A Generalized Network Model for Freight Car Distribution

Inaugural-Dissertation
zur
Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen
Fakultät
der Universität zu Köln

vorgelegt von
Birgit Engels
aus Mechernich

Mai 2011

Berichterstatter:	Prof. Dr. Rainer Schrader
	Prof. Dr. Sven O. Krumke
Tag der mündlichen Prüfung:	23.05.2011

Zusammenfassung

Wir betrachten das Problem der Disposition leerer Güterwagen (DP) bei DB Schenker Rail Deutschland AG unter Berücksichtigung vieler praxisrelevanter Nebenbedingungen und realer Daten(mengen). Das (DP) ist ein „Online"- Zuordnungsproblem zwischen dezentral verfügbaren Güterwagen (Beständen) und geographisch verteilten Kundenbestellungen (Bedarfen) für zukünftige Gütertransporte. Eine optimale Zuordnung minimiert im Wesentlichen die Transportkosten für leere Güterwagen.

Eine grundlegende Nebenbedingung ist die rechtzeitige Erreichbarkeit zwischen Bestand und Bedarf, die durch einen vorgegebenen Güterzug-Fahrplan bestimmt wird. Verschiedene Güter (wie zum Beispiel Schüttgut, Stahl-Coils, etc.) benötigen unterschiedliche Arten von Güterwagen für ihren Transport. Bestände und Bedarfe sind daher typisiert. Andererseits können verschiedene Wagentypen (nach festen Schemata und Anzahlen) gegeneinander ausgetauscht werden. Diese erlaubten Austauschbeziehungen geben eine weitere zentrale Nebenbedingung des (DP) vor.

Wir stellen in der vorliegenden Arbeit zahlreiche weitere „harte" und „weiche" Nebenbedingungen vor und skizzieren die zur Zeit durchgeführte Disposition bei DB Schenker Rail Deutschland AG. Diese wird nach Gruppen von Wagentypen getrennt und in mehreren, teils manuellen vor- bzw. nachgelagerten Schritten vorgenommen. So kann es zu global suboptimalen Dispositionen kommen. Um dies unter Berücksichtigung aller Nebenbedingungen zu vermeiden, modellieren wir das (DP) als generalisiertes Flussnetzwerk, welches ganzzahlig zu lösen ist. Letzteres ist NP-schwer, ebenso wie das (DP) unter allgemeinen Substitutionsbedingungen. Die Lösung des (DP) erfordert daher einen Kompromiss zwischen praktikabler Laufzeit und Qualität der erzeugten Disposition.

Wir erreichen dies einerseits durch Anwendung approximativer und heuristischer Lösungsmethoden für den Fall allgemeiner Substitutionsbedingungen, andererseits durch die Entwicklung einer netzwerkbasierten Reoptimierungsstrategie. Diese berechnet für eine Folge von Instanzen mit wenigen Datenänderungen in kurzer Zeit optimale (bzw. im allgemeinen Fall approximative und heuristische) zulässige Lösungen.

Diese Arbeit wurde durch ein 2-jähriges Forschungs- und Entwicklungsprojekt der DB Schenker Rail Deutschland AG in Kooperation mit der Arbeitsgruppe Faigle/Schrader der Universität zu Köln und mit der Arbeitsgruppe von Prof. Dr. Sven O. Krumke (Technische Universität Kaiser-

slautern) motiviert und finanziert. Das Projekt umfasste ebenfalls die Implementierung des entwickelten generalisierten Netzwerkmodells, der Reoptimierungsstrategie sowie der Approximationsmethoden und Heuristiken. Diese sollen als zukünftiger Optimierungskern für das (DP) bei DB Schenker Rail Deutschland AG genutzt werden.

Abstract

We consider the empty freight car distribution problem (DP) at DB Schenker Rail Deutschland AG under a wide range of application relevant constraints and real data sets. The (DP) is an online assignment problem between geographically distributed empty freight car supplies and customer demands for such cars in preparation of good transport. The objective is to minimize transport costs for empty cars while distributing them effectively with respect to the constraints.

In our case, one major constraint is given by prescheduled freight trains: obviously a supply can only be assigned to a demand if it reaches the latter in time. Further, the variety of goods (bulk cargo, steel coils, etc.) to be transported requires distinct types of freight cars. Freight cars of a certain type can be exchanged by cars of other types with respect to a given substitution scheme and different 'exchange rates'. Allowed substitutions are therefore another major constraint of the (DP).

We describe further 'hard' and 'soft' constraints and sketch the current work flow at DB Schenker Rail Deutschland AG to find an adequate solution for the (DP) on a daily base in practice. The (DP) is currently solved separately for groups of car types and in several steps. Moreover, some steps contain manual pre- and post-processing to ensure certain constraints. Hence global sub-optimal distributions can occur. We therefore integrate all constraints into a generalized network flow model for the (DP). A global optimal distribution is then provided by an integral minimum cost flow in the network. To find such a flow is NP-hard in general. We show that a general substitution scheme makes our notion of the (DP) also NP-hard. Hence independent of the applied model and with respect to practical runtime requirements, we have to find a compromise between solution time and quality .

We do so in two ways. Instances of the (DP) which correspond to classical flow networks are solved by an integral minimum cost flow, which can be obtained in polynomial time. We use such instances to polynomially obtain minimum cost flows of fixed bounded fractionality for certain general instances. For those instances occurring in the application we obtain half-integral flows, which can be rounded to approximate or heuristic distributions in linear time. Moreover, we develop a network-based reoptimization approach, which yields optimal solutions for subsequent instances with few changes very fast.

This thesis was inspired and funded by a 2-year research and development project of DB Schenker Rail Deutschland AG in cooperation with the work group Faigle/Schrader of the University of Cologne and the work group of Prof. Dr. Sven O. Krumke at the Technical University of Kaiserslautern. The project included the implementation of the generalized network model and the reoptimization, approximation and heuristic methods. The software is designed as a future optimization kernel for the (DP) at DB Schenker Rail Deutschland AG.

Front Credits - Der Vorspann - Danke

DIRECTED BY...

