598 715	*Out of memory during solving*		
C500D30TTLL-2	452 687	600 246	*Out of memory during solving*		
C500D30TTLL-3	497 105	647 354	10 000	0	*LB*
C500D30TTLL-4	581 619	766 774	*Out of memory during solving*		
C500D30TTLL-5	537 761	675 268	*Out of memory during solving*		

17 | Comparison of Exact Approaches

In Chapters 15 and 16, we examined the performance of two different exact approaches to solve the PCIMCFND to optimality by using either branch-and-cut or branch-and-cut-and-price. In the following, we will compare both approaches and give some hints regarding their applicability for problem instances of different size.

The branch-and-cut approach evaluated in Chapter 15 is able to solve all small instances of class C1D5 and C5D5 to optimality within a reasonable amount of time using limited hardware resources. Although an optimal solution could not be found for the majority of C10D10 instances, the solutions are always near-optimal. However, this approach is not applicable to larger instances of class C50D10, C100D30 or C500D30, which fail due to memory issues even when using a decent amount of main memory. A detailed listing of all instances and their solutions can be found in the appendix in Table 21.1.

In contrast, the branch-and-price-and-cut approach examined in Chapter 16 is able to handle all classes up to the large C100D30 instances. For the smallest ones of class C1D5, it reliably found the optimal solution, although it required more computing time than branch-and-cut. For C5D5 instances finding an optimal solution within the time limit is seldom and it occurred only once for the C10D10 instances. Additionally, branch-and-price-and-cut sometimes required more hardware resources to solve instances from these classes. Table 17.1 shows a direct comparison for class C10D10. There are only two instances (C10D10RL-2 and C10D10RL-3, highlighted in bold) for that branch-and-price-and-cut was able to provide a better solution than branch-and-cut. Although the gaps are tight for all C10D10 instances using any of both methods, branch-and-cut performs better in most cases. Hence, we suggest to use our branch-and-cut approach whenever the size of the MIP and the available hardware resources allow the application of this algorithm.

The situation changes when looking at the larger instances. Only branch-and-price-and-cut is able to handle classes C50D10 and C100D30 and showed memory issues only for the largest class C500D30 (see Table 22.1). Hence, there is no alternative to this approach. Nevertheless, the MIP gaps of these classes' solutions are not tight and we wondered if they will be acceptable in practice. However, to evaluate this in more detail, a comparison of the results against a human logistics planner would be required. We keep this question open for future research.

Table 17.1: Table comparing computing times and remaining MIP gap of the branch-and-cut and the branch-and-price-and-cut solver. While computing times are given in minutes and have been limited to 12 hours, MIP gaps are presented in percent.

	Branch-Cut		**Branch-Price-Cut**		
Instance	**Time**	**Gap**	**Time**	**Gap**	**ΔGap**
C10D10R-1	*Limit*	0.172	*Limit*	1.179	1.007
C10D10R-2	46.2	-	*Limit*	0.324	0.324
C10D10R-3	628.9	-	*Limit*	0.255	0.255
C10D10R-4	*Limit*	0.401	*Limit*	0.632	0.232
C10D10R-5	66.4	-	*Limit*	0.557	0.557
C10D10RR-1	*Limit*	0.811	*Limit*	1.805	0.994
C10D10RR-2	*Limit*	0.213	*Limit*	0.414	0.201
C10D10RR-3	171.4	-	*Limit*	0.402	0.402
C10D10RR-4	*Limit*	0.528	*Limit*	0.746	0.218
C10D10RR-5	*Limit*	0.407	*Limit*	0.499	0.091
C10D10RL-1	*Out of memory*		*Limit*	0.74	-
C10D10RL-2	*Limit*	0.79	*Limit*	0.537	**-0.253**
C10D10RL-3	*Limit*	0.969	*Limit*	0.629	**-0.34**
C10D10RL-4	205.4	-	*Limit*	0.138	0.138
C10D10RL-5	145.9	-	*Limit*	0.282	0.282
C10D10RLL-1	*Limit*	0.18	*Limit*	0.345	0.165
C10D10RLL-2	*Limit*	0.105	*Limit*	0.215	0.11
C10D10RLL-3	*Limit*	0.439	*Limit*	0.556	0.117
C10D10RLL-4	*Limit*	0.143	*Limit*	1.515	1.372
C10D10RLL-5	83.2	-	*Limit*	0.534	0.534
C10D10RRL-1	*Limit*	0.332	*Limit*	0.442	0.11

Instance	**Time**	**Gap**	**Time**	**Gap**	**ΔGap**
C10D10RRL-2	*Limit*	0.36	*Limit*	0.521	0.161
C10D10RRL-3	*Out of memory*		*Limit*	1.153	-
C10D10RRL-4	*Limit*	0.279	*Limit*	0.42	0.141
C10D10RRL-5	*Limit*	0.151	*Limit*	0.486	0.335
C10D10RRLL-1	*Limit*	0.587	*Limit*	0.999	0.412
C10D10RRLL-2	*Limit*	0.196	*Limit*	0.591	0.395
C10D10RRLL-3	*Limit*	0.095	*Limit*	0.391	0.296
C10D10RRLL-4	*Limit*	0.289	*Limit*	0.507	0.218
C10D10RRLL-5	*Limit*	0.309	*Limit*	0.426	0.118
C10D10T-1	*Limit*	0.065	*Limit*	0.677	0.611
C10D10T-2	*Limit*	0.133	*Limit*	1.445	1.312
C10D10T-3	*Limit*	0.957	*Limit*	1.131	0.174
C10D10T-4	*Limit*	0.703	*Limit*	1.834	1.13
C10D10T-5	*Limit*	0.749	*Limit*	0.992	0.243
C10D10TT-1	*Limit*	1.145	*Limit*	5.251	4.107
C10D10TT-2	*Limit*	0.089	*Limit*	0.757	0.668
C10D10TT-3	*Limit*	0.44	*Limit*	1.028	0.588
C10D10TT-4	*Limit*	0.108	*Limit*	0.458	0.35
C10D10TT-5	*Limit*	0.131	*Limit*	0.824	0.692
C10D10TL-1	*Limit*	0.185	*Limit*	0.359	0.174
C10D10TL-2	*Limit*	0.544	*Limit*	0.744	0.2
C10D10TL-3	*Out of memory*		*Limit*	1.798	-
C10D10TL-4	570.0	-	*Limit*	0.313	0.313
C10D10TL-5	*Limit*	0.715	*Limit*	0.976	0.262
C10D10TLL-1	*Limit*	1.063	*Limit*	1.13	0.066
C10D10TLL-2	*Limit*	1.786	*Limit*	2.144	0.358
C10D10TLL-3	*Out of memory*		*Limit*	0.565	-
C10D10TLL-4	*Limit*	0.072	*Limit*	0.793	0.721
C10D10TLL-5	*Limit*	0.209	*Limit*	0.465	0.256
C10D10TTL-1	2.6	-	6.7	-	0.0
C10D10TTL-2	164.5	-	*Limit*	0.246	0.246
C10D10TTL-3	*Limit*	0.219	*Limit*	2.177	1.959
C10D10TTL-4	*Limit*	0.148	*Limit*	1.17	1.023

Instance	**Time**	**Gap**	**Time**	**Gap**	**ΔGap**
C10D10TTL-5	88.4	-	*Limit*	1.876	1.876
C10D10TTLL-1	73.1	-	*Limit*	0.99	0.99
C10D10TTLL-2	*Limit*	0.154	*Limit*	0.703	0.549
C10D10TTLL-3	563.4	-	*Limit*	1.39	1.39
C10D10TTLL-4	*Limit*	0.198	*Limit*	0.249	0.051
C10D10TTLL-5	49.9	-	*Limit*	0.259	0.259

18 | Heuristic Approaches

18.1 Commodity Aggregation

In the previous sections we have always used the commodity aggregation that has been described in Section 5.7. Hence, it has been ensured that aggregating does not change the optimal solution of our MIPs. We will revoke that guarantee in the following and evaluate the more relaxed aggregation strategy that has been introduced in Chapter 9. Namely, we allow that commodities $k_1, k_2 \in K$ are aggregated regardless of whether their sets of usable arcs match ($A^{k_1} \neq A^{k_2}$). The resulting aggregated commodity will be able to use all arcs out of the union of those sets. Furthermore, we will ignore all differences concerning lateness or arc costs and we will use the average costs for the aggregated commodities instead. We assume that these relaxations will lead to a noticeable stronger aggregation and finally to a significantly smaller MIP. However, the resulting optimal solution might be sub-optimal for the original problem.

Figure 18.1 presents the differences in the amount of (aggregated) commodities between the exact and the two more relaxed aggregation strategies. Additionally, we visualize the impact of doing a single-source-single-sink aggregation, that has been used for our branch-and-price-and-cut approach, and using the multiple-source-single-sink aggregation from our branch-and-cut implementation. Obviously, for both types of aggregation using a more relaxed strategy leads to a significant reduction in the amount of (aggregated) commodities. For multiple-source-single-sink there are at least 80 % less commodities for the majority of instances, for single-source-single-sink they decrease by 62 %, when ignoring costs and allowed arcs during aggregation. So, for the first approach there are not only fewer commodities within the original formulation, but additionally aggregation performs better. Furthermore, neglecting differences in lateness and arc costs seems to have the largest effect on the amount of aggregated commodities, although the amount can be further reduced by ignoring allowed arcs.

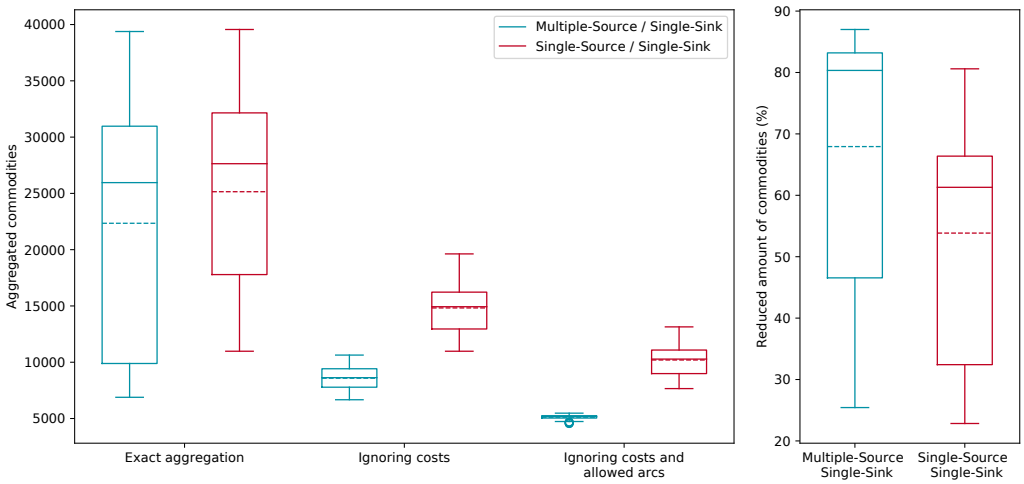


Figure 18.1: Box plots showing the amount of commodities for C50D10 instances after aggregating them in a single-source-single-sink and multiple-source-single-sink fashion using an exact and two more relaxed aggregation strategies. The box plots on the right compare the amount of commodities between the exact approach and the aggregation that ignores costs and allowed arcs.

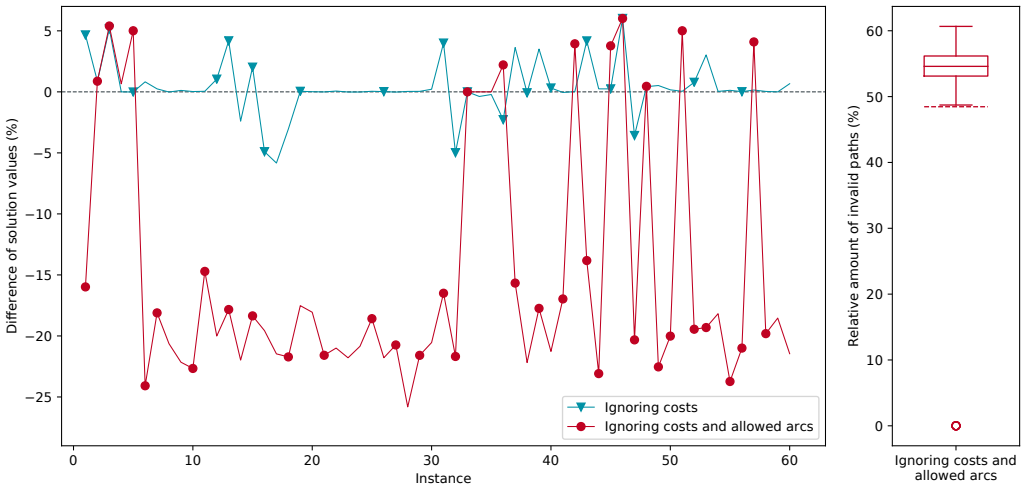


Figure 18.2: Charts showing the difference in (integer) solution values when comparing the best solution acquired with exact commodity aggregation to the ones with more relaxed strategies. The experiment has been performed on all C50D10 instances using branch-and-price-and-cut with a time limit of 12 hours. The box plot on the right shows the relative amount of paths that have been invalid due to the usage of prohibited arcs.

We wondered how that reduction impacts MIP solving and compared the different aggregation strategies, when solving all C50D10 instances with a time limit of 12 hours using our branch-and-price-and-cut approach. We were mainly interested in the difference of the best (integer) solution. However, in case of ignoring allowed arcs during aggregation, we also wanted to examine how many of the commodities' paths were invalid in terms of the original problem, due to the usage of prohibited arcs.

Figure 18.2 shows the relative differences of the best found solution value of the relaxed aggregation strategies to the best solution value found using exact aggregation. When ignoring costs only during aggregation the best found solutions after computing time of 12 hours are within a range of $-6\,\%$ to $6\,\%$ around the best solution found with the exact approach. For some instances even a better solution could be found. This is no surprise, since by reducing the size of the MIPs we were able to close the MIP gap for some of them, which can imply high quality solutions, even though these solutions are found on a modified problem (averaged costs for some aggregated commodities). Nevertheless, we were not able to reduce the MIP gap in a statistically significant manner, as is reported in Figure 18.3.

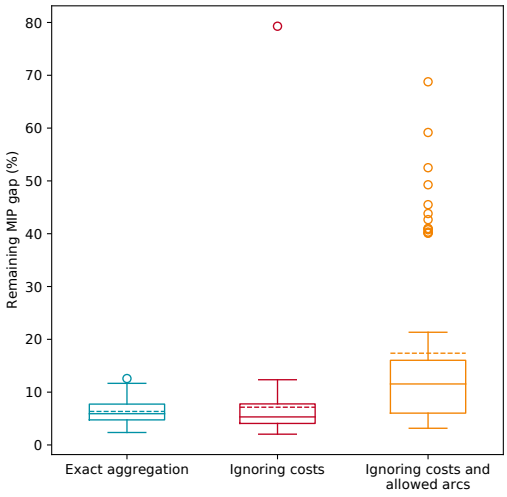


Figure 18.3: Charts showing the remaining MIP gap after 12 hours computing time of our branch-and-price-and-cut solver with exact commodity aggregation and with more relaxed aggregation strategies. The experiment has been performed on all C50D10 instances. We present the MIP gaps as they have been reported by SCIP.

The remaining MIP gap of the exact approach seems to be equally distributed to the aggregation that ignores costs. However, this does not apply to the case of ignoring costs and allowed arcs. Here, we examined considerable larger MIP gaps of up to 70 %. We assume that ignoring allowed arcs implies a noticeable larger solution space (there are more paths allowed for each commodity) and hence leads to a slower convergence to an optimal solution of our branch-and-price-and-cut solver. Figure 18.2 indicates that many of the paths within the best solution use (originally) prohibited arcs, which supports the assumption that there are more paths in the solution space that need to be considered. Furthermore, our experiment revealed that ignoring allowed arcs can significantly reduce the best solution value (up to 25 %), which might be beneficial from a business perspective. Nevertheless, from a mathematical point of view, ignoring allowed arcs leads to MIPs that seem to be harder to solve even though they are smaller, at least when looking at the resulting gaps. Additionally, the largest effect on reducing the size of the MIPs can be achieved by ignoring costs only. Hence, this aggregation strategy might be beneficial, especially if ignoring allowed arcs is not an option due to business operations. The gathered best solutions are comparable to the exact aggregation approach in this case.

18.2 Heuristic Path Search & Pricing

In Chapter 10, we introduced a hierarchical pricing scheme that combines reusing previously generated candidate columns with heuristic and exact path search approaches. In the following, we will examine the performance of this approach and compare it to the "classical" pricing explained in Chapter 8 and to heuristic pricing, that never determines a path exactly. In our experiments, we solve the root node LP-relaxation of all C50D10 instances using these three different approaches. The branch-and-price-and-cut solver has been configured equally to the best-performing configuration used in the experiments of Section 16.4. In accordance with Figure 10.1 the parameters introduced in Chapter 10 have been configured to $\alpha_1 = 1800$, $\alpha_* = 1000$ and $\alpha_\Delta = 500$. Please note that these settings are highly dependent on the problem instance, however, they worked well within our experiments.

Figure 18.4 summarizes the performance of the different pricing approaches by comparing their computing times when solving the root node LP-relaxations.

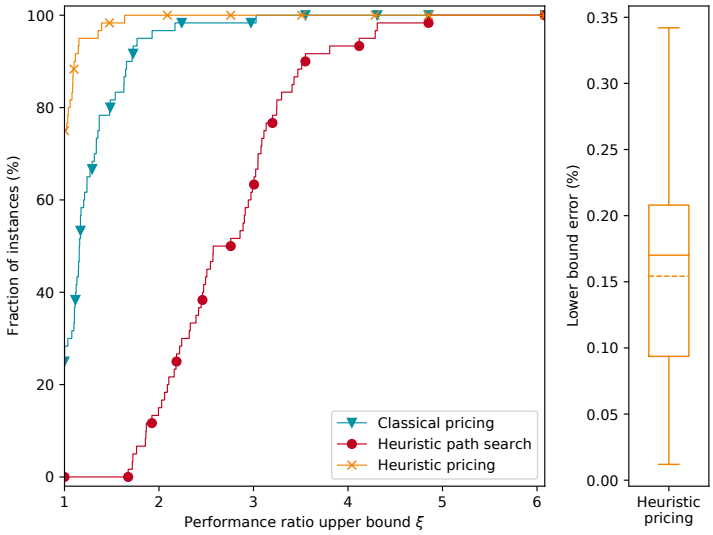


Figure 18.4: Performance profiles reporting computing times for solving the root node LP-relaxation of all C50D10 instances when using our branch-and-price-and-cut solver with heuristic path search algorithms or with heuristic pricing. The box plot shows the statistical distribution of the error in the lower bound that can occur when doing heuristic pricing.

Obviously, using heuristic path search combined with a final exact pricing iteration performs worst, even though a single pricing iteration is noticeable faster than in "classical" pricing. We figured out that the final exact pricing iteration often leads to further heuristic pricing iterations. Hence, although each of the iterations consumes less computing time, the amount of iterations prolongs the overall solving process. In contrast, avoiding the final exact pricing iterations leads to a significant speed-up. Nevertheless, there is a price to pay for this improvement. The error of the lower bound obtained from (heuristically) solving the LP-relaxation in the root node has been up to 0.35 % in our experiments. We assume that this is an acceptable result in practice, however, there is no guarantee that the error will be of similar magnitude for other problem instances.

18.3 Flow Rounding

Chapter 11 summarizes different heuristic approaches to derive an integral solution to PCIMCFND, when a solution with fractional multi-commodity flow is given. We will evaluate the performance of the different algorithms in the following. During

all experiments, we ignored the activation variables y_a in MIP (8.5) and fixed them in post-processing to $y_a = 0$ or $y_a = 1$ depending on whether they are actually required by the multi-commodity flow. Since we were interested in solving the root node LP-relaxation as quickly as possible, we used the tuned configuration for our branch-and-price-and-cut solver (see Chapter 16), but deactivated cut separation. We only searched for cuts during the branch-and-cut phase of the price-and-branch-and-cut approach and within both RENS approaches. Of course, when evaluating flow rounding algorithms, also the path flow variables have been relaxed. In our experiments, we solved all C50D10 instances using flow rounding, price-and-branch-and-cut and "classical" branch-and-price as explained in Chapter 8. The latter has been used as a baseline for performance comparison. We limited computing times for each instance to at most 4 hours.

Figure 18.5 reports some details on the flow rounding experiments. For this, we concatenated all rounding heuristics (from left to right) and terminated the experiment as soon as any of the algorithms has been able to provide a feasible integral solution. For 10 % of the instances, the root node LP-relaxation already provided an integral flow, hence no rounding has been required there. In contrast,

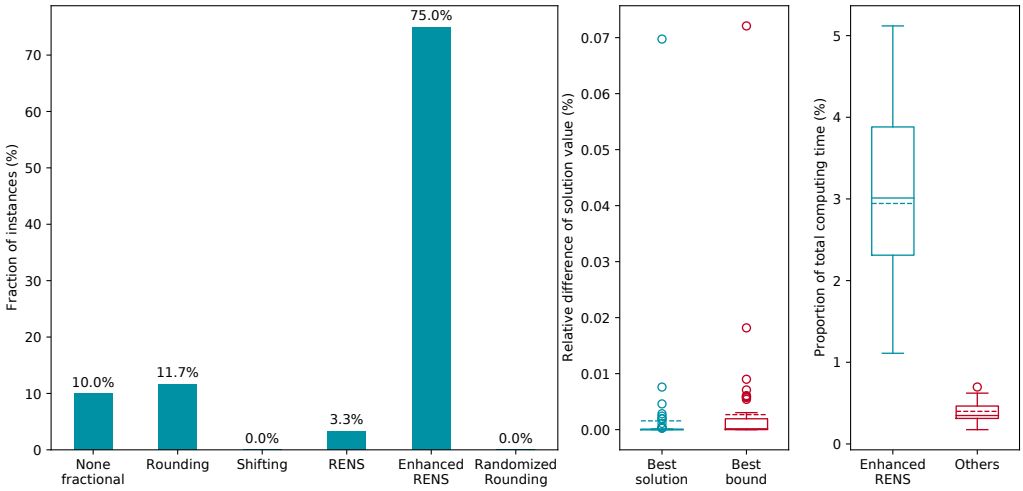


Figure 18.5: Bar charts showing the usage of different rounding methods when solving all C50D10 instances with relaxed flow and ignored activation variables. While the right box plots report the relative computing time of the rounding methods, the left ones evaluate the obtained solution quality when comparing against the best known solutions and bounds for these instances (without activation variables).