Als erstes möchte ich mich herzlich bei Herrn Prof. Schrader bedanken, dass er meine Promotion mit seiner Wertschätzung selbständigen Arbeitens („Tschöh, tschöh, tschüss...") und dem Stellen der richtigen Fragen („...ja aber, Frau Engels, ...") betreute.

CAST

Für die Ermöglichung der Projekt-Zusammenarbeit und damit einer spannenden Promotion vor realem Anwendungshintergrund, danke ich Herrn Prof. Schrader, Herrn Prof. Krumke und insbesondere unserer Bahnverbindungsfrau Katja Wolf (DB Schenker Railion Deutschland AG).

DB CREW

Mein Dank gilt ebenfalls Sascha Fichtner, Ralph Winkler, Mario Steinberg sowie Herrn Elias Dahlhaus und Tim Scheipers und allen weiteren Projektteilnehmern. Danke für die „handfeste" Verbessserung meiner Geographie-Kenntnisse („Wo liegt Sassnitz-Mukran?"), Herr Röhrig!

KAISERSLAUTERN CREW

Ich danke Herrn Prof. Krumke für die Übernahme des Zweitgutachtens von „down under" aus und wertvolle Diskussionen bei leckerem Grillgut. Danke an die „KL-Mädels" Katharina (Kathi) Beygang und Christiane Zeck für die Zusammenarbeit u.a. in der Schlacht gegen das Handbuch, bei der Instanzgenerierung und beim Testen, Testen, Testen...

COLOGNE CREW

Ich bedanke mich bei Herrn Prof. Schrader und Herrn Prof. Faigle, dass ich in Ihrer Arbeitsgruppe Forschung, Lehre und Projektarbeit leisten konnte und für alles was ich dabei gelernt habe. Eine Arbeitsgruppe, wäre natürlich keine Arbeitsgruppe ohne Kollegen, bei denen ich mich ebenfalls bedanken möchte. Besonders danke ich dabei Anna Schulze für die tolle Zusammenarbeit und Unterstützung in den Anfängen, Dominik Andres und Bernhard Fuchs, die (meine) erste DG-Generation komplettierten. Danke an Oliver Schaudt und Vera „DAMNet" Weil für das Gegenlesen dieser Arbeit und das gemeinsame Kiosken, an Jan Voss für die Frischluft-Pausen und besonderen Dank an Daniel Herrmann, DG – The Next Generation.

MY CREW

Ich danke meinen Eltern Anna-Maria und Hans-Karl Engels für die Ermöglichung meines Studiums in jeder Hinsicht und meinen Schwestern Stefanie und Silke Engels, die immer Ansprechpartner für mich sind. Meinem Freund Gregor Pardella danke ich für seine geduldige Unterstützung, die neuen Blickwinkel, die er mir aufgezeigt hat und vor allem danke ich ihm für seine Liebe, ohne die nichts wichtig wäre.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

The general concern of any cargo (railway) company is obviously the transport of goods between different customer sites and (air)ports for further transport. With regard to railway transport, goods must be loaded into empty freight cars at the initial location before this actual customer service takes place and can then be unloaded at the transport's destination. Mostly, the respective freight cars are owned by the operating railway company and not by the customers themselves.

Thus apart from the transport of loaded freight cars, a railway company also manages the subsequent rental of its freight car stock. Although the car rent might be included in the total transport costs, there are several reasons to distinguish between freight car allocation and transport service. On the one hand, loading and unloading the cars can take significant time compared to the transport time, especially with regard to 'just-in-time' production. Thus the car rent is also due for time periods without active transport.

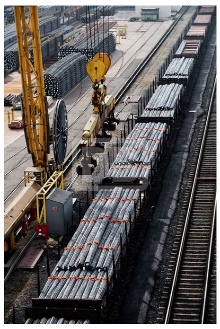
On the other hand, the allocation of empty freight cars for (transport) demands (and thereby the possibility to satisfy a demand) is a logistic problem, which rather depends on previous than upcoming transports. Of the complete stock of empty cars, only parts are temporarily available, as the rest is bound in current transports. Moreover, available cars are returned by customers to different sites or stored decentralized and the company cannot charge the transport cost for the empty cars to the upcoming customer demands. Thus the logistic goal is to assign all available empty freight cars to customer demands (or storage) with minimum total transport cost. We call this optimization task the *distribution problem (DP)*.

This thesis models and solves the (DP) as a real world application with respect to specific requirements for DB Schenker Rail Deutschland AG including a stand-alone software implementation (see Appendix A for an overview over the implementation and experimental results). It was funded by a two year's research and development project of DB Schenker Rail Deutschland AG[1] in cooperation with the work group Faigle/Schrader at the University of Cologne and the group of Prof. Dr. Sven O. Krumke at the Technical University of Kaiserslautern.[2]

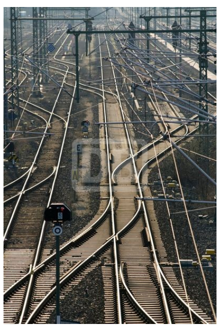
1 DB Schenker Rail Deutschland AG, Rheinstrasse 2, 55116 Mainz
2 AG Mathematische Optimierung, Technische Universität Kaiserslautern, Paul-Ehrlich-Str. 14, 67663 Kaiserslautern

(a) Franfurt, control center (DB4788) by Mario Vedder

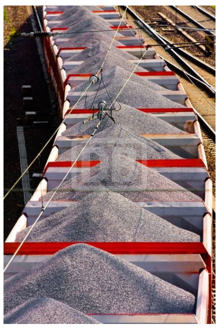
(b) Maschen, shunting yard (DB13244) by Volker Emersleben



(c) (DB28270) by Michael Neuhaus



(d) (DB9012) by Günter Jazbec



(e) (DB13010) by Bartlomiej Banaszak

(f) (DB4383) by Michael Neuhaus

Figure 1: Freight car distribution is a complex real-world process, where optimization and automatizing meet expert knowledge and man power. All pictures taken from https://mediathek.deutschebahn.com by courtesy of Deutsche Bahn AG.

The DB Schenker Rail Deutschland AG is one of the first cargo railway companies on the European continent measured for example by business volume (3.791 million Euros in 2009), transported tonnage (341 million tons in 2009) by the nearly 5000 operated trains per day resulting in about 10^5 million ton kilometres.[3] Planning and transport is carried out by more than 34 thousand employees managing a stock of 10^5 freight cars owned by DB Schenker Rail Deutschland AG and a considerable number of cars leased or temporarily distributed for other wagon keepers. A railway network of about 36 thousand kilometers interconnects about 4200 customer locations throughout Germany, Denmark, Italy, Switzerland, the Netherlands and other European countries with about 50 shunting facilities. The workload of the company's about 3500 locomotives spares highway traffic 10^5 trucks per working day.

At DB Schenker Rail Deutschland AG the (DP) has to be solved for 10^4 to 10^5 available empty freight cars and demands for single up to a hundred cars per day. As mentioned above, this also includes cars which are not permanently owned by the company, but currently taken care of in the distribution process. The complete stock is temporarily rented to customers for various transportation uses. For this period of time, a certain freight car drops out of our optimization world. As soon as it returns from a customer – mostly at other places than those it was rented – it becomes current supply and can be assigned to another customer demand again.

This short description already contains the basic variables and constraints which are to be met: *supplies* and *demands* of empty freight cars are connected to their *location*, the *time* they become available respectively need to be satisfied and to different *types* of cars due to their different transportation requirements like the kind of goods to be transported. Based on these and additional attributes, a number of constraints are imposed on the minimum cost solution.

The major constraints considered from our view on the (DP) are for supplies to be in time and of appropriate type for the demands they are assigned to. The former constraint depends on a predefined freight train schedule and the latter is defined by a so-called substitution scheme.

In the remainder of this introduction, we give a short sketch of the daily distributive process at DB Schenker Rail Deutschland AG in Section 1.1 and present a summary of research and optimization goals of the joint project in Section 1.2. A brief survey of the literature on related topics is given in Section 1.3. We close the introduction by an outline of this thesis in Section 1.4.

3 From: http://www.rail.dbschenker.de.

1.1 DISTRIBUTION AT DB SCHENKER RAIL DEUTSCHLAND AG

Today at DB Schenker Rail Deutschland AG the (DP) is solved once a day. The solution relies on a variety of computer programs for data acquisition and management in different planning and production stages with various interfaces and parameters. Handling and coordination of those devices depend on a considerable amount of manpower.

In the morning, system based estimates for numbers of available cars are consolidated with actual status messages from different yard locations. New demand information is partly gathered via different Internet portals addressing different ranges of customers and thus different groups of car types, but also via traditional communication channels such as telephone or fax. All variants, however, require manual processing before entering the optimization.

At a fixed time in the morning the (global) optimization process is started. To reduce running time and memory requirements, supply and demand data is split according to the seven main car type groups and seven runs of the optimization program are started, each of which takes about one minute. Before the optimization is started, fleet managers manually correlate the absolute supply and demand values with respect to type groups. A set of parameters allows to shorten demands regarding different priorities in case of low supply. This regulation is expected to include a cost independent fairness with regard to customer satisfaction.

The central application used to solve the above sketched optimization problem is an implementation of the *Out of Kilter* algorithm [92, 41, 2]. This algorithm already bases on a network model of the distribution problem designed by F. Schläpfer in the 1970s for the Swiss Federal Railway [49, 108, 109].

Generally, supplies are registered at *dispo stations* (frequently coinciding with shunting facilities) located throughout the German railroad network. Further, available cars are clustered with respect to types and availability time with the help of so called phases (time intervals). The same scheme is applied to demands with the difference that also customers in the central European countries are considered. It follows that the optimal distribution based on the clustered and thus reduced input data does not provide the individual assignments. This final assignment (often composed of two and up to three partial assignments) is done manually as a post-processing to the aggregated optimization. So far, supplies which are currently not allocated for demands and demands, which are not provided the number of ordered cars are administered manually as well as the storage capacities and the return of cars to other wagon keepers.

1.2 PROJECT AND THESIS GOALS

The research and development project of the DB Schenker Rail Deutschland AG was embedded in the fleet management project, which amongst others aims at the following topics. On the one hand, processes are planned to be more integrated and automated (but still manually controllable). On the other hand more flexibility in terms of data structure and volume as well as the work flow over time is targeted.

In this context the optimization kernel for the (DP) has to be adopted. With regard to integration and automatizing, for example the administration of operative storage capacities is now part of the optimization process as well as the retransfer of cars which do not belong to the DB Schenker Rail Deutschland AG stock. As these aspects concern all car types in a concurrent way, also the classification in type groups and the separate optimization runs have to be abandoned.

Further, the classical (threefold) assignment of cars as explained above can be replaced by a single assignment. This development is partly due to a number of additional attributes which makes formerly used clustering of supply and demand at dispo stations non-beneficial in terms of reducing the number of input data records. A single assignment is expected to decrease the wagon cycle time in the operation.

Both of these aspects may lead to a larger data volume for a single optimization run and thus demanded a more efficient implementation. A modular optimization software in the object oriented $C++$ programming language achieves this as well as the required flexibility and maintainability. New standards like another car type classification that is currently under consideration can then easily be prepared and later be included in the implementation.

The main target of flexibilizing is the up to now relatively static daily work flow over time, which does not adapt to new or changed data once the optimization is started. Regarding this aspect, we develop a reoptimization strategy. The latter uses a former solution and additional information to produce a new solution after data changes. For a reasonable number of changes (less than 20% of the complete data volume, see Appendix A.2), the latter is much faster than a new optimization 'from scratch' on the whole current data volume. This performance gain allows for multiple runs of the optimization process a day, while neither blocking the work flow with long waiting times nor leaving optimization marges unused.

The reoptimization also exploits a certain time interval between the time a supply becomes known to the system and the latest possible time the supply can be assigned a distribution without intermediate storage becoming necessary. A comparable interval exists for a demand, between it becomes known to the system and the latest time it can possibly be satisfied. The interval end depends on the assigned car supply (if any). We call those time intervals *planning intervals*.

The set of optimal assignments with respect to the given input data is therefore regarded as *tentative*, as long as both supply and demand concerned are still within their planning interval with respect to the next scheduled (re)optimization. (Note that the latter convention does not permit 'reoptimization on demand', without scheduled runs unless the end of each planning interval triggers a reoptimization run.) Thus assignments can be rearranged to save transport costs taking upcoming or cancelled supplies and demands into account.