15 % of all experiments ended with an integral flow only after using the rounding or RENS algorithm. Shifting did not provide any value in our study. However, for the majority of instance (75 %) only the enhanced RENS heuristic has been able to find an integral flow. The last resort of randomized rounding has not been required during our experiments, since all instances were solved by one of the other heuristics before. Computing times of all flow rounding heuristics except enhanced RENS are negligible. The latter requires up to 5 % of the total computing time, nevertheless, this is acceptable as we will see in the following. Besides computing times, also solution quality is surprisingly good. Both the gap to the best known solution and to the best known lower bound are below 0.005 % for most instances. We will investigate in Section 18.4 if this performance remains when embedding flow rounding into a meta-heuristic that takes care of fixing all arc activation variables.

Comparing the performance of flow rounding, price-and-branch-and-cut and branch-and-price in regard to computing times reveals that there is a minor advantage of flow rounding for the majority of instances (see Figure 18.6). Furthermore, there are a few instances where branch-and-price is not competitive at all. In contrast, price-and-branch-and-cut is less vulnerable to such significant discrepancies. Since

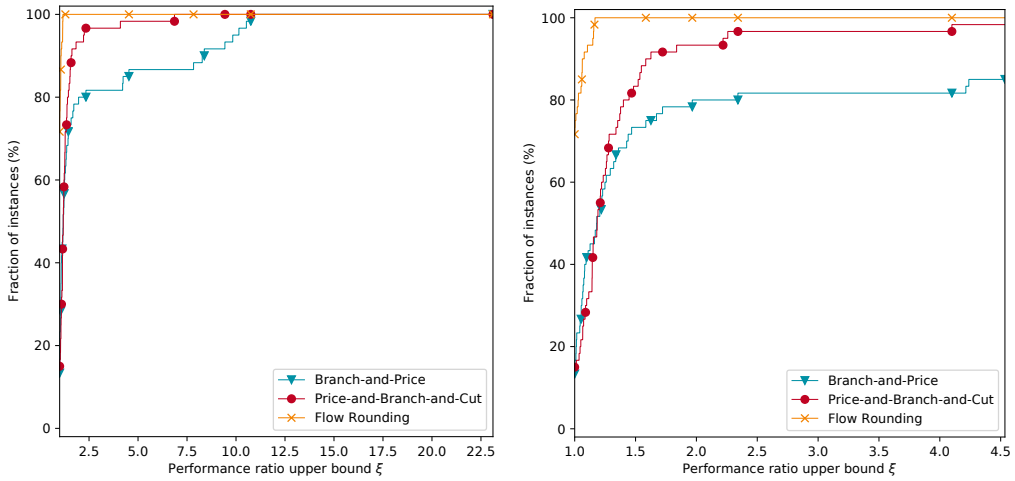


Figure 18.6: Performance profiles reporting computing times for solving all C50D10 instances with ignored activation variables by flow rounding methods compared to branch-and-price and price-and-branch-and-cut. While (a) shows the full profile, (b) restricts the view to the range $\xi \in [1, 5]$

flow rounding clearly outperforms both alternatives, we suggest to use that strategy for calculating an integral from a fractional solution. Nevertheless, if the flow rounding algorithms more often lead to randomized rounded solutions that violate certain constraints, considering price-and-branch-and-cut could be an option, since it would be easy to modify that approach to continue pricing as long as no integral solution has been found.

18.4 Simulated Annealing with Branch-and-Price (SABP)

In the following, we are going to evaluate the performance of our SABP heuristic as developed in Chapter 12. We used 4 cores and 32 GB memory to solve all C50D10 instances. Memory has been split into 75 % JVM heap and 25 % native memory. All experiments have been carried out once using JMH for execution and collection of performance metrics. Iterations of SABP have been limited equally to the original publication of Yaghini, Rahbar, and Karimi (2013). Hence, there will be a maximum of 100 temperature changes and at most 30 temperature changes that did not lead to an improvement of the best known solution. Furthermore, we limited all experiments to 12 hours computing time. For each temperature, there will be four iterations, each closing up to three and opening up to five arcs.

To compute the initial temperature, it is required to determine the maximal expected costs increase between two neighboring solutions. Since these costs cannot be determined exactly upfront, we need to come up with a proper estimate. We assumed that the maximal increase occurs when closing "the wrong" arc. Hence, we inspected the initial solution found by SABP and searched for the arc causing the highest costs, defined as the sum of the fixed-costs and the unit costs for all vehicles using that arc. These are the costs that will be saved when closing the corresponding arc. However, all vehicles that previously used the arc will be forced to travel an alternative path. From our experience in finished vehicle logistics, we assume that there will be alternatives that do not raise overall logistics costs by more than 15 %. This assumption turned out to work properly in preliminary testing. Hence, we used that 15 % of the highest arc costs as initial temperature in all our experiments.

During all experiments SABP reached the time limit of 12 hours, so the iteration limits had no effect in our case. Figure 18.7 shows the typical convergence of SABP

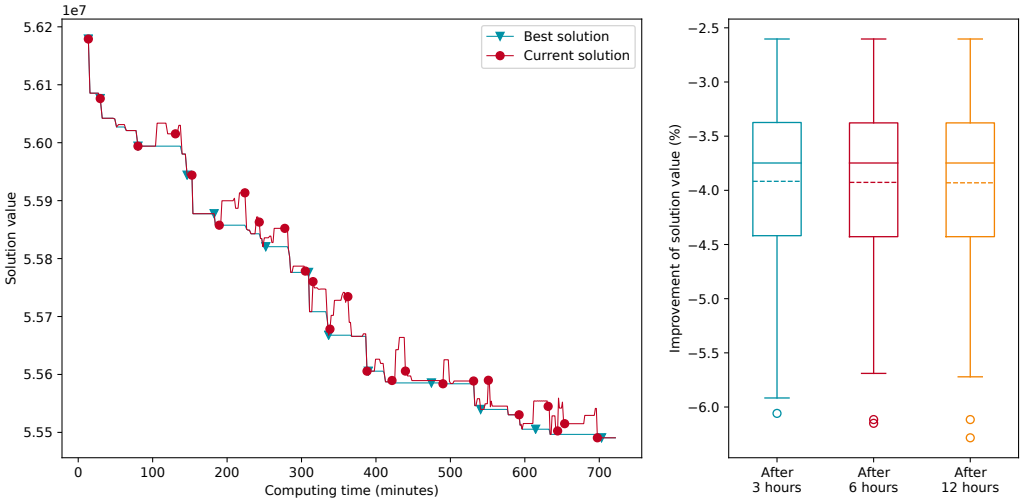


Figure 18.7: Charts showing the convergence of SABP from an initial to its final solution using instance C50D10R-1 as an example. The box plots show the relative improvement of the solution values after 3, 6, and 12 hours of solving.

from its initial to its final solution, using instance C50D10R-1 as example. The plots nicely visualize, how non-improving solutions, which increase the solution value, are accepted occasionally by the algorithm. Furthermore, the box plot reports that during a run of SABP the initial solution value improves between 2.5 % to 6.3 %. Most of this improvement is gained within the first hours of a run which indicates that the algorithm converges quickly against the final solution.

Since this final solution is not necessarily a good one, we compared its quality against the best solution found by branch-and-price-and-cut. For the latter we used the preferred configuration explained in Section 16.7 and set a time limit of 12 hours. Figure 18.8 reports the results from these experiments. Since both solution methods reached the time limit, it is reasonable to compare the values of the best found solutions here. Although branch-and-price-and-cut delivers better solutions for the majority of instances, SABP is still competitive with a maximum increase in the solution value of 1.6 %. Furthermore, for some instances SABP is able to find considerable better solutions with an improvement of the solution value of up to 5.6 %. Most importantly, the performance profiles in Figure 18.8 look nearly identical when considering the solutions that have been found by SABP after 3 hours instead of 12 hours. Hence, if computing time is a limiting factor, e.g., business operations do not allow to wait for 12 hours, it might be a good

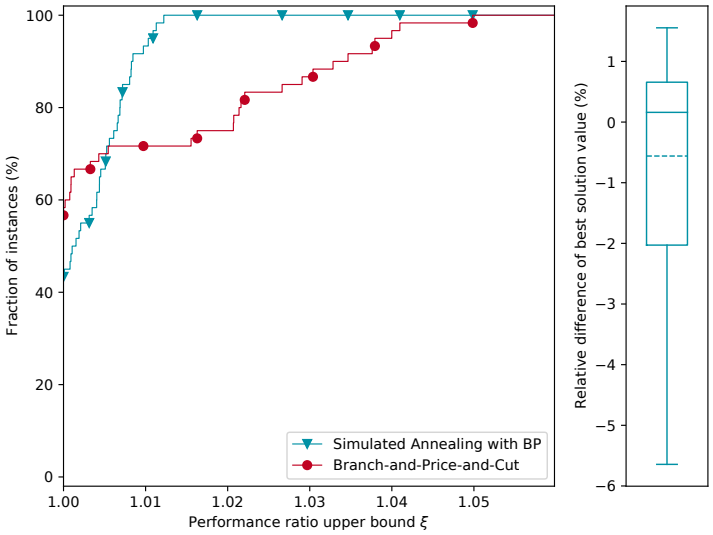


Figure 18.8: Performance profiles of SABP comparing the obtained solution quality to the best solutions found by branch-and-price-and-cut after 12 hours of solving C50D10 instances. The box plot reports the relative difference between the value of these solutions.

option to use SABP to find solutions quickly without significantly compromising quality. Besides that, we suggest to use SABP even if longer solving times are acceptable to find a warm-start solution for branch-and-price-and-cut. It would be beneficial to run SABP for 3 hours to acquire a high quality start solution which could be further improved by branch-and-price-and-cut for the remaining 9 hours. We expect that this strategy combines the best of both methods.

Since our initial intention of implementing a heuristic like SABP has been to solve the largest instances of our benchmark set, we carried out a final experiment and tried to solve all instances of class C500D30. It is no surprise that this experiment revealed similar results to what has been achieved in Section 16.7. As branch-and-price-and-cut had problems to solve the root LP-relaxation of most instances, the same problem occurred when embedding it into the SABP heuristic. We do not report these results explicitly here, since they would not add any additional value.

19 | Conclusion

Using our knowledge about the processes within an automotive outbound supply chain, we have identified the underexamined problem of distribution planning in finished vehicle logistics and were one of the first to propose a mathematical model for that planning task. Based on a comprehensive literature review, we derived a MIP from the arc-based formulation typically used for MCFND and examined its complexity and tractability. Using this MIP, we were able to solve instances of our problem to optimality, including scheduled routes for all finished vehicles of these instances.

Unfortunately, the arc-based formulation turns out to be cumbersome when introducing additional requirements restricting the feasible routes for a vehicle. Furthermore, performing branch-and-cut on this formulation has been successful only for all small and some medium size instances. The small instances have entirely been solved to optimality, however, for the others the separated cuts have been too weak to close the MIP gap efficiently. Our extensive computational study on randomly generated benchmark instances clearly shows this weakness. The underlying benchmark set has been created using a generator developed by us, which is designed to model the specifics and characteristics of a finished vehicle outbound supply chain.

To overcome the deficiencies of the branch-and-cut approach, we developed a path-based formulation of the problem, using the decomposition technique of Dantzig and Wolfe (1960). We successfully solved all medium-sized and some large instances of the resulting MIP by branch-and-price-and-cut. Unfortunately, none of these instances could be solved to optimality and it remains open, whether other approaches that are applied in practice today (like manually planning by human distribution planners) lead to solutions of higher quality. Furthermore, we showed that for small instances, the branch-and-cut approach outlined above performs considerably better and should be preferred over the suggested branch-and-price-and-cut approach.

An important ingredient of each branch-and-price procedure is the algorithm for solving the pricing sub-problem. We showed that the sub-problem is in our case best described by an SPPRC and we evaluated multiple algorithms to solve it efficiently. Finally, we successfully applied different techniques to accelerate these path search algorithms, primarily by using a novel approach depending on the relations between a network and its time-expanded counterpart.

Since our exact solution approaches were unable to solve the largest instances appropriately, we tried different heuristic techniques to speed up the solution process. We showed under which conditions individual vehicles can be aggregated to a single commodity (without losing optimality) and how these conditions can be relaxed to obtain 60 % to 80 % less commodities that must be considered in the MIPs. Having been able to demonstrate that this aggregation approach has only little impact on solution quality, we suggest to consider it in practice. The same applies to the heuristic path search algorithm which we introduced to accelerate pricing. However, this technique should only be applied if computing times spent for pricing are not dominated by the times required to solve the reduced master problem, as heuristically priced columns downgraded the lower bound by up to 0.35 %. Additionally, we revealed that branching on fractional flow within our branch-and-price-and-cut solver should be avoided, since the presented flow rounding techniques were able to produce near-optimal integral solutions using significantly less computing time.

Besides our attempts to heuristically accelerate the exact approaches, we also adopted a heuristic algorithm which has been proven to be successful in solving instances of MCFND. We selected an algorithm combining simulated annealing with column generation from an in-depth literature review and a comparison of (normalized) benchmark results presented by other researchers. Replacing the column generation part of the original method with our branch-and-price-and-cut solver and incorporating additional heuristic techniques like flow rounding and heuristic pricing, led to a solution approach that is able to find near-optimal solutions in remarkable less computing time. Since this approach finds high-quality solutions quickly but struggles to further improve them afterwards, we suggest to use these solutions as warm-start for branch-and-price-and-cut if longer running times are admissible.

By considering problem specific large and challenging instances, we were able to investigate the current limits of our approach. Our algorithms reliably find

solutions on network design problems with up to 100 000 commodities. In our view, that already is quite impressive, especially in comparison to the results reported by other researchers (for example see Chouman, Crainic, and Gendron 2018; Gendron, Hanafi, and Todosijević 2018), and forms a solid basis for future research on distribution planning in finished vehicle logistics (see the outlook in Chapter 20). The instances for which solutions can be found are of high relevance in practice since they correspond to real world scenarios such as the amount of vehicles, which large automobile manufactures like Toyota or VW would typically sell within a one month period within Europe or North America (Toyota Motor Corp. 2021; Volkswagen Group 2021). Hence, we expect that the framework presented in this thesis can be deployed to practice as part of a DSS.

20 | Outlook

While the results presented in this thesis look already promising to be implemented in practice, further improvements and extensions can be made to our approach. In the following, we will summarize some opportunities, that we identified while working on this thesis.

Problem Scope Throughout this thesis, we accepted the problem scope introduced in Chapter 2 as given. However, from a business perspective it might be valuable to widen it to a certain extend. There exist approaches allowing changes of the production sequence (Chen 2004; Jin, Luo, and Ekşioğlu 2008), which might be beneficial, if there are vehicles that urgently need to reach an infrequently available transport like a vessel. Furthermore, for some transports, especially for trains and vessels, it might be favorable to provide a proper mixture of small and large or light and heavy vehicles to simplify loading. Incorporating loading patterns into the problem scope could satisfy such requirements. Finally, since we expect that the data on which our optimization is based will change recurringly, it might be advantageous to explicitly consider robustness requirements, such as uncertain transport times (for an example, see Raack 2012). This could reduce the amount of plan amendments and thus lead to a more resilient supply chain.

Model The most significant downside of our model is the necessity to do a time expansion of the network, which leads to huge MIPs that are hard to solve and require an enormous amount of computer memory. Boland et al. (2017) and Marshall et al. (2021) suggested the use of continuous time networks to resolve this shortcoming. Alternatively, it might be beneficial to dynamically generate the time-expanded network similar to the proposal of Fischer and Helmberg (2012). Furthermore, besides the arc-based and path-based formulations that have been explained in this thesis, there exist other formulations that might reduce the size

of the MIPs even for time-expanded networks. An example is the cycle formulation introduced by Barnhart et al. (1994).

Path Search Although we already spent some effort in accelerating the path search algorithms, there exist techniques like bi-directional or hierarchical search that might improve their performance even further (Thomas, Calogiuri, and Hewitt 2019). Additionally, we did not examine an efficient handling of labels within these algorithms, albeit other researchers proposed techniques which can significantly reduce the amount of dominance checks (Desrochers and Soumis 1988; Sadykov, Uchoa, and Pessoa 2017). Finally, it would be interesting to investigate if the three-stage approach proposed by Zhu and Wilhelm (2013), which is heavily relying on preprocessing to speed up path search, can be applied in our context.

Exact Approaches Since branch-and-price-and-cut emerged as the method of choice for solving large problem instances, future research should focus primarily on that approach instead of branch-and-cut. We wondered, whether there exist more efficient cuts and separation algorithms, to tighten the lower bounds. For example, it could be interesting to see the performance of Zero Half and MIR cuts, which have been most successful for branch-and-cut. Besides cut separation, there are further ingredients of branch-and-price-and-cut, that might be improved. There could be superior approaches for scoring and variable selection during pricing. Furthermore, we examined node selection only superficially. Also the techniques for filtering and variable fixing that have been proposed by Chouman, Crainic, and Gendron (2018), might be worth to be incorporated. Moreover, comparing the solutions found by branch-and-price-and-cut to current real world solutions of corresponding instances, would help to get a better feeling in regard to the quality of solutions.

In addition to improving our branch-and-price-and-cut approach, we see more opportunities in the context of exact methods. In the existing literature, Lagrangian relaxation has been reported to show superior performance for FND. Applying this technique to our problem domain would be a logical next step. Finally, we just followed the suggestion of Chouman, Crainic, and Gendron (2016) concerning the usage of the single-source single-sink pattern for aggregating commodities, although this approach might be worth to be investigated in more detail.

Heuristics Various heuristic variations of exact methods have been applied in this thesis. It might be worth considering other (additional) acceleration techniques, such as heuristically cutting off unpromising branches (Chabrier 2003). Nevertheless, since the major issues we faced were directly related to the size of our problem instances, we expect the largest improvements when investigating advanced decomposition techniques. There exist algorithms for decomposing the time-dimension of our problem (Stadtler 2003; Helber and Sahling 2010; Dehghan Nayeri 2017) or the underlying network (Bärmann 2016). Both methods can significantly reduce the size of the MIPs, at the price of solving multiple smaller problems. Nevertheless, such decomposition might allow the handling of our largest benchmark instances and could offer an opportunity for parallelizing the solving process in a compute cluster.

Part V

Appendix

21 | Detailed Facts of MIPs Created by Branch-and-Cut Solver

Table 21.1: Table showing some facts about used computing resources during experiments and details about the MIPs generated for each benchmark instance when using our branch-and-cut solver. Instances without specified "best MIP gap" have been solved to optimality within a time limit of 12 hours. Memory is listed in Gigabyte (GB).

	Hardware		MIP size		Time	MIP gap (%)	
Instance	**CPU**	**Mem**	**Rows**	**Cols**	**(min)**	**Initial**	**Best**
C1D5R-1	3	12	206 506	256 988	0.1	6.8	-
C1D5R-2	3	12	322 368	393 683	0.1	7.5	-
C1D5R-3	3	12	540 159	658 005	0.2	6.8	-
C1D5R-4	3	12	328 642	399 565	0.1	5.7	-
C1D5R-5	3	12	322 846	401 842	0.2	4.1	-
C1D5RR-1	3	12	530 045	643 920	0.2	6.0	-
C1D5RR-2	3	12	121 149	151 968	0.1	1.5	-
C1D5RR-3	3	12	585 231	711 263	0.2	4.6	-
C1D5RR-4	3	12	305 732	361 015	0.1	16.2	-
C1D5RR-5	3	12	394 277	455 627	0.1	92.4	-
C1D5RL-1	3	12	838 627	1 064 926	0.7	4.1	-
C1D5RL-2	3	12	1 075 248	1 365 158	0.8	1.3	-
C1D5RL-3	3	12	1 081 085	1 330 514	0.3	11.5	-
C1D5RL-4	3	12	692 767	862 266	0.3	14.6	-
C1D5RL-5	3	12	1 454 939	1 819 436	0.6	6.0	-
C1D5RLL-1	3	12	521 944	659 999	0.4	8.7	-
C1D5RLL-2	3	12	1 120 412	1 407 197	0.7	6.4	-
C1D5RLL-3	3	12	648 502	747 219	0.2	67.5	-
C1D5RLL-4	3	12	789 411	999 780	0.6	14.5	-
C1D5RLL-5	3	12	1 249 588	1 565 090	0.4	6.8	-
C1D5RRL-1	3	12	733 708	916 571	0.2	6.3	-
C1D5RRL-2	3	12	508 705	629 225	0.3	6.5	-
C1D5RRL-3	3	12	340 976	427 020	0.1	8.8	-

Instance	CPU	Mem	Rows	Cols	Time	Initial	Best
C1D5RRL-4	3	12	1 237 406	1 566 951	1.8	13.9	-
C1D5RRL-5	3	12	567 509	704 005	0.3	7.8	-
C1D5RRLL-1	3	12	1 100 574	1 337 798	0.7	34.3	-
C1D5RRLL-2	3	12	399 119	495 680	0.1	1.9	-
C1D5RRLL-3	3	12	771 086	973 366	0.3	1.6	-
C1D5RRLL-4	3	12	494 487	622 661	0.3	21.5	-
C1D5RRLL-5	3	12	747 501	937 728	0.4	12.0	-
C1D5T-1	3	12	150 646	185 277	0.0	6.3	-
C1D5T-2	3	12	151 631	188 948	0.1	3.1	-
C1D5T-3	3	12	222 963	271 777	0.1	8.2	-
C1D5T-4	3	12	203 554	253 476	0.1	28.7	-
C1D5T-5	3	12	214 496	263 389	0.1	15.1	-
C1D5TT-1	3	12	320 135	398 112	0.1	18.0	-
C1D5TT-2	3	12	276 033	339 286	0.1	11.1	-
C1D5TT-3	3	12	225 309	278 569	0.1	8.7	-
C1D5TT-4	3	12	166 822	207 763	0.0	0.9	-
C1D5TT-5	3	12	306 655	377 135	0.1	3.8	-
C1D5TL-1	3	12	936 710	1 166 027	0.5	6.1	-
C1D5TL-2	3	12	434 641	542 866	0.2	12.0	-
C1D5TL-3	3	12	1 487 860	1 849 509	0.6	5.9	-
C1D5TL-4	3	12	1 098 415	1 365 194	0.5	11.1	-
C1D5TL-5	3	12	784 986	979 832	0.2	13.5	-
C1D5TLL-1	3	12	742 472	936 453	0.6	7.5	-
C1D5TLL-2	3	12	642 117	799 221	0.2	4.0	-
C1D5TLL-3	3	12	669 233	839 411	0.2	5.3	-
C1D5TLL-4	3	12	1 057 243	1 320 889	0.4	7.9	-
C1D5TLL-5	3	12	828 487	1 027 747	0.3	13.3	-
C1D5TTL-1	3	12	494 231	613 665	0.3	10.3	-
C1D5TTL-2	3	12	615 165	773 220	0.2	8.5	-
C1D5TTL-3	3	12	588 058	740 578	0.3	2.5	-
C1D5TTL-4	3	12	549 056	687 722	0.1	5.4	-
C1D5TTL-5	3	12	736 977	920 956	0.2	10.9	-
C1D5TTLL-1	3	12	588 148	736 252	0.3	11.5	-
C1D5TTLL-2	3	12	1 076 541	1 352 537	0.4	12.4	-
C1D5TTLL-3	3	12	560 470	699 786	0.2	6.3	-
C1D5TTLL-4	3	12	1 245 896	1 564 543	0.4	5.1	-
C1D5TTLL-5	3	12	775 041	965 378	0.2	5.7	-
C5D5R-1	3	12	1 274 597	1 587 774	1.2	5.3	-
C5D5R-2	3	12	568 104	729 405	4.0	1.6	-

Instance	CPU	Mem	Rows	Cols	Time	Initial	Best
C5D5R-3	3	12	2 023 684	2 477 167	4.8	14.8	-
C5D5R-4	3	12	1 088 156	1 353 547	1.4	5.3	-
C5D5R-5	3	12	1 535 503	1 923 938	6.2	6.9	-
C5D5RR-1	3	12	1 490 473	1 853 391	2.5	6.8	-
C5D5RR-2	3	12	999 808	1 260 468	1.1	28.3	-
C5D5RR-3	3	12	1 235 330	1 535 205	101.3	8.5	-
C5D5RR-4	3	12	1 341 144	1 670 058	0.8	5.8	-
C5D5RR-5	3	12	584 899	705 095	1.9	152.3	-
C5D5RL-1	3	12	2 632 094	3 226 723	0.7	8.4	-
C5D5RL-2	3	18	5 981 743	7 540 988	10.2	7.0	-
C5D5RL-3	3	18	6 172 553	7 805 923	11.2	6.9	-
C5D5RL-4	3	18	4 633 514	5 871 025	184.6	5.7	-
C5D5RL-5	3	18	4 890 907	6 130 774	9.0	8.7	-
C5D5RLL-1	3	18	6 038 171	7 535 747	3.1	15.1	-
C5D5RLL-2	3	12	3 498 800	4 379 510	15.9	5.9	-
C5D5RLL-3	3	12	3 623 015	4 576 034	13.8	5.6	-
C5D5RLL-4	3	12	3 484 158	4 379 670	3.9	8.0	-
C5D5RLL-5	3	12	3 424 666	4 340 712	26.0	4.4	-
C5D5RRL-1	3	18	4 780 437	6 058 994	50.3	5.2	-
C5D5RRL-2	3	18	4 611 089	5 842 676	*Limit*	9.1	0.242
C5D5RRL-3	3	18	5 868 895	7 387 017	22.6	10.2	-
C5D5RRL-4	3	12	3 984 138	5 103 666	151.9	7.5	-
C5D5RRL-5	3	18	4 822 565	6 071 085	18.9	7.0	-
C5D5RRLL-1	3	18	6 176 162	7 839 842	28.8	7.1	-
C5D5RRLL-2	3	12	3 793 955	4 759 205	3.9	7.6	-
C5D5RRLL-3	3	12	3 903 906	4 921 125	20.0	17.9	-
C5D5RRLL-4	3	18	5 888 842	7 070 198	8.2	33.3	-
C5D5RRLL-5	3	18	5 136 869	6 297 361	29.3	68.4	-
C5D5T-1	3	12	1 313 245	1 627 842	0.6	5.0	-
C5D5T-2	3	12	710 064	903 208	1.6	5.0	-
C5D5T-3	3	12	977 893	1 168 026	0.2	15.9	-
C5D5T-4	3	12	1 174 592	1 465 676	0.5	5.6	-
C5D5T-5	3	12	1 582 638	1 964 334	1.6	5.8	-
C5D5TT-1	3	12	790 000	983 224	0.5	7.9	-
C5D5TT-2	3	12	1 388 358	1 630 474	0.7	15.9	-
C5D5TT-3	3	12	1 514 197	1 889 405	1.4	7.0	-
C5D5TT-4	3	12	685 890	810 757	0.2	0.007	-
C5D5TT-5	3	12	1 837 668	2 231 313	3.7	12.7	-
C5D5TL-1	3	12	2 948 952	3 751 184	161.6	4.8	-
C5D5TL-2	3	12	2 711 356	3 463 498	28.9	6.3	-

Instance	CPU	Mem	Rows	Cols	Time	Initial	Best
C5D5TL-3	3	18	4 822 388	6 114 322	5.9	6.3	-
C5D5TL-4	3	12	3 339 955	4 213 356	49.6	5.8	-
C5D5TL-5	3	24	8 128 540	10 217 801	50.6	8.7	-
C5D5TLL-1	3	24	6 216 850	7 795 457	151.8	11.9	-
C5D5TLL-2	3	18	4 883 877	6 096 114	70.8	15.0	-
C5D5TLL-3	3	18	6 546 012	8 276 878	8.5	5.1	-
C5D5TLL-4	3	18	6 976 121	8 554 751	15.3	50.6	-
C5D5TLL-5	3	24	8 074 276	9 944 321	8.4	12.8	-
C5D5TTL-1	3	24	7 514 152	9 333 143	22.5	6.3	-
C5D5TTL-2	3	24	7 707 294	9 642 836	8.0	8.1	-
C5D5TTL-3	3	18	6 386 233	8 012 265	6.9	10.0	-
C5D5TTL-4	3	18	5 602 237	7 049 443	6.7	8.2	-
C5D5TTL-5	3	12	3 696 004	4 626 712	3.6	7.3	-
C5D5TTLL-1	3	18	5 433 975	6 799 243	13.1	5.1	-
C5D5TTLL-2	3	18	4 353 715	5 538 204	178.8	4.6	-
C5D5TTLL-3	3	12	3 239 906	4 134 457	71.6	3.7	-
C5D5TTLL-4	3	12	3 319 462	4 206 964	5.2	4.6	-
C5D5TTLL-5	3	24	8 402 241	10 549 165	16.0	7.1	-
C10D10R-1	6	24	2 341 718	2 955 768	*Limit*	14.1	0.172
C10D10R-2	6	24	1 454 550	1 865 633	46.2	3.4	-
C10D10R-3	6	24	2 266 536	2 866 488	628.9	1.0	-
C10D10R-4	6	24	2 651 844	3 344 654	*Limit*	7.9	0.401
C10D10R-5	6	24	2 372 152	2 999 953	66.4	7.1	-
C10D10RR-1	6	24	2 530 948	3 007 351	*Limit*	50.2	0.811
C10D10RR-2	6	24	2 365 597	2 969 748	*Limit*	4.8	0.213
C10D10RR-3	6	24	2 470 993	3 119 679	171.4	6.0	-
C10D10RR-4	6	24	1 882 709	2 380 698	*Limit*	9.0	0.528
C10D10RR-5	6	24	1 945 631	2 466 934	*Limit*	10.4	0.407
C10D10RL-1	6	48	19 967 097	25 272 338	*Out of memory*		
C10D10RL-2	6	36	11 217 435	14 249 284	*Limit*	6.9	0.79
C10D10RL-3	6	36	8 855 678	11 130 152	*Limit*	12.7	0.969
C10D10RL-4	6	24	7 992 375	10 271 338	205.4	3.1	-
C10D10RL-5	6	24	6 422 050	8 175 139	145.9	4.8	-
C10D10RLL-1	6	48	11 633 964	14 609 639	*Limit*	15.1	0.18
C10D10RLL-2	6	48	13 938 272	17 913 591	*Limit*	5.6	0.105
C10D10RLL-3	6	48	12 138 946	15 308 981	*Limit*	8.4	0.439
C10D10RLL-4	6	24	7 204 493	8 989 553	*Limit*	52.3	0.143
C10D10RLL-5	6	24	3 832 997	4 933 166	83.2	6.1	-
C10D10RRL-1	6	48	15 032 300	19 024 812	*Limit*	8.0	0.332

Instance	CPU	Mem	Rows	Cols	Time	Initial	Best
C10D10RRL-2	6	36	9 744 038	12 425 580	*Limit*	6.6	0.36
C10D10RRL-3	6	48	16 786 621	20 636 635	*Out of memory*		
C10D10RRL-4	6	36	12 122 719	14 983 885	*Limit*	8.3	0.279
C10D10RRL-5	6	24	6 572 368	8 239 170	*Limit*	6.4	0.151
C10D10RRLL-1	6	48	12 423 883	15 097 653	*Limit*	19.6	0.587
C10D10RRLL-2	6	24	7 223 439	9 212 125	*Limit*	6.6	0.196
C10D10RRLL-3	6	24	6 575 361	8 344 729	*Limit*	2.9	0.095
C10D10RRLL-4	6	24	8 132 475	10 342 816	*Limit*	4.8	0.289
C10D10RRLL-5	6	48	12 232 332	15 478 355	*Limit*	8.7	0.309
C10D10T-1	6	24	2 156 094	2 740 646	*Limit*	1.7	0.065
C10D10T-2	6	24	2 105 765	2 656 858	*Limit*	10.7	0.133
C10D10T-3	6	24	3 535 910	4 431 998	*Limit*	12.7	0.957
C10D10T-4	6	24	2 269 269	2 787 702	*Limit*	15.3	0.703
C10D10T-5	6	24	4 019 941	5 000 529	*Limit*	8.2	0.749
C10D10TT-1	6	24	3 582 639	4 360 525	*Limit*	29.7	1.145
C10D10TT-2	6	24	2 003 023	2 537 266	*Limit*	5.0	0.089
C10D10TT-3	6	24	2 970 167	3 688 666	*Limit*	11.3	0.44
C10D10TT-4	6	24	2 814 035	3 563 635	*Limit*	5.1	0.108
C10D10TT-5	6	24	2 853 148	3 578 375	*Limit*	7.8	0.131
C10D10TL-1	6	48	12 880 213	16 345 932	*Limit*	6.7	0.185
C10D10TL-2	6	36	7 398 636	9 330 863	*Limit*	14.6	0.544
C10D10TL-3	6	48	16 514 257	20 570 156	*Out of memory*		
C10D10TL-4	6	24	3 332 357	4 315 431	570.0	2.5	-
C10D10TL-5	6	24	4 821 821	6 134 585	*Limit*	26.5	0.715
C10D10TLL-1	6	36	8 561 377	10 532 992	*Limit*	13.0	1.063
C10D10TLL-2	6	48	11 205 280	14 060 935	*Limit*	11.8	1.786
C10D10TLL-3	6	48	17 836 800	22 549 149	*Out of memory*		
C10D10TLL-4	6	36	9 093 480	11 513 274	*Limit*	6.2	0.072
C10D10TLL-5	6	36	9 053 995	11 711 811	*Limit*	5.7	0.209
C10D10TTL-1	6	24	7 337 348	9 016 389	2.6	11.0	-
C10D10TTL-2	6	48	13 983 922	17 363 971	164.5	12.0	-
C10D10TTL-3	6	36	11 802 644	15 019 987	*Limit*	5.1	0.219
C10D10TTL-4	6	48	13 391 756	17 099 436	*Limit*	2.9	0.148
C10D10TTL-5	6	36	9 402 672	11 903 441	88.4	6.1	-
C10D10TTLL-1	6	48	13 702 264	17 274 663	73.1	10.6	-
C10D10TTLL-2	6	48	11 217 337	14 215 687	*Limit*	4.5	0.154
C10D10TTLL-3	6	48	12 459 651	15 525 592	563.4	11.1	-
C10D10TTLL-4	6	36	6 306 103	7 890 794	*Limit*	8.9	0.198
C10D10TTLL-5	6	36	9 551 362	12 091 893	49.9	5.1	-

22 | Detailed Facts of MIPs Created by Branch-and-Price-and-Cut Solver

Table 22.1: Table showing some facts about used computing resources during experiments and details about the MIPs generated for each benchmark instance when using our branch-and-price-and-cut solver. Memory is listed in Gigabyte (GB). Best gaps have been calculated within a time limit of 12 hours.

	Hardware		Initial MIP size		Time	MIP gap (%)	
Instance	**CPU**	**Mem**	**Rows**	**Cols**	**(min)**	**Initial**	**Best**
C1D5R-1	3	12	852	737	3.1	5.509	-
C1D5R-2	3	12	1025	732	7.3	6.217	-
C1D5R-3	3	12	1309	964	2.6	6.294	-
C1D5R-4	3	12	1034	659	2.7	5.185	-
C1D5R-5	3	12	974	884	4.2	3.215	-
C1D5RR-1	3	12	997	822	3.0	5.74	-
C1D5RR-2	3	12	535	542	0.4	1.336	-
C1D5RR-3	3	12	904	678	2.4	4.244	-
C1D5RR-4	3	12	913	593	1.1	14.702	-
C1D5RR-5	3	12	743	584	0.8	69.128	-
C1D5RL-1	3	12	983	1635	12.3	3.399	-
C1D5RL-2	3	12	1303	1555	3.0	1.041	-
C1D5RL-3	3	12	1176	1166	4.3	9.216	-
C1D5RL-4	3	12	1035	1090	5.4	12.565	-
C1D5RL-5	3	12	1389	1175	2.2	5.158	-
C1D5RLL-1	3	12	1009	1046	2.1	6.717	-
C1D5RLL-2	3	12	916	1364	4.8	4.094	-
C1D5RLL-3	3	12	1121	785	1.6	52.296	-
C1D5RLL-4	3	12	1012	1455	9.1	10.323	-
C1D5RLL-5	3	12	1181	1425	3.5	5.224	-
C1D5RRL-1	3	12	1118	1297	2.4	5.132	-
C1D5RRL-2	3	12	1328	862	6.1	5.199	-
C1D5RRL-3	3	12	694	619	2.3	6.833	-

Instance	CPU	Mem	Rows	Cols	Time	Initial	Best
C1D5RRL-4	3	12	1130	1578	29.5	10.404	-
C1D5RRL-5	3	12	1060	802	4.0	6.439	-
C1D5RRLL-1	3	12	1637	1623	4.5	28.311	-
C1D5RRLL-2	3	12	747	668	1.0	1.692	-
C1D5RRLL-3	3	12	1161	1480	7.3	1.112	-
C1D5RRLL-4	3	12	842	895	5.4	17.916	-
C1D5RRLL-5	3	12	1198	1096	6.8	10.012	-
C1D5T-1	3	12	486	380	0.8	5.313	-
C1D5T-2	3	12	531	419	3.3	2.727	-
C1D5T-3	3	12	840	605	2.1	7.657	-
C1D5T-4	3	12	787	562	0.8	26.314	-
C1D5T-5	3	12	957	702	5.8	12.429	-
C1D5TT-1	3	12	774	627	31.2	15.369	-
C1D5TT-2	3	12	1110	702	1.0	8.173	-
C1D5TT-3	3	12	887	610	3.1	7.303	-
C1D5TT-4	3	12	491	430	1.1	0.772	-
C1D5TT-5	3	12	774	550	7.5	3.162	-
C1D5TL-1	3	12	1684	1439	57.5	5.575	-
C1D5TL-2	3	12	728	650	5.5	8.359	-
C1D5TL-3	3	12	1818	1586	21.0	4.55	-
C1D5TL-4	3	12	1470	1171	6.3	8.879	-
C1D5TL-5	3	12	1334	1180	3.9	11.887	-
C1D5TLL-1	3	12	1325	1282	8.1	6.287	-
C1D5TLL-2	3	12	615	647	0.7	3.377	-
C1D5TLL-3	3	12	1158	1211	2.8	4.311	-
C1D5TLL-4	3	12	1244	1445	2.9	7.193	-
C1D5TLL-5	3	12	1245	1393	5.2	11.996	-
C1D5TTL-1	3	12	804	930	15.8	8.101	-
C1D5TTL-2	3	12	1121	1053	2.9	6.285	-
C1D5TTL-3	3	12	1114	1161	5.8	2.144	-
C1D5TTL-4	3	12	608	782	2.0	4.672	-
C1D5TTL-5	3	12	1323	1265	1.5	9.056	-
C1D5TTLL-1	3	12	1316	1064	23.5	9.302	-
C1D5TTLL-2	3	12	1262	1614	7.8	7.638	-
C1D5TTLL-3	3	12	991	981	6.2	5.326	-
C1D5TTLL-4	3	12	1048	1469	1.1	4.567	-
C1D5TTLL-5	3	12	1209	1065	6.3	4.753	-
C5D5R-1	4	16	2441	2563	*Limit*	4.464	0.03
C5D5R-2	4	16	1439	1761	*Limit*	1.471	0.052

Instance	CPU	Mem	Rows	Cols	Time	Initial	Best
C5D5R-3	3	12	2078	2023	91.4	12.369	-
C5D5R-4	3	12	2004	2410	*Limit*	4.667	0.207
C5D5R-5	3	12	2604	2150	*Limit*	5.705	0.182
C5D5RR-1	3	12	3364	2450	418.9	6.059	-
C5D5RR-2	3	12	2600	2520	77.7	24.464	-
C5D5RR-3	4	16	2555	2618	*Limit*	7.552	0.438
C5D5RR-4	3	12	2856	2635	117.8	5.316	-
C5D5RR-5	3	12	1899	2235	25.3	129.527	-
C5D5RL-1	3	12	3686	4995	32.4	6.414	-
C5D5RL-2	4	16	5229	6747	*Limit*	5.524	0.054
C5D5RL-3	4	16	4926	6462	*Limit*	5.353	0.084
C5D5RL-4	4	16	4874	6698	*Limit*	4.114	0.166
C5D5RL-5	3	12	4418	6082	39.3	6.931	-
C5D5RLL-1	3	12	3866	5664	*Limit*	12.307	0.039
C5D5RLL-2	4	16	3803	4386	*Limit*	5.173	0.11
C5D5RLL-3	4	16	3192	4255	*Limit*	4.725	0.366
C5D5RLL-4	3	12	3353	4575	178.5	6.268	-
C5D5RLL-5	3	12	4439	5300	*Limit*	3.687	0.145
C5D5RRL-1	3	12	4163	6799	*Limit*	3.898	0.485
C5D5RRL-2	4	16	4573	6149	*Limit*	7.503	0.421
C5D5RRL-3	4	16	4839	6129	*Limit*	8.229	0.185
C5D5RRL-4	4	16	3653	5605	*Limit*	5.698	0.226
C5D5RRL-5	4	16	4578	6093	*Limit*	5.631	0.142
C5D5RRLL-1	4	16	5252	6916	*Limit*	5.296	0.112
C5D5RRLL-2	3	12	3683	4405	*Limit*	6.094	0.218
C5D5RRLL-3	4	16	4637	5176	*Limit*	15.252	0.443
C5D5RRLL-4	3	12	6393	7474	204.1	28.366	-
C5D5RRLL-5	3	12	3721	4456	*Limit*	64.329	0.267
C5D5T-1	3	12	2640	2358	332.0	4.379	-
C5D5T-2	3	12	1706	2071	*Limit*	4.511	0.161
C5D5T-3	3	12	3353	2967	2.6	3.831	-
C5D5T-4	3	12	2459	2842	16.3	4.833	-
C5D5T-5	3	12	2740	2536	*Limit*	4.924	0.183
C5D5TT-1	3	12	2055	2133	35.4	6.626	-
C5D5TT-2	3	12	2478	2522	241.8	15.121	-
C5D5TT-3	3	12	2259	2110	60.0	5.741	-
C5D5TT-4	3	12	1233	1523	0.1	0.012	-
C5D5TT-5	3	12	2829	2674	51.9	9.769	-
C5D5TL-1	3	12	3321	4772	*Limit*	4.563	1.024
C5D5TL-2	4	16	2637	3785	*Limit*	4.501	0.095