As mentioned above, one of the major constraints to the (DP) is the substitution of a demanded car by assigning a supply of another type respecting the substitution scheme. If this scheme involves changes in the number of supplied cars with respect to the number of demanded cars, finding an optimal distribution becomes a computationally difficult problem. Hence, providing an optimal solution to the (DP) in such cases is generally too slow for successful application in practice, which includes a certain percentage of substitutions with number changes.

Currently, with respect to this special substitutions, the optimization follows a greedy approach integrated into the *Out of Kilter* algorithm which is rather not well-understood for the current model. Instead, we also employ the reoptimization as part of a heuristic solution for those (DP) instances and we investigate the trade-off between running time benefits and potential decrease in the solution quality in detail (see Appendix A.2).

The developed model and solution approaches to the (DP) are summarized into a prototypical stand-alone C++ implementation (see Appendix A.1).

1.3 RELATED TOPICS

From a broad perspective, there is a large number of optimization problems in context of 'rail industry'. They are addressed in the literature beginning in the 1960s up to now. Surveys with emphasis on different aspects are for example given in [94, 3, 23, 25, 28]. Following the recent survey of Newman [94] and others we classify the respective questions as strategic (long-term), tactical (mid- and short-term) and operational (short-term). Owing to the strong interdependence of the resulting optimization problems, we also sketch long- and mid-term applications, although our own focus is operational.

Strategic aspects are for example yard location (including the analysis of the impact of closing a yard) [3] and also rail installation and maintenance. While such strategic considerations require a system-wide view with respect to a long time horizon due to installation times and large investments, their large scale and integrated nature prevent detailed modelling of all aspects. The latter concern for example train routing and scheduling, which are of course influenced by the underlying network structure and the shunting capabilities of its yards. Installing a basic train service on a network to enable the routing of cars also requires to solve the corresponding scheduling problem (for example [16, 17]) together with locomotive (for example [5]) and crew scheduling for the trains to be operated.

Each individual problem being computationally hard and large scale, they are rather solved separately or iteratively, as in the above cited articles. Yet, those problems are interdependent: each scheduled train requires an engine of appropriate pulling-power and a crew association which meets all technical and contractual restrictions. Regarding the equipment inventory and crew management also an 'economies of scale' aspect comes into play [77, 116]. Subsequent assignment problems can thus render a preliminary train schedule (economically) infeasible, such that it needs to be rearranged. In turn, potential solutions to following questions like the (railroad) blocking problem [6, 95] depend on the train schedule.

The blocking problem occurs in the operational transport for empty as well as loaded freight cars: a freight car is not treated as a single entity, but assigned to a so called 'block', which means a group of cars handled together between a certain source and destination and we ask for an assignment of cars to (not necessarily predefined) blocks generating the minimum transportation and handling cost. The definition of blocks and their assignment to trains (referred to as make-up) must enable a transport between all given origin-destination pairs. This is often achieved by designing a set of itineraries, which means paths which have to be followed by each car from its origin to the given destination. A similar construct named 'Leitweg' is used by the DB Schenker Rail Deutschland AG for routing. For a survey focusing on routing and scheduling, we refer to [23].

Recently some dissertations also address integrated models for some of the above mentioned problems, for example [122, 87]. Not surprisingly the corresponding network design problems and mixed integer formulations are large scale and hard to solve in practice even at tactical time scales without real-time requirements. Thus (like already most of the individual problems) they are solved heuristically, for example by (very large scale) neighbourhood- [87], tabu-, ellipsoidal search [122] and hybrid approaches. The usage of different decomposition techniques, simulated annealing and neural networks is also mentioned in [23].

The above sketched problems are mostly modelled as variants of the network flow and network design problem respectively, where the latter emerges roughly speaking as a consequence of integrated routing and timetabling (including for example trip planning (itineraries), design of blocking policies). A detailed review of the application of network models to many optimization problems occurring in the railroad industry is given in [3]. Integrality requirements on those models generally lead to mixed integer program formulations which are either solved with the help of (commercial) LP-solvers, branch-and-bound/-cut, decomposition and (lagrangian) relaxation approaches or heuristically as mentioned above. We will return to the use of proprietary LP-solvers later on.

Note that in some of the more closely related work to the (DP), which is briefly reviewed in the following, 'empty freight car distribution' is also referred to as 'equipment distribution' and 'repositioning' or 'fleet management' (in a more general context).

One of the first approaches in literature to the distribution of empty freight cars as an optimization problem was a time-space expanded network model by White [120]. Following the early survey on 'Models for Rail Transportation' by Assad [9] the resulting minimum cost flow problem was solved via linear programming methods. Given that White considered a homogeneous fleet, which means only one car type, the relaxation of the resulting classical flow problem provided the necessary integral solutions. Assad also mentions the applicability of combinatorial algorithms in this case. For heterogeneous fleets of several car types with possible substitution, a multi-commodity flow problem was proposed to be solved. Further, Assad reports on the acceptance of the linear programming approach and its early use as 'part of the analytical arsenal of many railroads', for example at the Swiss Federal Railways [67].

Haghani [65] claims to be the first to develop an integrated (time-space expanded network) model for scheduling, make-up, loaded car routing and empty car distribution, after reviewing existent models in [64]. Around the same time Keaton [81] also addresses timetabling, blocking and car routing simultaneously. Both problems are formulated as mixed integer programs and solved by heuristic (decomposition) approaches, where Keaton relaxes his formulation with regard to train capacities and heuristically repairs capacity violations afterwards.

From these early integrated approaches we already see that the large-scale and computationally complex nature of the combined problems seems to prevent optimal solutions to be obtained fast enough, at least at operational level, although computation power has of course improved significantly since then. However, it partly explains why CSX, one of the major US railway companies, used a decision-support tool, which was based on a simpler single-commodity minimum cost network flow formulation for the empty freight car distribution by Turnquist and Markowicz [117] between 1990 and 1996 [63]. Then in 1997, CSX implemented the first real-time, fully integrated optimization system (DCP) for equipment distribution [62] still based upon a classical single-commodity network model. Later similar systems have been implemented for the BNSF Railway [62] and the Union Pacific Railway [93]. The latter uses a classical transportation model.