Instance	CPU	Mem	Rows	Cols	Time	Initial	Best
C5D5TL-3	3	12	4885	5557	*Limit*	4.639	0.083
C5D5TL-4	4	16	3956	4971	*Limit*	4.7	0.183
C5D5TL-5	4	16	5321	7655	*Limit*	6.601	0.02
C5D5TLL-1	4	16	5419	5981	*Limit*	9.145	0.213
C5D5TLL-2	4	16	5071	6563	*Limit*	11.707	0.306
C5D5TLL-3	4	16	4850	7089	*Limit*	3.987	0.414
C5D5TLL-4	3	12	5633	7999	*Limit*	45.52	1.625
C5D5TLL-5	3	12	4217	6672	76.0	8.156	-
C5D5TTL-1	3	12	4993	6083	*Limit*	4.089	0.453
C5D5TTL-2	3	12	5339	6110	218.4	6.293	-
C5D5TTL-3	4	16	5288	6983	*Limit*	7.036	0.06
C5D5TTL-4	3	12	4448	5850	669.4	6.232	-
C5D5TTL-5	3	12	4739	4290	*Limit*	6.379	0.106
C5D5TTLL-1	3	12	4126	4182	289.4	3.505	-
C5D5TTLL-2	3	12	4312	6660	*Limit*	3.936	0.536
C5D5TTLL-3	4	16	3433	5321	*Limit*	2.939	0.39
C5D5TTLL-4	3	12	3060	4006	658.3	3.619	-
C5D5TTLL-5	4	16	5284	6576	*Limit*	5.2	0.184
C10D10R-1	4	32	4349	5331	*Limit*	11.15	1.179
C10D10R-2	4	16	2851	4490	*Limit*	3.197	0.324
C10D10R-3	4	16	5405	6317	*Limit*	0.931	0.255
C10D10R-4	4	16	3489	4571	*Limit*	6.658	0.632
C10D10R-5	4	16	4218	5057	*Limit*	6.228	0.557
C10D10RR-1	4	32	6030	5727	*Limit*	45.767	1.805
C10D10RR-2	4	16	4200	5158	*Limit*	4.274	0.414
C10D10RR-3	4	16	5200	4895	*Limit*	5.725	0.402
C10D10RR-4	4	16	3932	5175	*Limit*	8.091	0.746
C10D10RR-5	4	16	4119	4793	*Limit*	9.044	0.499
C10D10RL-1	4	16	8372	12 090	*Limit*	7.167	0.74
C10D10RL-2	4	32	9421	13 113	*Limit*	5.359	0.537
C10D10RL-3	4	32	8700	10 887	*Limit*	9.728	0.629
C10D10RL-4	4	16	7223	11 654	*Limit*	2.552	0.138
C10D10RL-5	4	16	6169	8250	*Limit*	4.39	0.282
C10D10RLL-1	4	16	9244	14 507	*Limit*	13.551	0.345
C10D10RLL-2	4	16	8487	13 858	*Limit*	4.54	0.215
C10D10RLL-3	4	32	9183	10 987	*Limit*	7.092	0.556
C10D10RLL-4	4	16	6810	10 361	*Limit*	47.567	1.515
C10D10RLL-5	4	16	5020	6852	*Limit*	5.598	0.534
C10D10RRL-1	4	16	10 202	14 378	*Limit*	6.679	0.442

Instance	CPU	Mem	Rows	Cols	Time	Initial	Best
C10D10RRL-2	4	32	8660	12 177	*Limit*	5.343	0.521
C10D10RRL-3	4	32	8530	13 748	*Limit*	31.667	1.153
C10D10RRL-4	4	32	11 978	13 194	*Limit*	6.96	0.42
C10D10RRL-5	4	32	6590	7746	*Limit*	5.184	0.486
C10D10RRLL-1	4	16	9272	11 090	*Limit*	15.093	0.999
C10D10RRLL-2	4	16	7566	10 598	*Limit*	5.404	0.591
C10D10RRLL-3	4	32	6382	8615	*Limit*	2.573	0.391
C10D10RRLL-4	4	16	7482	10 364	*Limit*	4.097	0.507
C10D10RRLL-5	4	16	6649	9574	*Limit*	6.795	0.426
C10D10T-1	4	16	4491	5848	*Limit*	1.655	0.677
C10D10T-2	4	16	3228	4659	*Limit*	9.78	1.445
C10D10T-3	4	16	4623	4652	*Limit*	10.25	1.131
C10D10T-4	4	16	4968	6025	*Limit*	12.89	1.834
C10D10T-5	4	16	5561	5875	*Limit*	6.444	0.992
C10D10TT-1	4	16	6545	7041	*Limit*	26.876	5.251
C10D10TT-2	4	16	2806	3509	*Limit*	4.271	0.757
C10D10TT-3	4	16	5785	6563	*Limit*	9.611	1.028
C10D10TT-4	4	16	6972	7048	*Limit*	4.62	0.458
C10D10TT-5	4	16	4199	5288	*Limit*	7.2	0.824
C10D10TL-1	4	16	11 792	14 556	*Limit*	5.106	0.359
C10D10TL-2	4	16	7555	9195	*Limit*	12.202	0.744
C10D10TL-3	4	16	7977	11 743	*Limit*	6.371	1.798
C10D10TL-4	4	16	3758	6054	*Limit*	2.301	0.313
C10D10TL-5	4	16	6288	8754	*Limit*	24.315	0.976
C10D10TLL-1	4	32	7748	10 301	*Limit*	10.303	1.13
C10D10TLL-2	4	32	8012	10 162	*Limit*	8.417	2.144
C10D10TLL-3	4	16	10 276	15 570	*Limit*	9.887	0.565
C10D10TLL-4	4	16	8569	12 583	*Limit*	4.832	0.793
C10D10TLL-5	4	16	7555	13 281	*Limit*	4.762	0.465
C10D10TTL-1	4	16	10 288	16 172	6.7	1.378	-
C10D10TTL-2	4	16	7941	13 078	*Limit*	6.849	0.246
C10D10TTL-3	4	16	11 672	16 444	*Limit*	4.599	2.177
C10D10TTL-4	4	16	9348	15 673	*Limit*	2.68	1.17
C10D10TTL-5	4	16	9002	13 762	*Limit*	5.015	1.876
C10D10TTLL-1	4	16	6628	10 330	*Limit*	7.361	0.99
C10D10TTLL-2	4	16	8655	12 742	*Limit*	3.621	0.703
C10D10TTLL-3	4	32	7600	9876	*Limit*	6.779	1.39
C10D10TTLL-4	4	16	6504	10 261	*Limit*	7.505	0.249
C10D10TTLL-5	4	16	8209	10 799	*Limit*	4.03	0.259

Instance	CPU	Mem	Rows	Cols	Time	Initial	Best
C50D10R-1	4	16	24 941	27 863	*Limit*	11.089	4.557
C50D10R-2	4	16	47 716	47 278	*Limit*	12.337	11.364
C50D10R-3	4	16	31 422	35 182	*Limit*	9.957	4.325
C50D10R-4	4	16	42 160	40 799	*Limit*	10.644	9.922
C50D10R-5	4	16	48 634	47 254	*Limit*	10.392	5.128
C50D10RR-1	4	16	45 565	45 852	*Limit*	11.666	11.666
C50D10RR-2	4	16	36 962	38 162	*Limit*	12.577	12.577
C50D10RR-3	4	16	24 736	28 318	*Limit*	8.165	6.255
C50D10RR-4	4	16	33 316	33 529	*Limit*	12.315	5.957
C50D10RR-5	4	16	29 178	31 908	*Limit*	9.963	4.747
C50D10RL-1	4	16	40 901	51 808	*Limit*	8.584	6.801
C50D10RL-2	8	32	47 142	60 213	*Limit*	9.757	5.502
C50D10RL-3	8	32	61 741	75 332	*Limit*	9.646	5.618
C50D10RL-4	8	32	47 520	71 925	*Limit*	15.233	7.787
C50D10RL-5	4	16	46 129	65 606	*Limit*	6.086	3.338
C50D10RLL-1	4	16	51 909	61 512	*Limit*	12.057	5.752
C50D10RLL-2	8	32	61 631	78 334	*Limit*	13.436	8.184
C50D10RLL-3	8	32	48 767	67 707	*Limit*	8.687	4.326
C50D10RLL-4	4	16	53 613	61 353	*Limit*	9.821	7.718
C50D10RLL-5	4	16	40 680	49 113	*Limit*	8.421	4.234
C50D10RRL-1	4	16	46 763	58 139	*Limit*	8.619	4.231
C50D10RRL-2	4	16	55 099	67 477	*Limit*	11.604	6.093
C50D10RRL-3	4	16	43 035	51 315	*Limit*	11.301	6.127
C50D10RRL-4	4	16	52 013	66 571	*Limit*	11.632	5.875
C50D10RRL-5	4	16	55 154	68 325	*Limit*	13.618	9.877
C50D10RRLL-1	4	16	49 898	58 020	*Limit*	11.188	5.809
C50D10RRLL-2	4	16	52 177	66 622	*Limit*	12.908	6.083
C50D10RRLL-3	8	32	50 093	80 669	*Limit*	14.267	7.632
C50D10RRLL-4	4	16	54 485	63 638	*Limit*	8.365	3.828
C50D10RRLL-5	8	32	57 871	64 488	*Limit*	11.101	6.575
C50D10T-1	4	16	37 090	36 362	*Limit*	10.506	5.385
C50D10T-2	4	16	26 718	28 193	*Limit*	9.03	9.03
C50D10T-3	4	16	34 538	39 613	*Limit*	7.788	7.788
C50D10T-4	4	16	31 607	38 608	*Limit*	9.171	9.171
C50D10T-5	4	16	47 255	46 460	*Limit*	9.085	9.085
C50D10TT-1	4	16	24 445	28 565	*Limit*	10.425	4.16
C50D10TT-2	4	16	31 083	35 127	*Limit*	8.672	8.123
C50D10TT-3	4	16	44 589	46 766	*Limit*	9.894	9.398
C50D10TT-4	4	16	22 260	32 249	*Limit*	7.688	6.43

Instance	CPU	Mem	Rows	Cols	Time	Initial	Best
C50D10TT-5	4	16	32 019	31 995	*Limit*	8.509	6.823
C50D10TL-1	8	32	57 996	69 557	*Limit*	10.948	8.552
C50D10TL-2	8	32	56 718	74 721	*Limit*	11.172	6.88
C50D10TL-3	8	32	31 721	44 626	*Limit*	5.636	2.359
C50D10TL-4	8	32	43 124	54 776	*Limit*	10.812	6.678
C50D10TL-5	8	32	58 243	79 995	*Limit*	8.65	4.926
C50D10TLL-1	4	16	37 026	43 598	*Limit*	8.351	3.454
C50D10TLL-2	8	32	56 048	70 152	*Limit*	8.87	4.745
C50D10TLL-3	8	32	52 055	59 064	*Limit*	10.23	5.813
C50D10TLL-4	8	32	44 674	59 131	*Limit*	8.022	3.922
C50D10TLL-5	8	32	44 059	63 442	*Limit*	9.35	5.375
C50D10TTL-1	8	32	51 125	66 473	*Limit*	11.444	6.131
C50D10TTL-2	8	32	53 415	62 742	*Limit*	9.248	8.242
C50D10TTL-3	8	32	48 982	55 263	*Limit*	8.858	5.007
C50D10TTL-4	8	32	47 635	59 515	*Limit*	8.841	4.607
C50D10TTL-5	8	32	50 174	80 878	*Limit*	9.629	5.613
C50D10TTLL-1	8	32	56 129	68 884	*Limit*	10.158	6.423
C50D10TTLL-2	4	16	36 837	47 550	*Limit*	7.705	3.472
C50D10TTLL-3	8	32	54 596	67 213	*Limit*	10.307	5.789
C50D10TTLL-4	8	32	50 397	69 047	*Limit*	9.413	5.002
C50D10TTLL-5	4	16	37 876	40 846	*Limit*	9.636	4.511
C100D30R-1	4	16	32 858	55 202	*Limit*	7.048	7.048
C100D30R-2	4	16	49 618	77 080	*Limit*	9.704	9.704
C100D30R-3	4	16	54 298	71 028	*Limit*	12.346	4.815
C100D30R-4	4	16	50 957	62 801	*Limit*	8.387	7.632
C100D30R-5	4	16	49 934	68 736	*Limit*	9.388	9.388
C100D30RR-1	4	16	57 103	64 974	*Limit*	20.765	6.711
C100D30RR-2	4	16	46 363	59 242	*Limit*	15.451	11.571
C100D30RR-3	4	16	44 364	51 288	*Limit*	16.677	12.803
C100D30RR-4	4	16	40 271	49 556	*Limit*	13.222	9.129
C100D30RR-5	4	16	47 302	79 645	*Limit*	10.286	10.286
C100D30RL-1	8	32	86 092	110 319	*Limit*	11.112	10.637
C100D30RL-2	8	32	123 498	149 640	*Limit*	9.492	4.585
C100D30RL-3	8	32	81 001	108 599	*Limit*	9.24	7.818
C100D30RL-4	8	32	96 973	136 417	*Limit*	12.893	6.738
C100D30RL-5	8	32	69 814	96 317	*Limit*	9.365	9.028
C100D30RLL-1	8	32	85 527	132 104	*Limit*	13.744	6.734
C100D30RLL-2	8	32	93 463	135 249	*Limit*	8.532	4.617
C100D30RLL-3	8	32	105 447	142 152	*Limit*	14.513	7.632

Instance	CPU	Mem	Rows	Cols	Time	Initial	Best
C100D30RLL-4	8	32	75 467	117 724	*Limit*	21.279	14.348
C100D30RLL-5	8	32	93 714	123 642	*Limit*	9.73	4.788
C100D30RRL-1	8	32	98 310	141 511	*Limit*	12.119	6.153
C100D30RRL-2	8	32	81 822	130 447	*Limit*	15.65	5.083
C100D30RRL-3	8	32	105 894	133 197	*Limit*	14.03	12.45
C100D30RRL-4	8	32	60 911	88 630	*Limit*	10.063	3.768
C100D30RRL-5	8	32	91 275	120 172	*Limit*	12.875	7.482
C100D30RRLL-1	8	32	101 685	123 140	*Limit*	14.297	14.297
C100D30RRLL-2	8	32	86 456	120 056	*Limit*	11.647	5.198
C100D30RRLL-3	8	32	84 579	118 528	*Limit*	11.309	5.378
C100D30RRLL-4	8	32	82 026	137 310	*Limit*	11.873	3.166
C100D30RRLL-5	8	32	76 818	132 716	*Limit*	10.067	2.951
C100D30T-1	4	16	42 532	60 357	*Limit*	7.771	7.771
C100D30T-2	4	16	45 119	70 363	*Limit*	8.355	8.355
C100D30T-3	4	16	64 545	80 403	*Limit*	12.806	12.806
C100D30T-4	4	16	30 153	49 914	*Limit*	9.87	9.87
C100D30T-5	4	16	64 944	74 280	*Limit*	10.695	10.695
C100D30TT-1	4	16	48 551	67 689	*Limit*	7.873	7.873
C100D30TT-2	4	16	43 648	54 116	*Limit*	8.78	7.87
C100D30TT-3	4	16	34 917	47 542	*Limit*	11.188	9.373
C100D30TT-4	4	16	48 275	68 645	*Limit*	11.519	11.519
C100D30TT-5	4	16	42 977	61 330	*Limit*	31.348	31.348
C100D30TL-1	8	32	82 284	114 878	*Limit*	9.441	4.977
C100D30TL-2	8	32	77 963	118 846	*Limit*	7.35	7.35
C100D30TL-3	8	32	108 138	146 948	*Limit*	9.433	9.433
C100D30TL-4	8	32	86 665	123 011	*Limit*	10.39	5.221
C100D30TL-5	8	32	75 445	133 312	*Limit*	8.021	3.974
C100D30TLL-1	8	32	102 834	150 182	*Limit*	6.182	6.06
C100D30TLL-2	8	32	67 058	81 868	*Limit*	9.879	9.109
C100D30TLL-3	8	32	66 540	84 182	*Limit*	10.146	9.226
C100D30TLL-4	8	32	78 348	122 233	*Limit*	9.586	5.0
C100D30TLL-5	8	32	93 282	140 347	*Limit*	8.147	7.918
C100D30TTL-1	8	32	91 059	123 775	*Limit*	9.652	9.086
C100D30TTL-2	8	32	101 790	137 935	*Limit*	9.352	9.243
C100D30TTL-3	8	32	100 408	136 090	*Limit*	8.748	8.748
C100D30TTL-4	8	32	78 911	101 564	*Limit*	7.818	3.659
C100D30TTL-5	8	32	71 990	87 500	*Limit*	9.501	9.247
C100D30TTLL-1	8	32	97 126	154 065	*Limit*	10.569	7.155
C100D30TTLL-2	8	32	89 447	151 741	*Limit*	9.049	4.887
C100D30TTLL-3	8	32	80 855	124 785	*Limit*	7.925	4.286

Instance	CPU	Mem	Rows	Cols	Time	Initial	Best
C100D30TTLL-4	8	32	70 309	97 456	*Limit*	9.958	4.73
C100D30TTLL-5	8	32	93 035	131 445	*Limit*	9.031	9.031
C500D30R-1	8	96	428 970	488 757	*No lower bound*		
C500D30R-2	8	96	417 884	481 633	*No lower bound*		
C500D30R-3	8	96	385 175	481 101	*No lower bound*		
C500D30R-4	8	96	364 069	461 801	*No lower bound*		
C500D30R-5	8	96	375 858	471 748	*No lower bound*		
C500D30RR-1	8	96	360 511	493 829	*Limit*	19.718	19.718
C500D30RR-2	8	96	402 846	510 989	*Out of memory*		
C500D30RR-3	8	96	305 539	449 077	*Limit*	22.268	22.268
C500D30RR-4	8	96	395 933	485 428	*Out of memory*		
C500D30RR-5	8	96	367 130	472 324	*Limit*	31.088	31.088
C500D30RL-1	8	96	523 382	797 460	*Out of memory*		
C500D30RL-2	8	96	533 032	719 818	*Out of memory*		
C500D30RL-3	8	96	408 243	626 389	*Out of memory*		
C500D30RL-4	8	96	433 377	534 066	*Out of memory*		
C500D30RL-5	8	96	525 271	723 998	*Out of memory*		
C500D30RLL-1	8	96	383 751	507 977	*Out of memory*		
C500D30RLL-2	8	96	481 945	621 173	*Out of memory*		
C500D30RLL-3	8	96	433 780	645 107	*Out of memory*		
C500D30RLL-4	8	96	496 397	668 873	*Out of memory*		
C500D30RLL-5	8	96	531 327	787 180	*Out of memory*		
C500D30RRL-1	8	96	385 835	514 332	*Out of memory*		
C500D30RRL-2	8	96	434 801	666 784	*Out of memory*		
C500D30RRL-3	8	96	531 780	759 078	*Out of memory*		
C500D30RRL-4	8	96	485 518	767 749	*Out of memory*		
C500D30RRL-5	8	96	495 384	796 627	*Out of memory*		
C500D30RRLL-1	8	96	466 085	796 152	*Out of memory*		
C500D30RRLL-2	8	96	391 889	551 072	*Out of memory*		
C500D30RRLL-3	8	96	387 384	526 946	*Out of memory*		
C500D30RRLL-4	8	96	480 217	765 092	*Out of memory*		
C500D30RRLL-5	8	96	443 005	613 854	*Out of memory*		
C500D30T-1	8	96	440 289	500 266	*No lower bound*		
C500D30T-2	8	96	380 203	419 027	*No lower bound*		
C500D30T-3	8	96	282 067	407 588	*No lower bound*		
C500D30T-4	8	96	331 445	429 635	*No lower bound*		
C500D30T-5	8	96	439 320	517 328	*No lower bound*		
C500D30TT-1	8	96	364 877	466 235	*No lower bound*		
C500D30TT-2	8	96	369 906	478 065	*No lower bound*		

Instance	CPU	Mem	Rows	Cols	Time	Initial	Best
C500D30TT-3	8	96	435 442	538 014	*No lower bound*		
C500D30TT-4	8	96	278 012	441 237	*No lower bound*		
C500D30TT-5	8	96	399 737	499 170	*No lower bound*		
C500D30TL-1	8	96	460 184	748 489	*Out of memory*		
C500D30TL-2	8	96	505 801	683 875	*Out of memory*		
C500D30TL-3	8	96	423 015	523 119	*Out of memory*		
C500D30TL-4	8	96	499 390	654 670	*Out of memory*		
C500D30TL-5	8	96	451 451	703 450	*Out of memory*		
C500D30TLL-1	8	96	499 073	608 256	*Out of memory*		
C500D30TLL-2	8	96	464 499	664 457	*Out of memory*		
C500D30TLL-3	8	96	549 020	682 605	*Out of memory*		
C500D30TLL-4	8	96	477 336	643 853	*Out of memory*		
C500D30TLL-5	8	96	470 516	663 066	*Out of memory*		
C500D30TTL-1	8	96	513 611	745 157	*Out of memory*		
C500D30TTL-2	8	96	350 151	574 098	*Out of memory*		
C500D30TTL-3	8	96	594 424	775 387	*Out of memory*		
C500D30TTL-4	8	96	529 038	712 071	*Out of memory*		
C500D30TTL-5	8	96	429 230	683 318	*Out of memory*		
C500D30TTLL-1	8	96	462 634	598 715	*Out of memory*		
C500D30TTLL-2	8	96	452 687	600 246	*Out of memory*		
C500D30TTLL-3	8	96	485 686	635 778	*No lower bound*		
C500D30TTLL-4	8	96	581 619	766 774	*Out of memory*		
C500D30TTLL-5	8	96	537 761	675 268	*Out of memory*		

23 | Detailed Listings of Heuristics Performance Metrics

In the following, we list all the details regarding performance metrics that have been used to compare heuristic algorithms in Chapter 12. First, some details related to the benchmark instances provided by Crainic, Frangioni, and Gendron (2001) are shown. Afterwards, we repeat the detailed information published in the original publications regarding solution values, solution times and optimality gaps. We complete the overview by adding the normalized solution times.

Table 23.1: Listing of all C and C+ instances used to compare heuristics performance. If no optimal solution is known, the best known solution, the lower bound and the relative gap is reported instead.