Nevertheless, also multi-commodity network models are employed on an operational (for example for the Swedish Railways [71, 77]) and tactical level (for example Canadian Pacific Railway [72]).

Beside the integrated approach, another branch of development is the inclusion of stochastic data to model uncertainty of supply and demand forecasts. This is important for the development of general repositioning policies for empty cars (at a tactical level) as well as to cope with the online nature of their distribution at operational level. For example Powell and Carvalho apply stochastic modelling to a single-commodity flow approach with respect to freight car [101] and container management [102]. In both cases multiple successive assignments can be considered due to a predicted availability time of cars/containers after previous assignments. The stochastic approach is also extended to a multi-commodity flow model by Topaloglu and Powell [113]. In terms of forecasts, the latter provides the basis for a (heuristic) decision support system for locomotive scheduling implemented at Norfolk Southern Railroad [63, 94]. A survey of 'dynamic and stochastic models for the allocation of empty containers' is given by Crainic et. al. [25].

As we learn from [62], the dynamic car-planning system (DCP) of CSX relies on a minimum cost flow model alike our own considerations. For the (DCP) a linear program (LP) formulation is established and a proprietary (LP) solver is employed. Although this method yields fast solutions for large instances in general, we selected a combinatorial approach for two main reasons. Firstly, one of the project specification was to develop a prototype as a stand-alone software. The latter was designed to be embedded into the DB Schenker Rail Deutschland AG software landscape without causing license cost. The prototype was successfully embedded in a DB Schenker Rail Deutschland AG test environment at the end of the project. Secondly, the combinatorial solution approach enabled us to chose from various existing flow algorithms and fit our choice specifically for the needs of instances of the minimum cost flow problem, which arise from (DP). Examples are the strong prioritization of demands and the reoptimization, which are both achieved by modification of the selected *Successive Shortest Path* algorithm [2].

A proprietary (LP) solver can also be fine-tuned to the application, but thorough customization usually comes at high expenses. Moreover, especially in case of strong prioritization, it is not at all clear how to incorporate certain constrains in a standard (LP) model.

With regard to the amount of existing literature on the empty freight car distribution and closely related, interacting problems including some approaches that may seem similar to our own, we conclude our brief survey with a citation out of [94]: 'Martland and Sussman [89] point out that even fairly realistic models must be customized to suit a railroad's operating policies and user capabilities.' The model we develop in this thesis (see 1.4) is therefore tailored on the one hand to specific assumptions, which we worked out with DB Schenker Rail Deutschland AG (see 2) and on the other hand designed to aim at the major project goals (see 1.2).

1.4 THESIS OUTLINE

We describe the (DP) in detail in Chapter 2 from the application's point of view and derive formal definitions in Chapter 4 after introducing general terms and notation in Chapter 3. We also prove that the decision variant of (DP) is NP-complete with respect to a practice relevant constraint set in Chapter 4. The latter result is due to substitution schemes with number changes.

In the Chapters 5 and 7 we then model the (DP) as a network flow and a generalized network flow respectively such that we face increasing computational difficulty with respect to the required integrality of minimum cost flow solutions. Both models were developed in cooperation with DB Schenker Rail Deutschland AG, Prof. Dr. Rainer Schrader[4] and the group of Prof. Dr. Sven O. Krumke.[5]

An integral minimum cost flow [39, 2] in the classic network model has a one-to-one correspondence with an optimal distribution for (DP) instances where only substitutions without resulting changes in numbers of cars are allowed (homogeneous, 1-to-1-substitution). For substitutions which alter the number of required cars, like the (heterogeneous) 2-to-1-substitution occurring in the application, we need to solve a generalized minimum cost flow problem [75, 2] in integers. The latter being an NP-complete problem, its use is justified by the (DP)'s own proved complexity. Further, we characterize instances of the (DP) which allow to obtain a generalized minimum cost flow of bounded fractionality in polynomial time. This way we always obtain a half-integral flow in network models for (DP) instances from the application. (Some further theoretical results concerning special generalized flow networks in connection with known literature are postponed to Appendix B.)

In Chapters 5 and 7 we give an overview over known (generalized) minimum cost flow algorithms and develop a generalized variant for the *Successive Shortest Path* algorithm [76, 73, 15, 2]. We then use a modified *Successive Shortest Path* algorithm to solve the minimum cost flow case respecting strong prioritization.

4 AG Faigle/Schrader, Universität Köln, Weyertal 80, 50931 Köln
5 AG Mathematische Optimierung, Technische Universität Kaiserslautern, Paul-Ehrlich-Str. 14, 67663 Kaiserslautern

Dynamization of the distribution process as a basic project goal is addressed in Chapter 6. With regard to the online nature of the distribution problem (short-term changes in information/data), we present a reoptimization process on the basis of a formerly optimal flow and its associated residual network. The presentation of the reoptimization approach bases on the classical network flow model, but due to the generalization of the *Successive Shortest Path* algorithm it is also applicable to the generalized version.

So far, for the (DP) with general substitution we only obtained a solution to the relaxed problem, which means a fractional flow, in polynomial time. We dedicate Chapter 8 to finding adequate solutions in practice. Here, we have to cope with a trade-off between running time and solution quality in terms of cost. Based on rounding half-integral flows, we find optimal solutions for so called 0.5-upgraded (DP) instances associated with the original instances. We further establish a classical 4-approximation with presumptions on the (DP) instances, where single spare cars are redistributed with the help of our reoptimization approach from Chapter 6. Rounding and redistribution are then combined to yield a heuristic which is applicable to find feasible solutions for all (DP) instances in practice (see Appendix A).

Considerations to improve the approximation factor and/or relax the presumption on (DP) instances to obtain a fix factor approximation leads to a matching problem with a graph theoretic constraint, which is introduced in Chapter 9. As checking the constraint is $\mathcal{NP}$-complete for general graphs, we concentrate on such instances where the constraint can be represented by a tree. The latter leads to a polynomial algorithm employing dynamic programming, which can be applied to extend the set of (DP) instances with guaranteed 4-approximation.

2 FREIGHT CAR DISTRIBUTION

In this chapter we give an informal description of the freight car distribution problem (DP). The basic task of freight car distribution with respect to wagon load traffic is the assignment of currently available empty cars or *supplies* to a number of currently known (customer) *demands*. The former are determined by manual assertion and/or correction of a system based supply proposal. The latter are selected amongst all known demands by a time horizon, usually enclosing today, tomorrow and the day after tomorrow (see Section 2.6).

The objective is to satisfy the demands with minimum total transport costs. Supplies as well as demands are time dependent and the transport is carried out via generally presheduled freight trains (see Section 2.2). In times of high traffic between different locations additional freight trains can be arranged or superfluous scheduled trains can be cancelled. Both options require a pre-processing time, such that there is no direct dependence of the freight train schedule on the current repositioning of empty cars and our view on the timetable is static. We also exclude regular 'commuter' traffic of unit trains between two or more locations, but concentrate on less regular demands with changing number of cars per demand (wagon load traffic).

Transport costs obviously arise due to the different *localization* of both supplies and demands, which come up at various customer sites in the first place. With respect to the *production process*, which means the realization of an individual empty car transport, several locations can be aggregated to so-called *dispo(sitive) stations* (see Sections 2.1, 2.2). Often the latter are nodes in the railroad network of strategic or technical importance, but basically the aggregation to dispo stations is a tool to allow a reasonably scaled system-wide view of the network.

Considering the variety of goods to be transported, the freight car assigned to a demand has to meet different physical requirements like open or closed cars, different volume and weight capacities or special equipment for example in coil transport. Up to today, the physical features of freight cars are encoded in the *car type*, for our purposes identified by a natural number. Moreover, we distinguish *supply car types*, which always possess real world equivalents from *demand car types*, which can also represent groups of car types equivalently appropriate for the demanding customer. Possible assignments between supplies and demands are then defined by a set of allowed substitutions between car types (see Section 2.3). Also cars of the rolling stock of different companies (different *wagon keepers*) are sometimes distributed within our system. We identify the wagon keeper with a natural number and treat this feature as an extension to the car type.

Sometimes not all current supply can be distributed to demands. As a new feature, those are explicitly assigned to special storage tracks with free capacity, the so called *operative storage* or *short-term storage sidings*[1] as explained in Section 2.4. Thus the transport costs for stored cars are also controlled and influence the solution of the (DP). Although we claimed as the major goal to satisfy all demands (or at least as much as possible), the incorporation of the storage planning into the (DP) solution even allows to prioritize cost issues over demand satisfaction. Such differences in *priority* can also be useful between customers or demands for economic and fairness reasons (see Section 2.5).

Rather static information about instances of the (DP) like the freight train schedule or allowed substitutions and further detailed constraints are sometimes referred to as *master data* in the following as opposed to (current) *input data*, which mostly refer to actual supplies and demands. We remark that the currently installed process of empty freight car distribution at DB Schenker Rail Deutschland AG was also addressed by the diploma thesis of Katharina Beygang [12].

2.1 SUPPLIES AND DEMANDS

As described above, supplies and demands originate at customer sites. The former are returned or rather 'declared free' by the customer, as the car waits for the actual return at the facility after a rental period. The latter includes not only the actual good transport, but also a necessary time overhead to unload the car, which may be lengthy due to 'just in time' production. (The transported good is not unloaded and stored, but processed further directly from the car, such that the unloading time coincides with the production time for a certain final good.)

As rarely the sum of supplies and demands at the same location equal each other from the beginning, distribution and transport is a necessity. To specify the above mentioned demand satisfaction 'in time', we annotate supplies s_i and demands d_j with some attributes, which are explained in detail in the following.

Supplies s_i and demands d_j share the attribute of the *location* (l_i, l_j). The location l_i/l_j is from the range of shunting facilities called *dispo stations*[2]. The latter are distributed throughout the *production area*[3] of Germany and central Europe. We associate each dispo station with a natural number. As only at dispo stations shunting and thus different assignments of individual

1 [germ.] Operative Abstellung
2 [germ.] Dispostellen
3 [germ.] Produktionsgebiet

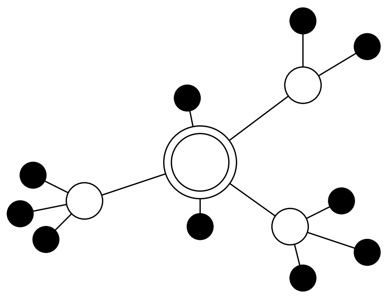


Figure 2: Hierarchic assignment of facilities/customer sites (filled black circles) to collection stations (empty circles) and to dispo stations (double lined circles).

freights cars are possible, the limitation of the range of the location attribute does not influence the optimality of a distribution within our setting.

Each individual customer site is usually assigned to one dispositive station in the master data. For some locations the associated dispo station with respect to returned cars (supplies) differs from the dispo station to which new customer demands are associated. The transport for each empty freight car from its last customer to a dispo station and from the last dispo station to a customer demand is carried out via *customer (fetch/delivery) tours.*[4] The latter possess their own *customer tour schedule.*[5] Figure 2 sketches the mostly hierarchic assignment of facilities (filled black circles) to collection stations (empty circles) usually without shunting facilities and their assignment to dispo stations (doubly lined circles).

The actual time when a supply s_i becomes available to a new assignment is its return time at the associated dispo station l_i for the last customer site it was transported to. More precisely, the earliest subsequent availability time of a supply is the *dissociation time*[6] of the freight train which returned the car(s) to the dispo station by a fetch tour. For dispositive purposes we are only interested in this adjusted availability time c_i of s_i at the dispositive station and not in the return time at the facility. For demands d_j the time attribute c_j is equally adjusted to be the latest time a car must be available at the associated dispositive station l_j, such that the next delivery tour to the actual customer location is early enough to satisfy the demand in time. In both cases, times are represented as natural numbers in a reverse date (year, month, day) and time (hours, minutes) format. (For example '201010181305' encodes the date 18th of October 2010 at 1 p.m. and 5 minutes.) We refer to values of the time attributes in this format as time stamps.