Inst.	Specification	Optimal solution	Lower Bound	Best solution	Gap
C01	25,100,10,V,L	14712	-	-	-
C02	25,100,10,F,L	14941	-	-	-
C03	25,100,10,F,T	49899	-	-	-
C04	25,100,30,V,T	365272	-	-	-
C05	25,100,30,F,L	37055	-	-	-
C06	25,100,30,F,T	85530	-	-	-
C07	100,400,10,V,L	28423	-	-	-
C08	100,400,10,F,L	23949	-	-	-
C09	100,400,10,F,T	-	63066.2	63753	1.1%
C10	100,400,30,V,T	384802	-	-	-
C11	100,400,30,F,L	49018	-	-	-
C12	100,400,30,F,T	-	132128.9	136803	3.5%
C33	20,230,40,V,L	423848	-	-	-
C35	20,230,40,F,T	643036	-	-	-
C36	20,230,40,V,T	371475	-	-	-
C37	20,230,200,V,L	94213	-	-	-
C38	20,230,200,F,L	137642	-	-	-
C39	20,230,200,V,T	97914	-	-	-
C40	20,230,200,F,T	-	135380.1	135863	0.4%
C41	20,300,40,V,L	429398	-	-	-
C42	20,300,40,F,L	586077	-	-	-
C43	20,300,40,V,T	464509	-	-	-
C44	20,300,40,F,T	604198	-	-	-
C45	20,300,200,V,L	-	74753.2	74811	0.1%
C46	20,300,200,F,L	-	113862.3	115526	1.5%
C47	20,300,200,V,T	74991	-	-	-
C48	20,300,200,F,T	-	106671.6	107102	0.4%
C49	30,520,100,V,L	53958	-	-	-
C50	30,520,100,F,L	-	93569.5	93922	0.4%
C51	30,520,100,V,T	52046	-	-	-
C52	30,520,100,F,T	-	96260.3	97098	0.9%
C53	30,520,400,V,L	-	112646.8	112774	0.1%
C54	30,520,400,F,L	-	147790.1	149151	0.9%
C55	30,520,400,V,T	114641	-	-	-
C56	30,520,400,F,T	-	150685	152477	1.2%
C57	30,700,100,V,L	47603	-	-	-
C58	30,700,100,F,L	-	59612.3	59958	0.6%
C59	30,700,100,V,T	45872	-	-	-
C60	30,700,100,F,T	54904	-	-	-
C61	30,700,400,V,L	-	97188.5	97845	0.7%
C62	30,700,400,F,L	-	131689.6	133976	1.7%
C63	30,700,400,V,T	-	94508.1	95250	0.8%
C64	30,700,400,F,T	-	128242.8	129990	1.4%

Table 23.2: Best solution values calculated by the different heuristics on the benchmark instances. Optimal solution values are highlighted in bold font, best solution values in italics.

Inst.	CTS	PR	MCA	CSH	IPS	LB	SACG1
C01	**14712**	**14712**	**14712**	**14712**	-[a]	**14712**	**14712**
C02	**14941**	**14941**	**14941**	15037	-[a]	**14941**	**14941**
C03	**49899**	**49899**	49937	50771	-[a]	**49899**	**49899**
C04	365385	365385	365385	**365272**	-[a]	**365272**	**365272**
C05	37583	37654	37607	37471	-[a]	37326	37060
C06	86296	86428	86461.3	85801	-[a]	**85530**	**85530**
C07	28677	28485	28553	28426	**28423**	**28423**	**28423**
C08	**23949**	24022	24022	24459	**23949**	24690	**23949**
C09	67014	65278	66284	73566	65885	67357	65172
C10	385508	384926	385282	384883	384836	384809	**384802**
C11	51552	51325	50456	51956	49694	49872	49641
C12	145144	141359	145721	144314	141365	141633	141538
C33	424778	424385	426702	424075	424385	**423848**	**423848**
C35	645812	645548	652894	644483	643187	**643036**	**643036**
C36	371893	371811	**371475**	371906	371779	**371475**	**371475**
C37	98995	100404	98582	94247	95097	95295	95295
C38	146535	147988	143150	**137642**	141253	143446	139762
C39	104752	104689	102030	97968	99410	98039	97984
C40	147385	147554	141188	136130	140273	141128	140763
C41	429535	**429398**	429837	**429398**	**429398**	**429398**	**429398**
C42	593322	590427	593544	587800	**586077**	**586077**	**586077**
C43	464724	**464509**	466004	464569	**464509**	**464509**	464627
C44	607100	609990	619203	**604198**	**604198**	**604198**	604201
C45	80819	78184	78209.5	74913	75319	76375	76003
C46	123347	123484	121951	115784	117543	119143	117942
C47	79619	78866.8	77251	75302	76198	76168	75419
C48	114484	113584	111173	107858	110344	109808	109710
C49	54958	54904	55754	54088	54113	54026	53983
C50	99586	102054	99817	94801	94388	96255	94307
C51	52985	53017	53512	52282	52174	52129	52390
C52	105523	106130	102477	98839	98883	101102	100184
C53	120652	119416	115671	112846	114042	114367	114201
C54	161098	163112	156601	149446	154218	157726	155837
C55	121588	120170	120980	**114641**	114922	115240	115614
C56	167939	163675	160217	152744	154606	168561	159976
C57	48398	48723	48869	47635	47612	**47603**	47998
C58	62471	63091	63756	60194	60700	60272	60581
C59	47025	47209	47457	46169	46046	45905	46147
C60	57886	56575.5	56910	55359	55609	55104	55012
C61	106777	105116	102631	97972	98718	103787	101726
C62	148950	145026	143988	135064	152576	169760	160522
C63	101672	101212	99194.9	95306	96168	99195	97046
C64	142778	141013	138266	130148	131629	144926	141491

[a] In the original publication these instances have not been used.

Table 23.3: Best solution values calculated by the different heuristics on the benchmark instances. Optimal solution values are highlighted in bold font, best solution values in italics.

Inst.	SACG2	CS-LB1	CS-LB2	CEA	ParLS	ILPH
C01	**14712**	-[a]	-[a]	**14712**	-[a]	**14712**
C02	**14941**	-[a]	-[a]	**14941**	-[a]	**14941**
C03	**49899**	-[a]	-[a]	**49899**	-[a]	**49899**
C04	**365272**	-[a]	-[a]	**365272**	-[a]	**365272**
C05	37060	-[a]	-[a]	37324	-[a]	**37055**
C06	**85530**	-[a]	-[a]	**85530**	-[a]	**85530**
C07	**28423**	**28423**	**28423**	**28423**	28486	**28423**
C08	**23949**	**23949**	**23949**	**23949**	24022	**23949**
C09	65172	64265	*63753*	65563	64207	65247
C10	**384802**	**384802**	**384802**	384999	**384802**	**384802**
C11	49250	**49018**	**49018**	49466	**49018**	**49018**
C12	141014	136909	*136803*	139535	136861	139177
C33	**423848**	**423848**	**423848**	**423848**	424075	**423848**
C35	**643036**	**643036**	**643036**	643187	**643036**	**643036**
C36	**371475**	**371475**	**371475**	**371475**	371573	**371475**
C37	94283	**94213**	**94213**	94468	**94213**	**94213**
C38	137842	137642.3	137642.3	138954	**137642**	138169
C39	**97914**	**97914**	**97914**	98209	**97914**	**97914**
C40	137072	135888	135863.1	137131	135867	136513
C41	**429398**	**429398**	**429398**	**429398**	**429398**	**429398**
C42	**586077**	**586077**	**586077**	**586077**	**586077**	**586077**
C43	464627	**464509**	**464509**	**464509**	**464509**	**464509**
C44	604201	**604198**	**604198**	**604198**	**604198**	**604198**
C45	74902	*74811*	*74811*	75279	*74811*	74971
C46	116431	115619	*115526*	116801	115580	116375
C47	**74991**	**74991**	**74991**	75444	**74991**	**74991**
C48	108638	107315	107167	107546	*107102*	107298
C49	53983	**53958**	**53958**	54099	53978	**53958**
C50	94066	94043	93967	94621	93967	94066
C51	52247	**52046**	**52046**	52182	**52046**	**52046**
C52	98543	*97098*	97107	97856	97862	97404
C53	113720	112774.4	112774.4	113193	112787	112974
C54	151009	149195.7	*149151*	151145	149677	149945
C55	115581	**114641**	**114641**	115697	**114641**	114798
C56	~~147369~~[b]	152634.8	152476.7	154425	154137	153856
C57	**47603**	**47603**	**47603**	**47603**	**47603**	**47603**
C58	60391	*59958*	*59958*	60538	60058	60049
C59	45956	**45872**	**45872**	46082	45879	45908
C60	54975	**54904**	54912	55135	**54904**	**54904**
C61	99316	97875	97853.4	98729	98090	98385
C62	*133976*	134589.8	134553.7	137112	136257	139663
C63	95538	95249.6	95249.6	96130	95651	95773
C64	131473	130051.2	*129990*	132425	131104	131141

[a] In the original publication these instances have not been used.
[b] The authors provided an erroneous value that is smaller than the lower bound.

Table 23.4: Optimality gap for the best solutions calculated by the different heuristics on the benchmark instances - Part I

Inst.	**CTS**	**PR**	**MCA**	**CSH**	**IPS**	**LB**	**SACG1**
C01	0.0%	0.0%	0.0%	0.0%	-[a]	0.0%	0.0%
C02	0.0%	0.0%	0.0%	0.6%	-[a]	0.0%	0.0%
C03	0.0%	0.0%	0.1%	1.7%	-[a]	0.0%	0.0%
C04	0.0%	0.0%	0.0%	0.0%	-[a]	0.0%	0.0%
C05	1.4%	1.6%	1.5%	1.1%	-[a]	0.7%	0.0%
C06	0.9%	1.0%	1.1%	0.3%	-[a]	0.0%	0.0%
C07	0.9%	0.2%	0.5%	0.0%	0.0%	0.0%	0.0%
C08	0.0%	0.3%	0.3%	2.1%	0.0%	3.1%	0.0%
C09	6.3%	3.5%	5.1%	16.6%	4.5%	6.8%	3.3%
C10	0.2%	0.0%	0.1%	0.0%	0.0%	0.0%	0.0%
C11	5.2%	4.7%	2.9%	6.0%	1.4%	1.7%	1.3%
C12	9.9%	7.0%	10.3%	9.2%	7.0%	7.2%	7.1%
C33	0.2%	0.1%	0.7%	0.1%	0.1%	0.0%	0.0%
C35	0.4%	0.4%	1.5%	0.2%	0.0%	0.0%	0.0%
C36	0.1%	0.1%	0.0%	0.1%	0.1%	0.0%	0.0%
C37	5.1%	6.6%	4.6%	0.0%	0.9%	1.1%	1.1%
C38	6.5%	7.5%	4.0%	0.0%	2.6%	4.2%	1.5%
C39	7.0%	6.9%	4.2%	0.1%	1.5%	0.1%	0.1%
C40	8.9%	9.0%	4.3%	0.6%	3.6%	4.2%	4.0%
C41	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%	0.0%
C42	1.2%	0.7%	1.3%	0.3%	0.0%	0.0%	0.0%
C43	0.0%	0.0%	0.3%	0.0%	0.0%	0.0%	0.0%
C44	0.5%	1.0%	2.5%	0.0%	0.0%	0.0%	0.0%
C45	8.1%	4.6%	4.6%	0.2%	0.8%	2.2%	1.7%
C46	8.3%	8.5%	7.1%	1.7%	3.2%	4.6%	3.6%
C47	6.2%	5.2%	3.0%	0.4%	1.6%	1.6%	0.6%
C48	7.3%	6.5%	4.2%	1.1%	3.4%	2.9%	2.8%
C49	1.9%	1.8%	3.3%	0.2%	0.3%	0.1%	0.0%
C50	6.4%	9.1%	6.7%	1.3%	0.9%	2.9%	0.8%
C51	1.8%	1.9%	2.8%	0.5%	0.2%	0.2%	0.7%
C52	9.6%	10.3%	6.5%	2.7%	2.7%	5.0%	4.1%
C53	7.1%	6.0%	2.7%	0.2%	1.2%	1.5%	1.4%
C54	9.0%	10.4%	6.0%	1.1%	4.3%	6.7%	5.4%
C55	6.1%	4.8%	5.5%	0.0%	0.2%	0.5%	0.8%
C56	11.5%	8.6%	6.3%	1.4%	2.6%	11.9%	6.2%
C57	1.7%	2.4%	2.7%	0.1%	0.0%	0.0%	0.8%
C58	4.8%	5.8%	7.0%	1.0%	1.8%	1.1%	1.6%
C59	2.5%	2.9%	3.5%	0.6%	0.4%	0.1%	0.6%
C60	5.4%	3.0%	3.7%	0.8%	1.3%	0.4%	0.2%
C61	9.9%	8.2%	5.6%	0.8%	1.6%	6.8%	4.7%
C62	13.1%	10.1%	9.3%	2.6%	15.9%	28.9%	21.9%
C63	7.6%	7.1%	5.0%	0.8%	1.8%	5.0%	2.7%
C64	11.3%	10.0%	7.8%	1.5%	2.6%	13.0%	10.3%

[a] In the original publication these instances have not been used.

Table 23.5: Optimality gap for the best solutions calculated by the different heuristics on the benchmark instances - Part II

Inst.	SACG2	CS-LB1	CS-LB2	CEA	ParLS	ILPH
C01	0.0%	-[a]	-[a]	0.0%	-[a]	0.0%
C02	0.0%	-[a]	-[a]	0.0%	-[a]	0.0%
C03	0.0%	-[a]	-[a]	0.0%	-[a]	0.0%
C04	0.0%	-[a]	-[a]	0.0%	-[a]	0.0%
C05	0.0%	-[a]	-[a]	0.7%	-[a]	0.0%
C06	0.0%	-[a]	-[a]	0.0%	-[a]	0.0%
C07	0.0%	0.0%	0.0%	0.0%	0.2%	0.0%
C08	0.0%	0.0%	0.0%	0.0%	0.3%	0.0%
C09	3.3%	1.9%	1.1%	4.0%	1.8%	3.5%
C10	0.0%	0.0%	0.0%	0.1%	0.0%	0.0%
C11	0.5%	0.0%	0.0%	0.9%	0.0%	0.0%
C12	6.7%	3.6%	3.5%	5.6%	3.6%	5.3%
C33	0.0%	0.0%	0.0%	0.0%	0.1%	0.0%
C35	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
C36	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
C37	0.1%	0.0%	0.0%	0.3%	0.0%	0.0%
C38	0.1%	0.0%	0.0%	1.0%	0.0%	0.4%
C39	0.0%	0.0%	0.0%	0.3%	0.0%	0.0%
C40	1.2%	0.4%	0.4%	1.3%	0.4%	0.8%
C41	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
C42	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
C43	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
C44	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
C45	0.2%	0.1%	0.1%	0.7%	0.1%	0.3%
C46	2.3%	1.5%	1.5%	2.6%	1.5%	2.2%
C47	0.0%	0.0%	0.0%	0.6%	0.0%	0.0%
C48	1.8%	0.6%	0.5%	0.8%	0.4%	0.6%
C49	0.0%	0.0%	0.0%	0.3%	0.0%	0.0%
C50	0.5%	0.5%	0.4%	1.1%	0.4%	0.5%
C51	0.4%	0.0%	0.0%	0.3%	0.0%	0.0%
C52	2.4%	0.9%	0.9%	1.7%	1.7%	1.2%
C53	1.0%	0.1%	0.1%	0.5%	0.1%	0.3%
C54	2.2%	1.0%	0.9%	2.3%	1.3%	1.5%
C55	0.8%	0.0%	0.0%	0.9%	0.0%	0.1%
C56	~~-2.2%~~[b]	1.3%	1.2%	2.5%	2.3%	2.1%
C57	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
C58	1.3%	0.6%	0.6%	1.6%	0.7%	0.7%
C59	0.2%	0.0%	0.0%	0.5%	0.0%	0.1%
C60	0.1%	0.0%	0.0%	0.4%	0.0%	0.0%
C61	2.2%	0.7%	0.7%	1.6%	0.9%	1.2%
C62	1.7%	2.2%	2.2%	4.1%	3.5%	6.1%
C63	1.1%	0.8%	0.8%	1.7%	1.2%	1.3%
C64	2.5%	1.4%	1.4%	3.3%	2.2%	2.3%

[a] In the original publication these instances have not been used.
[b] The authors provided an erroneous value that is smaller than the lower bound.

Table 23.6: Computing times (in seconds) for the best solutions calculated by the different heuristics on the benchmark instances - Part I

Inst.	**CTS**	**PR**	**MCA**	**CSH**	**IPS**	**LB**	**SACG1**
C01	49	13	6	1	-[a]	12	25
C02	54	14	7	7	-[a]	600	17
C03	51	24	12	4	-[a]	600	29
C04	224	101	51	3	-[a]	161	65
C05	215	75	38	15	-[a]	600	74
C06	225	97	49	10	-[a]	600	201
C07	336	89	45	6	35	547	163
C08	307	83	41	93	9	600	194
C09	626	210	105	56	813	600	169
C10	1975	493	246	18	330	600	428
C11	1301	315	158	622	886	600	600
C12	1870	481	240	154	888	600	600
C33	370	149	74	3	4	328	118
C35	423	172	86	4	45	600	168
C36	436	157	78	3	41	440	386
C37	2663	2495	1247	442	822	600	600
C38	2718	2878	1439	1658	691	600	600
C39	2566	2211	1105	524	821	600	600
C40	3120	3386	1693	1944	156	600	600
C41	612	225	112	3	19	146	197
C42	582	228	114	6	29	600	283
C43	590	248	124	4	24	600	111
C44	560	214	107	4	68	600	184
C45	4087	3566	1783	348	802	600	600
C46	4368	4013	2006	1290	686	600	600
C47	3808	3924	1962	429	388	600	600
C48	4658	3857	1929	1722	396	600	600
C49	3356	1194	597	27	218	600	600
C50	4032	1460	730	406	226	600	600
C51	3481	1514	757	37	455	600	600
C52	3927	1523	761	200	815	600	600
C53	36531	27477	13739	568	394	600	600
C54	429296	36669	18335	2610	750	600	600
C55	28214	23089	11545	230	621	600	600
C56	40011	52173	26087	1674	466	600	600
C57	4396	1861	930	38	32	600	600
C58	4755	1838	919	114	741	600	600
C59	4560	1894	947	46	371	600	600
C60	4866	1706	853	97	387	600	600
C61	24817	22315	11157	475	222	600	600
C62	69540	75665	37832	1783	860	600	600
C63	34975	24289	12144	749	365	600	600
C64	51878	44936	22468	1487	225	600	600

[a] In the original publication these instances have not been used.

Table 23.7: Computing times (in seconds) for the best solutions calculated by the different heuristics on the benchmark instances - Part II

Inst.	SACG2	CS-LB1	CS-LB2	CEA	ParLS	ILPH
C01	25	-[a]	-[a]	13	-[a]	0
C02	17	-[a]	-[a]	29	-[a]	4
C03	29	-[a]	-[a]	876	-[a]	54
C04	65	-[a]	-[a]	128	-[a]	0
C05	74	-[a]	-[a]	786	-[a]	57
C06	201	-[a]	-[a]	48	-[a]	9
C07	163	77	78	98	3	389
C08	194	6627	10423	567	1	1079
C09	169	2	2	3782	53	2758
C10	428	925	2165	1987	42	735
C11	937	6921	18731	4003	29	610
C12	1207	72	69	12764	79	1816
C33	118	1	1	1082	2	1
C35	168	12	10	965	10	4
C36	386	3	3	124	1	1
C37	2460	2836	6061	3765	123	2413
C38	1100	2523	6722	4054	144	313
C39	2351	2120	2706	3300	52	240
C40	1249	4308	9707	3129	240	947
C41	197	1	1	108	1	169
C42	283	8	8	379	16	21
C43	111	4	4	3456	23	0
C44	184	4	4	267	3	129
C45	2964	4381	10622	3246	361	320
C46	1046	4436	14044	4112	250	2562
C47	4022	1991	2550	3567	58	141
C48	2486	3884	8720	7823	122	1256
C49	1372	476	1680	5324	20	1211
C50	928	4040	7500	6234	83	2755
C51	9127	1670	4048	12878	32	1573
C52	1008	10025	15792	3678	158	2786
C53	12904	9398	9502	15328	542	791
C54	9731	5463	11131	12333	463	434
C55	18000	3598	8263	4234	461	2405
C56	~~9346~~[b]	7178	19594	6454	288	1538
C57	5783	64	64	4101	179	20
C58	3992	3249	7632	12345	111	1549
C59	4201	4264	8807	4801	258	820
C60	6844	3368	7640	11432	173	1842
C61	18000	4792	13312	6346	243	2714
C62	13982	8026	16947	3456	223	1066
C63	18000	5062	10743	10053	374	884
C64	14286	5188	13656	12343	428	1472

[a] In the original publication these instances have not been used.
[b] The authors provided an erroneous value that is smaller than the lower bound.