4 [germ.] (Abhol-/Zuführ-) Bedienfahrt
5 [germ.] Nahbereichsfahrplan
6 [germ.] Zugauflösezeit

The range of time stamps c_i for supplies s_i is given by all arrivals or dissociation times respectively of customer fetch tours at location l_i. The range of time stamps c_j for demands d_j is analogously given by the set of *formation times*[7] for the delivery tours. This restriction does also not affect the possible assignments as any time stamp in between two scheduled fetch/delivery tours at the same dispo station can be mapped to the time stamp of the next corresponding tour. On the contrary the time stamps allow for more precise time management than the formerly used time *phases* (see Section 2.6).

The number of cars per supply or demand range from a single car up to tens, which shows the typical situation in wagon load traffic. We associate the number of available and demanded cars, n_i and n_j respectively, with the accumulated supply s_i and demand d_j. To try and reduce the data volume, currently supplies and demands are clustered with respect to location and phases.

2.2 TIMETABLE AND TRANSPORT COSTS

The timetable is predefined such that the distribution decisions do not affect frequency and capacity of freight trains between different locations directly as part of the outcome. Due to seasonal or short-term requirements, scheduled trains can be cancelled or trains can be added at higher expenses. However, this cannot be done 'just in time' due to a (DP) solution, but has to be decided some time in advance. There is also a long-term adjustment of the timetable, which is carried out every two month.

Moreover, loaded and empty freight cars share the same scheduled freight trains, yet both are distributed independently and thus capacity control or management is (so far) well beyond the goals of the project. It is also general production policy to prefer loaded cars over empty cars competing for the same block or train with strained capacity limits. Figure 3 gives an impression of shunting facilities and loaded trains in motion. Further, detailed routing for the individual freight car is not part of the optimization process. Although a more integrated view is now in consideration, during our project and up to now, a so-called 'Richtzahlverfahren' is used to route cars. The latter resembles a postal code based distribution and is comparable to a local Internet routing protocol in the following sense.

Imagine each dispositive station A holds a look-up table with entries for all possible destinations. The entry specifies exactly to which dispositive station the car should be sent next, given its destination. This can either be the associated dispositive station B to the destination, in which case the routing is complete apart from local transport to the customer siding, or

7 [germ.] Zugbildungszeit

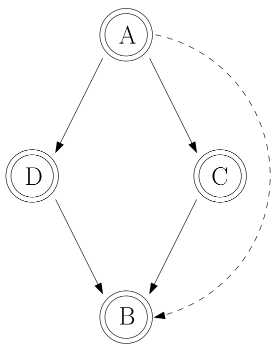
Figure 3: Maschen, shunting yard (DB13244) by Volker Emersleben. Loaded and empty freight cars are assembled to trains together in shunting facilities often equipped with (limited) operational storage. Picture taken from https://mediathek.deutschebahn.com by courtesy of Deutsche Bahn AG.

another dispositive station C on the way. Then the car is classified (shunted) appropriately to be part of the next (possible) train leaving A and passing B or C respectively. In fact, instead of using a look-up table, the classification follows (immediately) from the numeral code, the 'Richtzahlen', of the dispositive stations A and B. The routing between A and B is thus fixed to a single resulting route, the so-called 'Leit(ungs)weg', through the network, as long as no code changes occur. Such a route consists of one or more freight train connections which are (de-)composed at start dispo, target dispo and possibly rearranged at intermediate yards[8]. We call those yards *via dispositive stations* (like C in the above example).

Figure 4 shows the route from A to B (via C, not via D). Straight, solid arcs correspond to direct freight train connections and the dashed arc corresponds to the predefined routing between A and B, which is composed of two such connections. The right hand side table displays the artificial timetable, which contains both direct train connections and composed connections corresponding to the current 'Leitweg'. (Note that due to the (de-)composition of a train in C, the cost c_{AB} are not the sum of c_{AC} and c_{CB}, but include an additional shunting cost term depending on the dispo station C.)

The 'Leitweg' can change over time (with the freight train schedule or independently), but is unique with respect to a certain time interval.

8 [germ.] Umstellungen



l_i	c_i	l_j	c_j	c_{ij}
A	201011300500	C	201011301000	253
A	201011300500	B	201011301251	586
A	201011300600	D	201011301128	359
...	...	...	...	...
C	201011301042	B	201011301251	313
...	...	...	...	...

Figure 4: 'Leitwege' define a set of via stations for each pair of dispositive stations.

This simplifies the routing of individual freight cars and enables us to develop a more compact model. Nevertheless, the concept of 'Leitweg' owes more to technical requirements than to simplification: As mentioned in the introduction, handling each car individually would be far to expensive (both in terms of cost and work load). The distinct assembling of cars into trains in manifold directions otherwise complicates the shunting process or exceeds capacities of a single dispositive station, especially at central points of the railway network.

Therefore cars are grouped into blocks and blocks are assigned to trains, such that even at intermediate rearrangements of trains only complete blocks and no individual cars must be handled, as long as the corresponding dispositive station is a via station and not the destination of the block. On the other hand, this multistage assignment basis on which the transport is operated makes individual operational car routing a difficult and time consuming task, which is not addressed in this thesis.

2.2.1 *First Train Possible*

For our model assumptions, DB Schenker Rail Deutschland AG follows a *first train possible policy*. This means that a supply allocated to a demand uses the first possible connection between supply and demand location. In practice, a train must not only provide the appropriate connection according to the respective 'Leitweg', but also sufficient capacity. The latter is not modeled explicitly, which is partly due to the lack of data with respect to loaded cars as mentioned above.

Thus whenever a freight car is to be added to a train or changes a train during a connection, the train can already have exceeded its maximum length or weight (due to loaded cars) in the actual production. In this case, the car has to wait for the next appropriate train. This can also cause a need for temporal storage (otherwise the empty car hinders subsequent shunting operations), which can again result in a delay and the missing of the next possible train.

To compensate for such effects of increasing waiting and transport times, we forbid a deliberate delay and (implicitly) assign the first possible train to an assignment between a certain supply-demand pair. As a consequence, freight cars might arrive early at the demand dispo and have to be buffered before the next delivery tour consuming operational storage capacity. Although we incorporate management of operative storage capacity (see Section 2.4), we decided in favour of the first train possible policy for our model as a good trade-off between data availability, model complexity and operational reality.