Table 23.8: Normalized computing times (in seconds) for the best solutions calculated by the different heuristics on the benchmark instances - Part I

Inst.	CTS	PR	MCA	CSH	IPS	LB	SACG1
C01	0.1	0.0	0.8	0.0	-[a]	0.1	0.4
C02	0.1	0.0	0.9	0.1	-[a]	7.1	0.3
C03	0.1	0.0	1.6	0.1	-[a]	7.1	0.5
C04	0.5	0.2	6.6	0.1	-[a]	1.9	1.1
C05	0.4	0.2	4.9	0.2	-[a]	7.1	1.2
C06	0.5	0.2	6.3	0.2	-[a]	7.1	3.3
C07	0.7	0.2	5.8	0.1	1.0	6.5	2.7
C08	0.6	0.2	5.4	1.5	0.3	7.1	3.2
C09	1.3	0.4	13.7	0.9	24.2	7.1	2.8
C10	4.0	1.0	32.1	0.3	9.8	7.1	7.1
C11	2.6	0.6	20.5	10.1	26.4	7.1	10.0
C12	3.8	1.0	31.4	2.5	26.5	7.1	10.0
C33	0.8	0.3	9.7	0.0	0.1	3.9	2.0
C35	0.9	0.4	11.2	0.1	1.3	7.1	2.8
C36	0.9	0.3	10.2	0.1	1.2	5.2	6.4
C37	5.4	5.1	162.7	7.1	24.5	7.1	10.0
C38	5.5	5.9	187.7	26.8	20.6	7.1	10.0
C39	5.2	4.5	144.1	8.5	24.5	7.1	10.0
C40	6.4	6.9	220.7	31.4	4.6	7.1	10.0
C41	1.2	0.5	14.7	0.1	0.6	1.7	3.3
C42	1.2	0.5	14.9	0.1	0.9	7.1	4.7
C43	1.2	0.5	16.2	0.1	0.7	7.1	1.8
C44	1.1	0.4	14.0	0.1	2.0	7.1	3.1
C45	8.3	7.3	232.5	5.6	23.9	7.1	10.0
C46	8.9	8.2	261.6	20.9	20.4	7.1	10.0
C47	7.8	8.0	255.8	6.9	11.6	7.1	10.0
C48	9.5	7.9	251.5	27.8	11.8	7.1	10.0
C49	6.8	2.4	77.9	0.4	6.5	7.1	10.0
C50	8.2	3.0	95.2	6.6	6.7	7.1	10.0
C51	7.1	3.1	98.7	0.6	13.6	7.1	10.0
C52	8.0	3.1	99.3	3.2	24.3	7.1	10.0
C53	74.4	56.0	1791.4	9.2	11.7	7.1	10.0
C54	874.6	74.7	2390.7	42.2	22.3	7.1	10.0
C55	57.5	47.0	1505.3	3.7	18.5	7.1	10.0
C56	81.5	106.3	3401.5	27.1	13.9	7.1	10.0
C57	9.0	3.8	121.3	0.6	1.0	7.1	10.0
C58	9.7	3.7	119.8	1.9	22.1	7.1	10.0
C59	9.3	3.9	123.5	0.7	11.1	7.1	10.0
C60	9.9	3.5	111.2	1.6	11.5	7.1	10.0
C61	50.6	45.5	1454.8	7.7	6.6	7.1	10.0
C62	141.7	154.2	4933.1	28.8	25.6	7.1	10.0
C63	71.3	49.5	1583.6	12.1	10.9	7.1	10.0
C64	105.7	91.6	2929.7	24.0	6.7	7.1	10.0

[a] In the original publication these instances have not been used.

Table 23.9: Normalized computing times (in seconds) for the best solutions calculated by the different heuristics on the benchmark instances - Part II

Inst.	SACG2	CS-LB1	CS-LB2	CEA	ParLS	ILPH
C01	0.4	-[a]	-[a]	0.1	-[a]	0.0
C02	0.3	-[a]	-[a]	0.2	-[a]	0.3
C03	0.5	-[a]	-[a]	5.9	-[a]	4.2
C04	1.1	-[a]	-[a]	0.9	-[a]	0.0
C05	1.2	-[a]	-[a]	5.3	-[a]	4.4
C06	3.3	-[a]	-[a]	0.3	-[a]	0.7
C07	2.7	5.4	5.5	0.7	3	29.9
C08	3.2	464.4	730.4	3.8	1	82.9
C09	2.8	0.1	0.1	25.3	53	212.1
C10	7.1	64.8	151.7	13.3	42	56.5
C11	15.6	485.0	1312.6	26.8	29	46.9
C12	20.1	5.0	4.8	85.3	79	139.6
C33	2.0	0.1	0.1	7.2	2	0.1
C35	2.8	0.9	0.7	6.5	10	0.3
C36	6.4	0.2	0.2	0.8	1	0.1
C37	41.0	198.7	424.7	25.2	123	185.5
C38	18.3	176.8	471.1	27.1	144	24.1
C39	39.1	148.5	189.6	22.1	52	18.4
C40	20.8	301.9	680.2	20.9	240	72.8
C41	3.3	0.1	0.1	0.7	1	13.0
C42	4.7	0.6	0.6	2.5	16	1.6
C43	1.8	0.3	0.3	23.1	23	0.0
C44	3.1	0.3	0.2	1.8	3	10.0
C45	49.4	307.0	744.3	21.7	361	24.6
C46	17.4	310.9	984.2	27.5	250	197.0
C47	67.0	139.5	178.7	23.8	58	10.9
C48	41.4	272.2	611.1	52.3	122	96.6
C49	22.8	33.4	117.8	35.6	20	93.1
C50	15.5	283.1	525.6	41.7	83	211.8
C51	152.0	117.0	283.7	86.1	32	120.9
C52	16.8	702.5	1106.6	24.6	158	214.3
C53	214.9	658.6	665.8	102.5	542	60.8
C54	162.0	382.8	780.0	82.5	463	33.4
C55	299.7	252.1	579.0	28.3	461	185.0
C56	~~155.6~~[b]	503.0	1373.0	43.1	288	118.2
C57	96.3	4.5	4.5	27.4	179	1.5
C58	66.5	227.7	534.8	82.5	111	119.1
C59	69.9	298.8	617.1	32.1	258	63.0
C60	114.0	236.0	535.4	76.4	173	141.6
C61	299.7	335.8	932.8	42.4	243	208.7
C62	232.8	562.4	1187.6	23.1	223	82.0
C63	299.7	354.8	752.8	67.2	374	68.0
C64	237.9	363.5	957.0	82.5	428	113.2

[a] In the original publication these instances have not been used.
[b] The authors provided an erroneous value that is smaller than the lower bound.

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ACCPM	Analytic Center Cutting-Plane Method
API	Application Programming Interface
CPU	Central Processing Unit
DSS	Decision Support System
EDI	Electronic Data Interchange
ESPPRC	Elementary Shortest Path Problem with Resource Constraints
FND	Fixed-Charge Network Design Problem
GDP	Gross Domestic Product
IIS	Irreducible Inconsistent Subset
IP	Integer Program
IQR	Inter-Quartile Range
JMH	Java Microbenchmark Harness
JVM	Java Virtual Machine
LP	Linear Program
LSP	Logistics Service Provider
MCF	Multi-Commodity Network Flow Problem
MCFND	Multi-Commodity Capacitated Fixed-Charge Network Design Problem
MIP	Mixed-Integer Program
MIR	Mixed-Integer Rounding

NLP	Network Loading Problem
PCIMCFND	Path-Constrained Integer Multi-Commodity Capacitated Fixed-Charge Network Design Problem
PTAS	Polynomial-Time Approximation Scheme
R&D	Research and Development
RENS	Relaxation Enforced Neighborhood Search
RFID	Radio-Frequency Identification
RoRo	Roll-on Roll-off
SABP	Simulated Annealing with Branch-and-Price
SACG	Simulated Annealing with Column Generation
SLA	Service Level Agreement
SND	Service Network Design Problem
SPPRC	Shortest Path Problem with Resource Constraints
SPPTW	Shortest Path Problem with Time Windows
TDD	Target Delivery Date
UFND	Uncapacitated Fixed-Charge Network Design Problem
VHC	Vehicle Holding Center
VLNS	Very Large-Scale Neighborhood Search
VMC	Vehicle Modification Center

List of Symbols

$\lvert x \rvert$	Absolute value or modulus of x defined as the non-negative value of x without regard to its sign.
$\lvert S \rvert$	Caridnality of set S measuring the number of elements of the set.
$A \times B$	Cartesian product of sets A and B defined as $\{ (a, b) \mid a \in A, b \in B \}$.
$\lceil x \rceil$	Ceiling function mapping x to the least integer greater than or equal to x.
$\lfloor x \rfloor$	Floor function mapping x to the greatest integer less than or equal to x.
$\text{conv}(X)$	Convex-hull of set X.
$\exp(x)$	Exponential function with argument x.
$n!$	Factorial of a positive integer n defined as $n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$.
$\mathfrak{P}(X)$	Power set of set X.
$\mathcal{P}(X)$	Probability of occurence of event X.
$\mathbb{R}$	Set of real numbers.
$\mathbb{R}_{\geq 0}$	Set of real numbers greater or equal to zero.
$\mathbf{a}$	Vectors are always highlighted in bold font.
$\mathbf{a}^\top$	Transpose of vector $\mathbf{a}$ switching from a row to a column vector and vice versa.
$\mathbb{Z}$	Set of integer numbers.
$\mathbb{Z}_{\geq 0}$	Set of integer numbers greater or equal to zero.

Bibliography

ACEA (2019). *Facts about the Automobile Industry*. Apr. 2019. URL: `https://www.acea.be/automobile-industry/facts-about-the-industry` (visited on 04/29/2019).

Achterberg (2009a). "Constraint Integer Programming." PhD thesis. Technische Universität Berlin, Jan. 28, 2009.

Achterberg (2009b). "SCIP: solving constraint integer programs." In: *Mathematical Programming Computation* 1.1 (July 1, 2009), pp. 1–41.

Achterberg and Wunderling (2013). "Mixed Integer Programming: Analyzing 12 Years of Progress." In: *Facets of Combinatorial Optimization*. Berlin, Heidelberg: Springer, Jan. 1, 2013, pp. 449–481.

Agarwal (2006). "k-Partition-based facets of the network design problem." In: *Networks* 47.3 (May 1, 2006), pp. 123–139.

Agarwal and Aneja (2017). "Fixed charge multicommodity network design using p-partition facets." In: *European Journal of Operational Research* 258.1 (Apr. 1, 2017), pp. 124–135.

Agbegha, Ballou, and Mathur (1998). "Optimizing Auto-Carrier Loading." In: *Transportation Science* 32.2 (May 1, 1998), pp. 174–188.

Aggarwal, Oblak, and Vemuganti (1995). "A heuristic solution procedure for multicommodity integer flows." In: *Computers & Operations Research* 22.10 (Dec. 1995), pp. 1075–1087.

Ahuja, Magnanti, and Orlin (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Feb. 28, 1993.

Alvarez, González-Velarde, and De-Alba (2005). "Scatter Search for Network Design Problem." In: *Annals of Operations Research* 138.1 (Sept. 1, 2005), pp. 159–178.

Alvelos (2005). "Branch-and-price and multicommodity flows." PhD thesis. University of Minho, 2005.

Amdahl (1967). "Validity of the single processor approach to achieving large scale computing capabilities." In: *Proceedings of the April 18-20, 1967, spring joint computer conference*. AFIPS '67 (Spring). New York: Association for Computing Machinery (ACM), Apr. 18, 1967, pp. 483–485.

Amiri and Pirkul (1997). "New formulation and relaxation to solve a concave-cost network flow problem." In: *Journal of the Operational Research Society* 48.3 (Mar. 1, 1997), pp. 278–287.

Andersen, Christiansen, Crainic, and Grønhaug (2010). "Branch and Price for Service Network Design with Asset Management Constraints." In: *Transportation Science* 45.1 (Sept. 24, 2010), pp. 33–49.

Andersen, Crainic, and Christiansen (2009a). "Service network design with asset management: Formulations and comparative analyses." In: *Transportation Research Part C: Emerging Technologies*. Selected papers from the Sixth Triennial Symposium on Transportation Analysis (TRISTAN VI) 17.2 (Apr. 1, 2009), pp. 197–207.

Andersen, Crainic, and Christiansen (2009b). "Service network design with management and coordination of multiple fleets." In: *European Journal of Operational Research* 193.2 (Mar. 1, 2009), pp. 377–389.

Assad (1978). "Multicommodity network flows—A survey." In: *Networks* 8.1 (Mar. 1, 1978), pp. 37–91.

Atamtürk (2001). "Flow pack facets of the single node fixed-charge flow polytope." In: *Operations Research Letters* 29.3 (Oct. 1, 2001), pp. 107–114.

Atamtürk (2002). "On capacitated network design cut–set polyhedra." In: *Mathematical Programming* 92.3 (May 1, 2002), pp. 425–437.

Atamtürk and Günlük (2007). "Network design arc set with variable upper bounds." In: *Networks* 50.1 (Apr. 16, 2007), pp. 17–28.

Atamtürk and Rajan (2002). "On splittable and unsplittable flow capacitated network design arc–set polyhedra." In: *Mathematical Programming* 92.2 (Apr. 1, 2002), pp. 315–333.

Awerbuch and Leighton (1994). "Improved Approximation Algorithms for the Multicommodity Flow Problem and Local Competitive Routing in Dynamic Networks." In: *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*. STOC '94. New York: Association for Computing Machinery (ACM), 1994, pp. 487–496.

Babonneau, du Merle, and Vial (2006). "Solving Large-Scale Linear Multicommodity Flow Problems with an Active Set Strategy and Proximal-ACCPM." In: *Operations Research* 54.1 (Feb. 1, 2006), pp. 184–197.

Balakrishnan and Graves (1985). *A Composite Algorithm for the Concave-Cost LTL Consolidation*. Research rep. Massachusetts Institute of Technology, Sloan School of Management, 1985.

Balakrishnan, Magnanti, and Mirchandani (1997). "Network design." In: *Annotated bibliographies in combinatorial optimization* (1997), pp. 311–334.

Bang-Jensen and Gutin (2008). *Digraphs: Theory, Algorithms and Applications*. London: Springer, Dec. 17, 2008. 812 pp.

Bannister and Eppstein (2012). "Randomized Speedup of the Bellman-Ford Algorithm." In: *2012 Proceedings of the Ninth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*. Society for Industrial and Applied Mathematics (SIAM), Jan. 16, 2012, pp. 41–47.

Barahona (1996). "Network Design Using Cut Inequalities." In: *SIAM Journal on Optimization* 6.3 (Aug. 1, 1996), pp. 823–837.

Bärmann (2016). *Solving Network Design Problems via Decomposition, Aggregation and Approximation*. Wiesbaden: Springer Fachmedien, 2016.

Barnhart, Hane, Johnson, and Sigismondi (1994). "A column generation and partitioning approach for multi-commodity flow problems." In: *Telecommunication Systems* 3.3 (Oct. 1, 1994), pp. 239–258.

Barnhart, Hane, and Vance (1996). "Integer multicommodity flow problems." In: *Integer Programming and Combinatorial Optimization*. International Conference on Integer Programming and Combinatorial Optimization. Ed. by Cunningham, McCormick, and Queyranne. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, June 3, 1996, pp. 58–71.

Barnhart, Hane, and Vance (2000). "Using Branch-and-Price-and-Cut to Solve Origin-Destination Integer Multicommodity Flow Problems." In: *Operations Research* 48.2 (Apr. 1, 2000), pp. 318–326.

Barnhart and Sheffi (1993). "A Network-Based Primal-Dual Heuristic for the Solution of Multicommodity Network Flow Problems." In: *Transportation Science* 27.2 (May 1, 1993), pp. 102–117.

Barr, Golden, Kelly, Resende, and Stewart (1995). "Designing and reporting on computational experiments with heuristic methods." In: *Journal of Heuristics* 1.1 (Sept. 1, 1995), pp. 9–32.

BEA (2019). *Auto and Truck Seasonal Adjustment*. Dec. 2019. URL: `https://www.bea.gov/docs/gdp/auto-and-truck-seasonal-adjustment` (visited on 12/18/2019).

Beasley and Christofides (1989). "An algorithm for the resource constrained shortest path problem." In: *Networks* 19.4 (1989), pp. 379–394.

Bektaş (2017). *Freight Transport and Distribution: Concepts and Optimisation Models*. CRC Press, June 19, 2017. 286 pp.

Bektaş, Chouman, and Crainic (2010). "Lagrangean-based decomposition algorithms for multicommodity network design problems with penalized constraints." In: *Networks* 55.3 (May 1, 2010), pp. 171–180.

Ben-Ameur (2004). "Computing the Initial Temperature of Simulated Annealing." In: *Computational Optimization and Applications* 29 (Dec. 1, 2004), pp. 369–385.

Berthold (2006). "Primal heuristics for mixed integer programs." MA thesis. Technische Universität Berlin, 2006.

Bertsekas (1993). "A simple and fast label correcting algorithm for shortest paths." In: *Networks* 23.8 (1993), pp. 703–709.

Bhasker and Samad (1991). "The clique-partitioning problem." In: *Computers & Mathematics with Applications* 22.6 (Jan. 1, 1991), pp. 1–11.

Bienstock, Chopra, Günlük, and Tsai (1998). "Minimum cost capacity installation for multicommodity network flows." In: *Mathematical Programming* 81.2 (Apr. 1, 1998), pp. 177–199.

Bienstock and Günlük (1996). "Capacitated Network Design - Polyhedral Structure and Computation." In: *INFORMS Journal on Computing* 8.3 (Aug. 1, 1996), pp. 243–259.

Boland, Dethridge, and Dumitrescu (2006). "Accelerated label setting algorithms for the elementary resource constrained shortest path problem." In: *Operations Research Letters* 34.1 (Jan. 1, 2006), pp. 58–68.

Boland, Hewitt, Marshall, and Savelsbergh (2017). "The Continuous Time Service Network Design Problem." In: *Operations Research* 65.5 (June 25, 2017), pp. 1303–1321.

Bookbinder and Sethi (1980). "The dynamic transportation problem: A survey." In: *Naval Research Logistics Quarterly* 27.1 (Mar. 1, 1980), pp. 65–87.

Boujelben, Gicquel, and Minoux (2012). "A supply chain network design problem under flow consolidation constraints." In: *9th International Conference on Modeling, Optimization & Simulation (MOSIM'12)*. 2012.

Brélaz (1979). "New methods to color the vertices of a graph." In: *Communications of the ACM* 22 (Apr. 1, 1979), pp. 251–256.

CAAM (2019). *Automotive Statistics*. Dec. 2019. URL: `http://www.caam.org.cn/chn/21/cate_463/list_1.html` (visited on 12/18/2019).

Çakır (2009). "Benders decomposition applied to multi-commodity, multi-mode distribution planning." In: *Expert Systems with Applications* 36.4 (May 1, 2009), pp. 8212–8217.

Castro (2000). "A Specialized Interior-Point Algorithm for Multicommodity Network Flows." In: *SIAM Journal on Optimization* 10.3 (Jan. 1, 2000), pp. 852–877.

Chabrier (2003). "Heuristic branch-and-price-and-cut to solve a network design problem." In: *Proceedings of the Fifth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2003)*. Ed. by Gendreau, Pesant, and Rousseau. May 8, 2003.

Chabrier (2006). "Vehicle Routing Problem with elementary shortest path based column generation." In: *Computers & Operations Research*. Part Special Issue: Constraint Programming 33.10 (Oct. 1, 2006), pp. 2972–2990.

Chabrier, Danna, Le Pape, and Perron (2004). "Solving a Network Design Problem." In: *Annals of Operations Research* 130.1 (Aug. 1, 2004), pp. 217–239.

Chandra, Ghosh, and Srivastava (2016). "Outbound logistics management practices in the automotive industry: an emerging economy perspective." In: *Decision* 43.2 (June 1, 2016), pp. 145–165.

Chang (2008). "Best routes selection in international intermodal networks." In: *Computers & Operations Research*. Part Special Issue: Bio-inspired Methods in Combinatorial Optimization 35.9 (Sept. 1, 2008), pp. 2877–2891.

Chardaire and Lisser (2002). "Simplex and Interior Point Specialized Algorithms for Solving Nonoriented Multicommodity Flow Problems." In: *Operations Research* 50.2 (Apr. 1, 2002), pp. 260–276.

Chen (2004). "Integrated Production and Distribution Operations." In: *Handbook of Quantitative Supply Chain Analysis*. Ed. by Simchi-Levi, Wu, and Shen. International Series in Operations Research & Management Science. Boston: Springer, 2004, pp. 711–745.

Cherkassky and Goldberg (1999). "Negative-cycle detection algorithms." In: *Mathematical Programming* 85.2 (1999), pp. 277–311.

Chouman and Crainic (2010). *A MIP-tabu search hybrid framework for multicommodity capacitated fixed-charge network design*. Tech. rep. CIRRELT-2010-31. Montréal: Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, 2010.

Chouman, Crainic, and Gendron (2003). *A cutting-plane algorithm based on cutset inequalities for multicommodity capacitated fixed charge network design*. Tech. rep. Montréal: Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, 2003.

Chouman, Crainic, and Gendron (2009). *A cutting-plane algorithm for multicommodity capacitated fixed-charge network design*. Tech. rep. CIRRELT-2009-20. Montréal: Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, 2009.

Chouman, Crainic, and Gendron (2016). "Commodity Representations and Cut-Set-Based Inequalities for Multicommodity Capacitated Fixed-Charge Network Design." In: *Transportation Science* (July 27, 2016).

Chouman, Crainic, and Gendron (2018). "The impact of filtering in a branch-and-cut algorithm for multicommodity capacitated fixed charge network design." In: *EURO Journal on Computational Optimization* 6.2 (June 1, 2018), pp. 143–184.

Christiansen, Fagerholt, Nygreen, and Ronen (2007). "Maritime Transportation." In: *Handbooks in Operations Research and Management Science*. Ed. by Laporte and Barnhart. Vol. 14. Transportation. Elsevier, 2007, pp. 189–284.

Cordeau, Dell'Amico, Falavigna, and Iori (2015). "A rolling horizon algorithm for auto-carrier transportation." In: *Transportation Research Part B: Methodological* 76 (June 1, 2015), pp. 68–80.

Cordeau, Pasin, and Solomon (2006). "An integrated model for logistics network design." In: *Annals of Operations Research* 144.1 (Apr. 1, 2006), pp. 59–82.

Cordeau, Toth, and Vigo (1998). "A Survey of Optimization Models for Train Routing and Scheduling." In: *Transportation Science* 32.4 (Nov. 1, 1998), pp. 380–404.

Cormen, Leiserson, Rivest, and Stein (2013). *Introduction to Algorithms*. 3rd ed. The MIT Press, Dec. 20, 2013. 1292 pp.

Costa (2005). "A survey on benders decomposition applied to fixed-charge network design problems." In: *Computers & Operations Research* 32.6 (June 1, 2005), pp. 1429–1450.

Costa, Cordeau, and Gendron (2009). "Benders, metric and cutset inequalities for multicommodity capacitated network design." In: *Computational Optimization and Applications* 42.3 (Apr. 1, 2009), pp. 371–392.

Costa, Cordeau, Gendron, and Laporte (2012). "Accelerating benders decomposition with heuristic master problem solutions." In: *Pesquisa Operacional* 32.1 (Apr. 2012), pp. 03–20.

Crainic (2000). "Service network design in freight transportation." In: *European Journal of Operational Research* 122.2 (Apr. 16, 2000), pp. 272–288.

Crainic (2003). "Long-Haul Freight Transportation." In: *Handbook of Transportation Science*. Ed. by Hall. International Series in Operations Research & Management Science 56. Boston: Springer, 2003, pp. 451–516.

Crainic, Frangioni, and Gendron (2001). "Bundle-based relaxation methods for multicommodity capacitated fixed charge network design." In: *Discrete Applied Mathematics*. Combinatorial Optimization Symposium, Selected Papers 112.1 (Sept. 15, 2001), pp. 73–99.

Crainic and Gendreau (2002). "Cooperative Parallel Tabu Search for Capacitated Network Design." In: *Journal of Heuristics* 8.6 (Nov. 1, 2002), pp. 601–627.

Crainic and Gendreau (2007). "A Scatter Search Heuristic for the Fixed-Charge Capacitated Network Design Problem." In: *Metaheuristics: Progress in Complex Systems Optimization*. Ed. by Doerner, Gendreau, Greistorfer, Gutjahr, Hartl, and Reimann. Operations Research/Computer Science Interfaces Series. Boston: Springer, 2007, pp. 25–40.

Crainic, Gendreau, and Farvolden (2000). "A Simplex-Based Tabu Search Method for Capacitated Network Design." In: *INFORMS Journal on Computing* 12.3 (Aug. 1, 2000), pp. 223–236.

Crainic, Gendron, and Hernu (2004). "A Slope Scaling/Lagrangean Perturbation Heuristic with Long-Term Memory for Multicommodity Capacitated Fixed-Charge Network Design." In: *Journal of Heuristics* 10.5 (Sept. 1, 2004), pp. 525–545.

Crainic, Hewitt, Toulouse, and Vu (2014). "Service Network Design with Resource Constraints." In: *Transportation Science* 50.4 (July 2, 2014), pp. 1380–1393.

Crainic and Kim (2007). "Intermodal Transportation." In: *Handbooks in Operations Research and Management Science*. Ed. by Laporte and Barnhart. Vol. 14. Transportation. Elsevier, 2007, pp. 467–537.

Crainic and Laporte (1997). "Planning models for freight transportation." In: *European Journal of Operational Research* 97.3 (Mar. 16, 1997), pp. 409–438.

Crainic, Li, and Toulouse (2006). "A first multilevel cooperative algorithm for capacitated multicommodity network design." In: *Computers & Operations Research*. Part Special Issue: Anniversary Focused Issue of Computers & Operations Research on Tabu Search 33.9 (Sept. 1, 2006), pp. 2602–2622.

Crainic and Semet (2013). "Operations Research and Goods Transportation." In: *Applications of Combinatorial Optimization*. Ed. by Paschos. New York: John Wiley & Sons, 2013, pp. 111–175.

Croxton, Gendron, and Magnanti (2003a). "A Comparison of Mixed-Integer Programming Models for Nonconvex Piecewise Linear Cost Minimization Problems." In: *Management Science* 49.9 (Sept. 1, 2003), pp. 1268–1273.

Croxton, Gendron, and Magnanti (2003b). "Models and Methods for Merge-in-Transit Operations." In: *Transportation Science* 37.1 (Feb. 1, 2003), pp. 1–22.

Daihatsu Motor Co. (2019). *Facilities*. Apr. 2019. URL: `https://www.daihatsu.com/company/facilities/` (visited on 04/19/2019).

Dall'Orto, Crainic, Leal, and Powell (2006). "The single-node dynamic service scheduling and dispatching problem." In: *European Journal of Operational Research* 170.1 (Apr. 1, 2006), pp. 1–23.

Dantzig (1951). "Maximization of a linear function of variables subject to linear inequalities." In: *Activity analysis of production and allocation* 13 (1951), pp. 339–347.

Dantzig, Fulkerson, and Johnson (1954). "Solution of a large-scale traveling-salesman problem." In: *Journal of the operations research society of America* 2.4 (1954), pp. 393–410.

Dantzig and Wolfe (1960). "Decomposition Principle for Linear Programs." In: *Operations Research* 8.1 (Feb. 1, 1960), pp. 101–111.

Dechter and Pearl (1985). "Generalized best-first search strategies and the optimality of A*." In: *Journal of the ACM* 32.3 (July 1, 1985), pp. 505–536.

Dehghan Nayeri (2017). "Decomposition Algorithm in Fixed Charge Time-Space Network Flow Problems." MA thesis. Norman: University of Oklahoma, Dec. 15, 2017.

Dell'Amico, Righini, and Salani (2006). "A Branch-and-Price Approach to the Vehicle Routing Problem with Simultaneous Distribution and Collection." In: *Transportation Science* 40.2 (May 1, 2006), pp. 235–247.

Desaulniers, Desrosiers, and Solomon (2002). "Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems." In: *Essays and Surveys in Metaheuristics*. Ed. by Ribeiro and Hansen. Operations Research/Computer Science Interfaces Series. Boston: Springer, 2002, pp. 309–324.

Desrochers (1986). *An algorithm for the shortest path problem with resource constraints*. Université de Montréal, Centre de recherche sur les transports, 1986.

Desrochers and Soumis (1988). "A Generalized Permanent Labelling Algorithm For The Shortest Path Problem With Time Windows." In: *INFOR: Information Systems and Operational Research* 26.3 (Jan. 1, 1988), pp. 191–212.

Desrosiers and Lübbecke (2005). "A Primer in Column Generation." In: *Column Generation*. Ed. by Desaulniers, Desrosiers, and Solomon. Boston: Springer, 2005, pp. 1–32.

Di Puglia Pugliese and Guerriero (2012). "A Reference Point Approach for the Resource Constrained Shortest Path Problems." In: *Transportation Science* 47.2 (June 21, 2012), pp. 247–265.

Di Puglia Pugliese and Guerriero (2013). "A survey of resource constrained shortest path problems: Exact solution approaches." In: *Networks* 62.3 (2013), pp. 183–200.

Dial, Glover, Karney, and Klingman (1979). "A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees." In: *Networks* 9.3 (1979), pp. 215–248.

Diestel (2017). *Graph Theory*. 5th ed. Berlin, Heidelberg: Springer, 2017.

Dijkstra (1959). "A note on two problems in connexion with graphs." In: *Numerische Mathematik* 1.1 (1959), pp. 269–271.

Dolan and Moré (2002). "Benchmarking optimization software with performance profiles." In: *Mathematical Programming* 91.2 (Jan. 1, 2002), pp. 201–213.

Dongarra (2014). *Performance of Various Computers Using Standard Linear Equations Software, (Linpack Benchmark Report)*. Tech. rep. CS-89-85. Knoxville: University of Tennessee, 2014.

Dorndorf and Kneis (2014). "Optimiertes Terminalmanagement in der Fahrzeugdistribution." In: *Zukunftsperspektiven des Operations Research*. Ed. by Lübbecke, Weiler, and Werners. Wiesbaden: Springer Gabler, 2014, pp. 117–128.

Drexl (2020). *Resource-Constrained Shortest Paths*. Boost Graph Library. Feb. 2020. URL: `https://www.boost.org/doc/libs/release/libs/graph/doc/r_c_shortest_paths.html` (visited on 02/11/2020).

Dror (1994). "Note on the Complexity of the Shortest Path Models for Column Generation in VRPTW." In: *Operations Research* 42.5 (Oct. 1, 1994), pp. 977–978.

Ekşioğlu, Ekşioğlu, Walden, Jin, and Acharya (2010). "Automotive distribution network design: a support system for transportation infrastructure decision makers." In: *International Journal of Business and Systems Research* 4.4 (Jan. 1, 2010), pp. 379–401.

Ekşioğlu, Pardalos, and Romeijn (2002). "A Dynamic Slope Scaling Procedure for the Fixed-Charge Cost Multi-Commodity Network Flow Problem." In: *Financial Engineering, E-commerce and Supply Chain*. Ed. by Pardalos and Tsitsiringos. Applied Optimization. Boston: Springer, 2002, pp. 247–270.

Eppstein (2000). "Fast hierarchical clustering and other applications of dynamic closest pairs." In: *Journal of Experimental Algorithmics* 5 (2000).

Eskigun, Uzsoy, Preckel, Beaujon, Krishnan, and Tew (2005). "Outbound supply chain network design with mode selection, lead times and capacitated vehicle distribution centers." In: *European Journal of Operational Research* 165.1 (Aug. 16, 2005), pp. 182–206.

Farvolden and Powell (1994). "Subgradient Methods for the Service Network Design Problem." In: *Transportation Science* 28.3 (Aug. 1, 1994), pp. 256–272.

Farvolden, Powell, and Lustig (1993). "A Primal Partitioning Solution for the Arc-Chain Formulation of a Multicommodity Network Flow Problem." In: *Operations Research* 41.4 (Aug. 1, 1993), pp. 669–693.

Feillet, Dejax, Gendreau, and Gueguen (2004). "An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems." In: *Networks* 44.3 (Oct. 1, 2004), pp. 216–229.

Fischer and Helmberg (2012). "Dynamic graph generation for the shortest path problem in time expanded networks." In: *Mathematical Programming* 143.1 (Oct. 30, 2012), pp. 257–297.

Fischetti and Lodi (2003). "Local branching." In: *Mathematical Programming* 98.1 (Sept. 1, 2003), pp. 23–47.

Fleischer and Skutella (2007). "Quickest Flows Over Time." In: *SIAM Journal on Computing* 36.6 (Jan. 1, 2007), pp. 1600–1630.

Fleischer and Tardos (1998). "Efficient continuous-time dynamic network flow algorithms." In: *Operations Research Letters* 23.3 (Oct. 1998), pp. 71–80.

Ford and Fulkerson (1962). *Flows in Networks*. Princeton University Press, June 1962. 198 pp.

Ford Motor Company (2019). *Operations Worldwide*. Dec. 2019. URL: `https://corporate.ford.com/company/operation-list.html` (visited on 12/14/2019).

Fragkos, Cordeau, and Jans (2017). *The multi-period multi-commodity network design problem*. Tech. rep. CIRRELT-2017-63. Montréal: Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, 2017.

Frangioni and Gendron (2009). "0–1 reformulations of the multicommodity capacitated network design problem." In: *Discrete Applied Mathematics*. Reformulation Techniques and Mathematical Programming 157.6 (Mar. 28, 2009), pp. 1229–1241.

Frangioni and Gendron (2013). "A stabilized structured Dantzig–Wolfe decomposition method." In: *Mathematical Programming* 140.1 (Aug. 1, 2013), pp. 45–76.

Fu, Sun, and Rilett (2006). "Heuristic shortest path algorithms for transportation applications: State of the art." In: *Computers & Operations Research*. Part Special Issue: Operations Research and Data Mining 33.11 (Nov. 2006), pp. 3324–3343.

Garey and Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman & Co., Apr. 26, 1979. 340 pp.

Gendron (2011). "Decomposition Methods for Network Design." In: *Procedia - Social and Behavioral Sciences*. The State of the Art in the European Quantitative Oriented Transportation and Logistics Research – 14th Euro Working Group on Transportation & 26th Mini Euro Conference & 1st European Scientific Conference on Air Transport 20 (Jan. 1, 2011), pp. 31–37.

Gendron and Crainic (1994). *Relaxations for multicommodity capacitated network design problems*. Tech. rep. CRT-965. Université de Montréal, Centre de recherche sur les transports, 1994.

Gendron, Crainic, and Frangioni (1999). "Multicommodity Capacitated Network Design." In: *Telecommunications Network Planning*. Ed. by Sansò and Soriano. Centre for Research on Transportation. Boston: Springer, 1999, pp. 1–19.

Gendron and Gouveia (2016). "Reformulations by Discretization for Piecewise Linear Integer Multicommodity Network Flow Problems." In: *Transportation Science* (June 20, 2016).

Gendron, Hanafi, and Todosijević (2018). "Matheuristics based on iterative linear programming and slope scaling for multicommodity capacitated fixed charge network design." In: *European Journal of Operational Research* 268.1 (July 1, 2018), pp. 70–81.

Gendron and Larose (2014). "Branch-and-price-and-cut for large-scale multicommodity capacitated fixed-charge network design." In: *EURO Journal on Computational Optimization* 2.1 (June 1, 2014), pp. 55–75.

General Motors (2019). *Plants & Facilities*. Dec. 2019. URL: `https://media.gm.com/media/us/en/gm/plants-facilities.html` (visited on 12/14/2019).

Geoffrion and Graves (1974). "Multicommodity Distribution System Design by Benders Decomposition." In: *Management Science* 20.5 (Jan. 1, 1974), pp. 822–844.

GeoNames (2019). *Gazetteer Export Dumps*. Dec. 2019. URL: `http://download.geonames.org/export/dump/` (visited on 12/20/2019).

Ghamlouche, Crainic, and Gendreau (2003). "Cycle-Based Neighbourhoods for Fixed-Charge Capacitated Multicommodity Network Design." In: *Operations Research* 51.4 (Aug. 1, 2003), pp. 655–667.

Ghamlouche, Crainic, and Gendreau (2004). "Path Relinking, Cycle-Based Neighbourhoods and Capacitated Multicommodity Network Design." In: *Annals of Operations Research* 131.1 (Oct. 1, 2004), pp. 109–133.

Ghamlouche, Crainic, Gendreau, and Sbeity (2011). "Learning mechanisms and local search heuristics for the fixed charge capacitated multicommodity network design." In: *International Journal of Computer Science Issues* 8.6 (2011), p. 5.

Gilmore and Gomory (1961). "A Linear Programming Approach to the Cutting-Stock Problem." In: *Operations Research* 9.6 (Dec. 1, 1961), pp. 849–859.

Glover and Laguna (1998). "Tabu search." In: *Handbook of combinatorial optimization*. Boston: Springer, 1998, pp. 2093–2229.

Glover, Laguna, and Martí (2000). "Fundamentals of scatter search and path relinking." In: *Control and cybernetics* 29.3 (2000), pp. 653–684.

Goffin and Vial (2002). "Convex nondifferentiable optimization: A survey focused on the analytic center cutting plane method." In: *Optimization Methods and Software* 17.5 (Jan. 1, 2002), pp. 805–867.

Gomory (1958). "Outline of an algorithm for integer solutions to linear programs." In: *Bulletin of the American Mathematical society* 64.5 (1958), pp. 275–278.

Gong (1996). "Capacitated network design with column generation." PhD thesis. Georgia Institute of Technology, May 1996.

Google Earth (2021). *Satellite image of a vehicle compound (coordinates 51°18'33"N 3°13'07"E)*. 2021. URL: `https://earth.google.com/` (visited on 07/24/2021).

Gu, Nemhauser, and Savelsbergh (1998). "Lifted Cover Inequalities for 0-1 Integer Programs: Computation." In: *INFORMS Journal on Computing* 10.4 (Nov. 1, 1998), pp. 427–437.

Gu, Nemhauser, and Savelsbergh (1999). "Lifted Cover Inequalities for 0-1 Integer Programs: Complexity." In: *INFORMS Journal on Computing* 11.1 (Feb. 1, 1999), pp. 117–123.

Guélat, Florian, and Crainic (1990). "A Multimode Multiproduct Network Assignment Model for Strategic Planning of Freight Flows." In: *Transportation Science* 24.1 (Feb. 1, 1990), pp. 25–39.

Gunasekaran and Ngai (2005). "Build-to-order supply chain management: a literature review and framework for development." In: *Journal of Operations Management*. The Build to Order Supply Chain (BOSC) 23.5 (July 1, 2005), pp. 423–451.

Günlük (1999). "A branch-and-cut algorithm for capacitated network design problems." In: *Mathematical Programming* 86.1 (Sept. 1, 1999), pp. 17–39.

Haghani and Oh (1996). "Formulation and solution of a multi-commodity, multi-modal network flow model for disaster relief operations." In: *Transportation Research Part A: Policy and Practice* 30.3 (May 1996), pp. 231–250.

Hall, Hippler, and Skutella (2007). "Multicommodity flows over time: Efficient algorithms and complexity." In: *Theoretical Computer Science*. Automata, Languages and Programming 379.3 (June 15, 2007), pp. 387–404.

Hart, Nilsson, and Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths." In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

Haubrich (2017). "Optimierung der Fahrzeugdistribution mit Software der INFORM GmbH." In: *Automobillogistik*. Ed. by Göpfert, Braun, and Schulz. Wiesbaden: Springer Gabler, 2017, pp. 287–304.

Hawick and James (2008). "Enumerating Circuits and Loops in Graphs with Self-Arcs and Multiple-Arcs." In: *Proceedings of 2008 International Conference on Foundations of Computer Science (FCS'08)*. Las Vegas: CSREA, 2008, pp. 14–20.

Hei, Meng, Wang, and Mao (2014). "Optimal Automobile Distribution Model in Multimodal Freight Transportation Networks." In: *Transportation Research Record: Journal of the Transportation Research Board* 2410 (Sept. 18, 2014), pp. 50–57.

Helber and Sahling (2010). "A fix-and-optimize approach for the multi-level capacitated lot sizing problem." In: *International Journal of Production Economics* 123.2 (Feb. 2010), pp. 247–256.

Hewitt, Nemhauser, and Savelsbergh (2009). "Combining Exact and Heuristic Approaches for the Capacitated Fixed-Charge Network Flow Problem." In: *INFORMS Journal on Computing* 22.2 (Sept. 21, 2009), pp. 314–325.

Hewitt, Nemhauser, and Savelsbergh (2012). "Branch-and-Price Guided Search for Integer Programs with an Application to the Multicommodity Fixed-Charge Network Flow Problem." In: *INFORMS Journal on Computing* 25.2 (Apr. 11, 2012), pp. 302–316.

Hino Motors (2019). *Global Network*. Apr. 2019. URL: `https://www.hino-global.com/corp/about_us/network/` (visited on 04/19/2019).

Holmberg and Yuan (2000). "A Lagrangian Heuristic Based Branch-and-Bound Approach for the Capacitated Network Design Problem." In: *Operations Research* 48.3 (June 1, 2000), pp. 461–481.

Holmberg and Yuan (2003). "A Multicommodity Network-Flow Problem with Side Constraints on Paths Solved by Column Generation." In: *INFORMS Journal on Computing* 15.1 (Feb. 1, 2003), pp. 42–57.

Holweg and Miemczyk (2002). "Logistics in the "three-day car" age: Assessing the responsiveness of vehicle distribution logistics in the UK." In: *International Journal of Physical Distribution & Logistics Management* 32.10 (Dec. 1, 2002), pp. 829–850.

Hoppe and Tardos (2000). "The Quickest Transshipment Problem." In: *Mathematics of Operations Research* 25.1 (Feb. 1, 2000), pp. 36–62.

Hu, Zhao, Tao, and Sheng (2015). "Finished-vehicle transporter routing problem solved by loading pattern discovery." In: *Annals of Operations Research* 234.1 (Nov. 1, 2015), pp. 37–56.

Hyundai Group (2019). *Manufacturing*. Apr. 2019. URL: `https://www.hyundai.com/worldwide/en/company/corporate/networks/manufacturing` (visited on 04/24/2019).

Iijima and Sugawara (2005). "Logistics innovation for Toyota's world car strategy." In: *International Journal of Integrated Supply Management* 1.4 (2005), p. 478.

Irnich and Desaulniers (2005). "Shortest Path Problems with Resource Constraints." In: *Column Generation*. Ed. by Desaulniers, Desrosiers, and Solomon. Boston: Springer, 2005, pp. 33–65.

JAMA (2019). *Active Matrix Database System*. Dec. 2019. URL: `http://jamaserv.jama.or.jp/newdb/eng/index.html` (visited on 12/16/2019).

Jarrah, Johnson, and Neubert (2009). "Large-Scale, Less-than-Truckload Service Network Design." In: *Operations Research* 57.3 (Mar. 11, 2009), pp. 609–625.

Jayaraman (1998). "Transportation, facility location and inventory issues in distribution network design: An investigation." In: *International Journal of Operations & Production Management* 18.5 (May 1, 1998), pp. 471–494.

Jin, Ekşioğlu, Ekşioğlu, and Wang (2010). "Mode Selection for Automotive Distribution with Quantity Discounts." In: *Networks and Spatial Economics* 10.1 (Mar. 1, 2010), pp. 1–13.

Jin, Luo, and Ekşioğlu (2008). "Integration of production sequencing and outbound logistics in the automotive industry." In: *International Journal of Production Economics*. Special Section on Advanced Modeling and Innovative Design of Supply Chain 113.2 (June 1, 2008), pp. 766–774.

Jones, Lustig, Farvolden, and Powell (1993). "Multicommodity network flows: The impact of formulation on decomposition." In: *Mathematical Programming* 62.1 (Feb. 1, 1993), pp. 95–117.

Karmarkar (1984). "A new polynomial-time algorithm for linear programming." In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. New York: Association for Computing Machinery (ACM), 1984, pp. 302–311.

Karp (1972). "Reducibility among Combinatorial Problems." In: *Complexity of Computer Computations*. Ed. by Miller, Thatcher, and Bohlinger. The IBM Research Symposia Series. Boston: Springer, 1972, pp. 85–103.

Katayama (2015). "A combined capacity scaling and local branching approach for capacitated multi-commodity network design problem." In: *Far East Journal of Applied Mathematics* 92.1 (2015), pp. 1–30.

Katayama, Chen, and Kubo (2009). "A capacity scaling heuristic for the multicommodity capacitated network design problem." In: *Journal of Computational and Applied Mathematics*. Special Issue: Honor of Professor Hideo Kawarada on the Occasion of his 70th Birthday 232.1 (Oct. 1, 2009), pp. 90–101.

Kauder and Meyr (2009). "Strategic network planning for an international automotive manufacturer." In: *OR Spectrum* 31.3 (June 1, 2009), pp. 507–532.

Kennington (1978). "A Survey of Linear Cost Multicommodity Network Flows." In: *Operations Research* 26.2 (Apr. 1, 1978), pp. 209–236.

Kia Motors Corp. (2019). *Facilities & Buildings*. Apr. 2019. URL: `https://www.kiapressoffice.com/facilitiesnbuildings` (visited on 04/24/2019).

Kim (1997). “Large scale transportation service network design : models, algorithms and applications.” PhD thesis. Massachusetts Institute of Technology, 1997.

Kim and Barnhart (1999). “Transportation Service Network Design: Models and Algorithms.” In: *Computer-Aided Transit Scheduling*. Ed. by Wilson. Lecture Notes in Economics and Mathematical Systems. Berlin, Heidelberg: Springer, 1999, pp. 259–283.

Kim, Ok, Kumara, and Yee (2010). “A market-based approach for dynamic vehicle deployment planning using radio frequency identification (RFID) information.” In: *International Journal of Production Economics*. Integrating the Global Supply Chain 128.1 (Nov. 1, 2010), pp. 235–247.

Kirkpatrick, Gelatt, and Vecchi (1983). “Optimization by Simulated Annealing.” In: *Science* 220.4598 (May 13, 1983), pp. 671–680.

Kleeman, Seibert, Lamont, Hopkinson, and Graham (2012). “Solving Multicommodity Capacitated Network Design Problems Using Multiobjective Evolutionary Algorithms.” In: *IEEE Transactions on Evolutionary Computation* 16.4 (Aug. 2012), pp. 449–471.

Kliewer and Timajev (2005). “Relax-and-Cut for Capacitated Network Design.” In: *Algorithms – ESA 2005*. European Symposium on Algorithms. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, Oct. 3, 2005, pp. 47–58.

Kohl (1995). “Exact methods for time constrained routing and related scheduling problems.” PhD thesis. Technical University of Denmark, 1995.

Köhler, Möhring, and Skutella (2009). “Traffic Networks and Flows over Time.” In: *Algorithmics of Large and Complex Networks*. Ed. by Lerner, Wagner, and Zweig. Lecture Notes in Computer Science 5515. Berlin, Heidelberg: Springer, 2009, pp. 166–196.

Kolar and Abu-Ghazaleh (2006). “A multi-commodity flow approach for globally aware routing in multi-hop wireless networks.” In: *Fourth Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM'06)*. Mar. 2006, 10 pp.–317.

Korte and Vygen (2012). *Combinatorial optimization: Theory and Algorithms*. 5th ed. Berlin, Heidelberg: Springer, 2012.

Krishnan (1998). “Design of large scale transportation service networks with consolidation: Models, algorithms and applications.” PhD thesis. Massachusetts Institute of Technology, 1998.

Lamar, Sheffi, and Powell (1987). *A lower bound for uncapacitated, multicommodity fixed charge network design problems*. Vol. 87. Irvine: University of California, Institute of Transportation Studies, 1987.

Larsson and Yuan (2004). "An Augmented Lagrangian Algorithm for Large Scale Multicommodity Routing." In: *Computational Optimization and Applications* 27.2 (Feb. 1, 2004), pp. 187–215.

Leighton, Stein, Makedon, Tardos, Plotkin, and Tragoudas (1991). "Fast Approximation Algorithms for Multicommodity Flow Problems." In: *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*. STOC '91. New York: Association for Computing Machinery (ACM), 1991, pp. 101–111.

Letchford and Souli (2019). "New valid inequalities for the fixed-charge and single-node flow polytopes." In: *Operations Research Letters* 47.5 (Sept. 1, 2019), pp. 353–357.

Lewis (2016). *A Guide to Graph Colouring: Algorithms and Applications*. Cham: Springer, 2016.

Lin (2014). "A lagrangian heuristic algorithm for an automobile distribution network optimization problem." In: *Applied Mathematics & Information Sciences* 8.6 (2014), p. 2991.

Lozano, Duque, and Medaglia (2015). "An Exact Algorithm for the Elementary Shortest Path Problem with Resource Constraints." In: *Transportation Science* 50.1 (Jan. 23, 2015), pp. 348–357.

Lozano and Medaglia (2013). "On an exact method for the constrained shortest path problem." In: *Computers & Operations Research* 40.1 (Jan. 1, 2013), pp. 378–384.

Lübbecke and Desrosiers (2005). "Selected Topics in Column Generation." In: *Operations Research* 53.6 (Dec. 1, 2005), pp. 1007–1023.

Luo and Kianfar (2018). *n-step cutset inequalities: facets for multi-module capacitated network design problem*. Feb. 11, 2018. URL: `http://www.optimization-online.org/DB_HTML/2018/11/6904.html` (visited on 01/01/2019).

Magnanti, Mirchandani, and Vachani (1991). *Modeling and Solving the Capacitated Network Loading Problem*. Tech. rep. OR 239-91. Massachusetts Institute of Technology, Operations Research Center, Jan. 1991.

Magnanti, Mirchandani, and Vachani (1993). "The convex hull of two core capacitated network design problems." In: *Mathematical Programming* 60.1 (June 1, 1993), pp. 233–250.

Magnanti, Mirchandani, and Vachani (1995). "Modeling and Solving the Two-Facility Capacitated Network Loading Problem." In: *Operations Research* 43.1 (Feb. 1, 1995), pp. 142–157.

Magnanti and Wong (1984). "Network design and transportation planning: Models and algorithms." In: *Transportation Science* 18.1 (1984), pp. 1–55.

Mamer and McBride (2000). "A Decomposition-Based Pricing Procedure for Large-Scale Linear Programs: An Application to the Linear Multicommodity Flow Problem." In: *Management Science* 46.5 (May 1, 2000), pp. 693–709.

MAN SE (2019). *Production Plants*. Apr. 2019. URL: `https://www.mantruckandbus.com/en/company/production-sites-man-truck-and-bus/production/production-plants.jsp` (visited on 04/24/2019).

Marchand, Martin, Weismantel, and Wolsey (2002). "Cutting planes in integer and mixed integer programming." In: *Discrete Applied Mathematics* 123.1 (Nov. 15, 2002), pp. 397–446.

Marshall, Boland, Savelsbergh, and Hewitt (2021). "Interval-based Dynamic Discretization Discovery for Solving the Continuous-Time Service Network Design Problem." In: *Transportation Science* 55.1 (2021), pp. 29–51.

Mattfeld (2006). *The Management of Transshipment Terminals: Decision Support for Terminal Operations in Finished Vehicle Supply Chains*. Boston: Springer, June 1, 2006. 185 pp.

Mattfeld and Kopfer (2003). "Terminal operations management in vehicle transshipment." In: *Transportation Research Part A: Policy and Practice* 37.5 (June 1, 2003), pp. 435–452.

Matuschke (2013). "Network flows and network design in theory and practice." PhD thesis. Technische Universität Berlin, Dec. 10, 2013.

Melo, Nickel, and Saldanha-da-Gama (2009). "Facility location and supply chain management – A review." In: *European Journal of Operational Research* 196.2 (July 16, 2009), pp. 401–412.

Meyr (2009). "Supply chain planning in the German automotive industry." In: *Supply Chain Planning: Quantitative Decision Support and Advanced Planning Solutions*. Ed. by Meyr and Günther. Berlin, Heidelberg: Springer, 2009, pp. 1–23.

Miller, Wise, and Clair (1996). "Transport network design and mode choice modeling for automobile distribution: A case study." In: *Location Science* 4.1 (May 1, 1996), pp. 37–48.

Munguía, Ahmed, Bader, Nemhauser, Goel, and Shao (2017). "A parallel local search framework for the Fixed-Charge Multicommodity Network Flow problem." In: *Computers & Operations Research* 77 (Jan. 1, 2017), pp. 44–57.

Muriel and Munshi (2004). "Capacitated multicommodity network flow problems with piecewise linear concave costs." In: *IIE Transactions* 36.7 (July 1, 2004), pp. 683–696.

Muter, Birbil, and Bülbül (2013). "Simultaneous column-and-row generation for large-scale linear programs with column-dependent-rows." In: *Mathematical Programming* 142.1 (Dec. 1, 2013), pp. 47–82.

Neame (2000). "Nonsmooth Dual Methods in Integer Programming." PhD thesis. University of Melbourne, 2000.

NGA (2020). *World Port Index*. Jan. 2020. URL: `https://msi.nga.mil/Publications/WPI` (visited on 01/02/2020).

Nieuwenhuis, Beresford, and Choi (2012). "Shipping or local production? CO2 impact of a strategic decision: An automotive industry case study." In: *International Journal of Production Economics*. Sustainable Development of Manufacturing and Services 140.1 (Nov. 1, 2012), pp. 138–148.

OICA (2019a). *Production Statistics 2018*. Apr. 2019. URL: `http://www.oica.net/category/production-statistics/2018-statistics/` (visited on 04/18/2019).

OICA (2019b). *Sales Statistics 2005 - 2017*. Apr. 2019. URL: `http://www.oica.net/category/sales-statistics/` (visited on 04/18/2019).

Ortega and Wolsey (2003). "A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem." In: *Networks* 41.3 (May 1, 2003), pp. 143–158.

Ouorou, Mahey, and Vial (2000). "A Survey of Algorithms for Convex Multicommodity Flow Problems." In: *Management Science* 46.1 (Jan. 1, 2000), pp. 126–147.

Ozdaglar and Bertsekas (2004). "Optimal Solution of Integer Multicommodity Flow Problems With Application in Optical Networks." In: *Frontiers in Global Optimization*. Ed. by Floudas and Pardalos. Nonconvex Optimization and Its Applications 74. Boston: Springer, 2004, pp. 411–435.

Padberg, van Roy, and Wolsey (1985). "Valid Linear Inequalities for Fixed Charge Problems." In: *Operations Research* 33.4 (Aug. 1, 1985), pp. 842–861.

Pan and Nagi (2013). "Multi-echelon supply chain network design in agile manufacturing." In: *Omega* 41.6 (Dec. 2013), pp. 969–983.

Paraskevopoulos, Bektaş, Crainic, and Potts (2016). "A cycle-based evolutionary algorithm for the fixed-charge capacitated multi-commodity network design problem." In: *European Journal of Operational Research* 253.2 (Sept. 1, 2016), pp. 265–279.

Parment (2016). *Die Zukunft des Autohandels*. Wiesbaden: Springer Fachmedien, 2016.

PassMark Software (2019). *PassMark - CPU Benchmarks*. Oct. 2019. URL: `https://www.cpubenchmark.net/` (visited on 10/01/2019).

Pedersen, Crainic, and Madsen (2008). "Models and Tabu Search Metaheuristics for Service Network Design with Asset-Balance Requirements." In: *Transportation Science* 43.2 (July 31, 2008), pp. 158–177.

Peinhardt (2003). "Integer Multicommodity Flows in Optical Networks." MA thesis. Technische Universität Berlin, Mar. 2003.

Pessoa, Sadykov, Uchoa, and Vanderbeck (2018). "Automation and Combination of Linear-Programming Based Stabilization Techniques in Column Generation." In: *INFORMS Journal on Computing* 30.2 (May 1, 2018), pp. 339–360.

Powell (1986). "A Local Improvement Heuristic for the Design of Less-than-Truckload Motor Carrier Networks." In: *Transportation Science* 20.4 (Nov. 1, 1986), pp. 246–257.

Powell and Chen (1997). "A generalized threshold algorithm for the shortest path problem with time windows." In: *Network Design: Connectivity and Facilities Location*. Vol. 40. DIMACS - Series in Discrete Mathematics and Theoretical Computer Science. Providence: American Mathematical Society, 1997, pp. 303–318.

Powell and Sheffi (1989). "Design and Implementation of an Interactive Optimization System for Network Design in the Motor Carrier Industry." In: *Operations Research* 37.1 (May 1989), pp. 12–29.

Raack (2012). "Capacitated Network Design - Multi-Commodity Flow Formulations, Cutting Planes, and Demand Uncertainty." PhD thesis. Technische Universität Berlin, June 26, 2012.

Raack, Koster, Orlowski, and Wessäly (2011). "On cut-based inequalities for capacitated network design polyhedra." In: *Networks* 57.2 (Feb. 11, 2011), pp. 141–156.

Raghavan and Tompson (1987). "Randomized rounding: A technique for provably good algorithms and algorithmic proofs." In: *Combinatorica* 7.4 (Dec. 1, 1987), pp. 365–374.

Rardin and Wolsey (1993). "Valid inequalities and projecting the multicommodity extended formulation for uncapacitated fixed charge network flow problems." In: *European Journal of Operational Research* 71.1 (Nov. 26, 1993), pp. 95–109.

Robusto (1957). "The Cosine-Haversine Formula." In: *The American Mathematical Monthly* 64.1 (1957), pp. 38–40.

Rodríguez-Martín and José Salazar-González (2010). "A local branching heuristic for the capacitated fixed-charge network design problem." In: *Computers & Operations Research*. Hybrid Metaheuristics 37.3 (Mar. 1, 2010), pp. 575–581.

Rohde, Meyr, and Wagner (2000). "Die Supply Chain Planning Matrix." In: *PPS-Management* 5.1 (2000), pp. 10–15.

Rothenbächer, Drexl, and Irnich (2016). "Branch-and-price-and-cut for a service network design and hub location problem." In: *European Journal of Operational Research* 255.3 (Dec. 16, 2016), pp. 935–947.

Sadykov (2019). "Modern Branch-Cut-and-Price." Habilitation thesis. Université de Bordeaux, 2019.

Sadykov, Uchoa, and Pessoa (2017). *A Bucket Graph Based Labeling Algorithm with Application to Vehicle Routing*. Tech. rep. L-2017-7. Niterói: Universidade Federal Fluminense, LOGIS, Oct. 2017, p. 45.

Sadykov and Vanderbeck (2011). "Column Generation for Extended Formulations." In: *Electronic Notes in Discrete Mathematics*. LAGOS'11 – VI Latin-American Algorithms, Graphs and Optimization Symposium 37 (Supplement C Aug. 1, 2011), pp. 357–362.

Salimifard and Bigharaz (2020). "The multicommodity network flow problem: state of the art classification, applications, and solution methods." In: *Operational Research* (Apr. 23, 2020).

Schrijver (1998). *Theory of Linear and Integer Programming*. John Wiley & Sons, July 7, 1998. 488 pp.

Schrijver (2003). *Combinatorial optimization*. Red. by Cook, Korte, Lovász, Wigderson, and Ziegler. Algorithms and Combinatorics 24. Berlin, Heidelberg: Springer, 2003.

Schulz (2014). *Logistikintegrierte Produktentwicklung: Eine zukunftsorientierte Analyse am Beispiel der Automobilindustrie*. Wiesbaden: Springer Gabler, 2014.

Sedeño-Noda and Alonso-Rodríguez (2015). "An enhanced K-SP algorithm with pruning strategies to solve the constrained shortest path problem." In: *Applied Mathematics and Computation* 265 (Aug. 15, 2015), pp. 602–618.

Sellmann, Kliewer, and Stein (2002). "Lagrangian Cardinality Cuts and Variable Fixing for Capacitated Network Design." In: *Algorithms – ESA 2002*. European Symposium on Algorithms. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, Sept. 17, 2002, pp. 845–858.

Skutella (2009). "An Introduction to Network Flows over Time." In: *Research Trends in Combinatorial Optimization*. Ed. by Cook, Lovász, and Vygen. Berlin, Heidelberg: Springer, 2009, pp. 451–482.

Srinivasan (2001). "Distributions on level-sets with applications to approximation algorithms." In: *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*. Oct. 2001, pp. 588–597.

Stadtler (2003). "Multilevel Lot Sizing with Setup Times and Multiple Constrained Resources: Internally Rolling Schedules with Lot-Sizing Windows." In: *Operations Research* 51.3 (June 2003), pp. 487–502.

Stadtler (2005). "Supply chain management and advanced planning—basics, overview and challenges." In: *European Journal of Operational Research*. Supply Chain Management and Advanced Planning 163.3 (June 16, 2005), pp. 575–588.

Stadtler, Kilger, and Meyr, eds. (2015). *Supply Chain Management and Advanced Planning*. Springer Texts in Business and Economics. Berlin, Heidelberg: Springer, 2015.

SteadieSeifi, Dellaert, Nuijten, van Woensel, and Raoufi (2014). "Multimodal freight transportation planning: A literature review." In: *European Journal of Operational Research* 233.1 (Feb. 16, 2014), pp. 1–15.

Sun (2018). "Heuristic Algorithms for Graph Coloring Problems." PhD thesis. Université d'Angers, Nov. 29, 2018.

Suthikarnnarunai (2008). "Automotive supply chain and logistics management." In: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. Vol. 2. 2008, pp. 1–7.

Teypaz, Schrenk, and Cung (2010). "A decomposition scheme for large-scale Service Network Design with asset management." In: *Transportation Research Part E: Logistics and Transportation Review* 46.1 (Jan. 2010), pp. 156–170.

Thomas, Calogiuri, and Hewitt (2019). "An exact bidirectional A* approach for solving resource-constrained shortest path problems." In: *Networks* 73.2 (2019), pp. 187–205.

Toth and Vigo (2014). *Vehicle routing: problems, methods, and applications*. Society for Industrial and Applied Mathematics (SIAM), 2014.

Toyota Motor Corp. (2018). *Facilities*. Dec. 2018. URL: `https://global.toyota/en/company/profile/facilities/` (visited on 04/19/2019).

Toyota Motor Corp. (2021). *Financial Summary*. FY2021. May 12, 2021.

Vanderbeck (1994). "Decomposition and column generation for integer programs." PhD thesis. Université Catholique de Louvain, 1994.

Vanderbeck (2005). "Implementing Mixed Integer Column Generation." In: *Column Generation*. Ed. by Desaulniers, Desrosiers, and Solomon. Boston: Springer, 2005, pp. 331–358.

VDA (2019). *Monthly figures*. Dec. 2019. URL: `https://www.vda.de/en/services/facts-and-figures/monthly-figures` (visited on 12/16/2019).

Vecchiato (2012). "Lean Distribution." PhD thesis. University of Padua, Jan. 29, 2012.

Volkswagen Group (2019). *Portrait & Production Plants*. Apr. 2019. URL: https://www.volkswagenag.com/en/group/portrait-and-production-plants.html (visited on 04/24/2019).

Volkswagen Group (2021). *Volkswagen Group strengthens market position in 2020 and hits the ground running in e-offensive*. Volkswagen Newsroom. Jan. 13, 2021. URL: https://www.volkswagen-newsroom.com/en/press-releases/volkswagen-group-strengthens-market-position-in-2020-and-hits-the-ground-running-in-e-offensive-6752 (visited on 04/24/2019).

Vu, Crainic, and Toulouse (2013). "A three-phase matheuristic for capacitated multi-commodity fixed-cost network design with design-balance constraints." In: *Journal of Heuristics* 19.5 (Oct. 1, 2013), pp. 757–795.

Wentges (1997). "Weighted Dantzig-Wolfe decomposition for linear mixed-integer programming." In: *International Transactions in Operational Research* 4.2 (Mar. 1, 1997), pp. 151–162.

Wieberneit (2008a). "Linienverkehrsplanung in speditionellen Sammelgutnetzen." PhD thesis. Universität Augsburg, 2008.

Wieberneit (2008b). "Service network design for freight transportation: a review." In: *OR Spectrum* 30.1 (Jan. 1, 2008), pp. 77–112.

Wollmer (1971). "Multicommodity networks with resource constraints: The generalized multicommodity flow problem." In: *Networks* 1.3 (Jan. 1, 1971), pp. 245–263.

Wolsey and Nemhauser (1999). *Integer and Combinatorial Optimization*. John Wiley & Sons, July 22, 1999. 784 pp.

Yaghini, Rahbar, and Karimi (2013). "A hybrid simulated annealing and column generation approach for capacitated multicommodity network design." In: *Journal of the Operational Research Society* 64.7 (July 1, 2013), pp. 1010–1020.

Yanagisawa (2006). *Fast Shortest Path Computation for Solving the Multicommodity Flow Problem*. Research rep. RT0688. IBM Research, Tokyo Research Laboratory, 2006.

Zeng, Hu, and Huang (2013). "The Transportation Mode Distribution of Multimodal Transportation in Automotive Logistics." In: *Procedia - Social and Behavioral Sciences*. Intelligent and Integrated Sustainable Multimodal Transportation Systems: Proceedings from the 13th COTA International Conference of Transportation Professionals (CICTP2013) 96 (Nov. 6, 2013), pp. 405–417.

Zhu, Crainic, and Gendreau (2014). "Scheduled Service Network Design for Freight Rail Transportation." In: *Operations Research* 62.2 (Apr. 1, 2014), pp. 383–400.

Zhu and Wilhelm (2013). "Implementation of a three-stage approach for the dynamic resource-constrained shortest-path sub-problem in branch-and-price." In: *Computers & Operations Research* 40.1 (Jan. 1, 2013), pp. 385–394.

Ziegelmann (2001). "Constrained shortest paths and related problems." PhD thesis. Universität des Saarlandes, 2001.

Zuse Institute (2018). *MIPLIB 2017*. Nov. 5, 2018. URL: `http://miplib.zib.de` (visited on 08/03/2019).

Zwillinger (2018). *CRC Standard Mathematical Tables and Formulas*. 33rd ed. Taylor & Francis, Jan. 19, 2018. 872 pp.