2.2.2 *Transport Cost*

Each connection in the timetable delivers the primary costs c_{ij} for a single car transport between supply location l_i and demand location l_j. The transport costs are assumed to be non-negative integer values (in Euros or Euro-cents). Those are comprised from the transport costs per track kilometer and an individual cost term for a shunting operation at possible via stations. The latter generally includes shunting at the start, but not the target station.

2.2.3 *Local Transport Cost*

Due to the local accumulation of supplies and demands with respect to dispo stations and the corresponding timetable, the transport costs account only for transfer between dispositive stations. There are comparatively few such stations in the production area, such that transport from/to customer locations to/from the corresponding dispositive station cannot always be neglected. Yet, the cost for those transports by fetch or delivery tours are fixed due to the fixed assignment of the customer location to a dispositive station. Thus we annotate the local transport cost as an attribute g_i and g_j to supply and demand respectively. The local transport costs are also assumed to be non-negative integer values (in Euros or Euro-cents).

2.3 CAR TYPES AND SUBSTITUTION

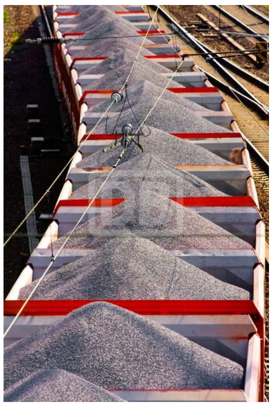
Any supply or demand of empty freight cars specifies a certain car type. Such a type can also define a group[9] of possible car types. Figure 5 shows a small collection of different car types. We speak of (real) *supply car types* and *demand car types* (including artificial types), if distinction is necessary. All types are given by one continuous numbering. As mentioned above,

9 [germ.] (Misch-)Bedarfstyp

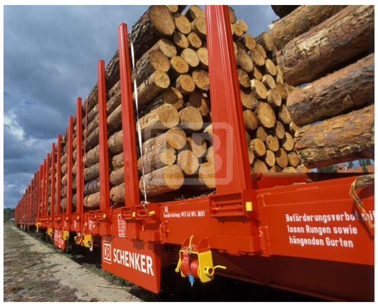
(b) paper transport (DB2868) by Margit Brettmann

(a) coil transport (DB13572) by Michael Neuhaus

(d) bulk transport (DB962) by Petra Schwaiger



(c) gravel transport (DB13010) by Bartlomiej Banaszak



(e) wood transport (DB11838) by Wolfgang Klee

Figure 5: Different Car Types. All pictures taken from https://mediathek.deutschebahn.com by courtesy of Deutsche Bahn AG.

equality of car types between supply and demand is not always necessary for a valid assignment. Instead, each (demand) car type has an associated list of allowed (supply) car types to substitute its explicit type. The allowed supply car types for each demand car type are given as a set of substitution rules in the master data.

In the context of substitution we employ a distinction between homogeneous and heterogeneous substitution in terms of the 'car exchange rate', which can be phrased as 'How many cars of type A are necessary to substitute n cars of type B?'

In the majority of cases the number of supplied cars equals the number of demanded cars (*homogeneous* or 1-to-1 substitution). Nevertheless, there are also cases in which the number of cars necessary to satisfy a demand differs from the demanded number of cars according to the supply type. As a practical example, consider a customer intending to transport bulk cargo (sand, grain, etc.) and a cargo company which offers two appropriate supply car types X and Y. Let X be smaller and Y larger, such that a car of type Y supplies the double transport volume than a car of type X. The customer needs to transport a certain volume of bulk cargo in either n big cars or $2n$ small cars or any appropriate combination adding up to the desired total volume.

In our case, the cargo company provides an artificial demand type Z, such that a car of this type is allowed to be substituted by either one car of type Y or two cars of type X. We call this constellation of substitution rules *heterogeneous* (or in this case 2-to-1) substitution.

As in practice only 1-to-1 substitution and 2-to-1 substitution occurs for our application, we concentrate on (approximate) solutions for this case, as the general heterogeneous substitution renders the problem significantly harder to solve to optimality (see Chapter 4).

2.3.1 *Foreign Wagon Keepers*

The basic substitution rules only rely on the notion of (artificial) car types. An additional attribute f_i/f_j of supply and demand respectively represents the *wagon keeper*[10] by a natural number. The entity in charge of maintenance of a car, or wagon keeper for short, is the company which is generally in charge amongst others of the repositioning of a car. (And it is not necessarily, but often, its owner.) Due to international transport, freight cars of a company's stock emerge rather remote from their own central production area as available supply. In such cases, other local companies take over the dispositive process for the so-called 'foreign car' for a limited number

10 [germ.] Wagenverfügungsberechtigter

of assignments. The latter is subject to additional rules, which distinguish between three kinds of reuse of foreign cars:

1. domestic transport
2. cross-border transport
3. return of empty wagons to the wagon keeper

The wagon keeper can be seen as an actual expansion of the car type and is as such an attribute of supplies and demands. The customer can prefer a car of a certain type and wagon keeper to a car of the same type, but different wagon keeper as the latter sometimes offers minor but specific physical differences.

Note that for the above classification of transports not only the demand or customer location, but also the destination of the following loaded run has to be considered. The latter is specified by the appropriate dispositive station r_j as an additional attribute of the demand d_j. If no destination of the loaded run is known for d_j, r_j is set to zero and foreign car reuse is forbidden for d_j, as rules based on the above classification cannot be employed.

To further distinguish which rules should be applied to certain demands, we annotate the demand with an additional flag a_j, which indicates a domestic ($a_j = 1$) or foreign ($a_j = 0$) destination for the loaded run. We call r_j the *lr-target*[11] of d_j for short. If the wagon keeper of a demand is not specified (represented by $f_j = 0$) it allows the assignment of every 'foreign car' with appropriate type respecting the following rules. Those are currently reformed to be represented by a *Car Reuse and Return Matrix (CReM)*[12]. Up to now, such individual regulations only exist between few wagon keepers and mainly reflect the rules we describe below.

2.3.2 *Domestic Transports*

Generally, a foreign car can be repositioned once within the productive area, which means the assignment to a demand with an lr-target in this area. Each supply carries a Boolean flag u_i to distinguish whether is has already been reused for this purpose ($u_i = 0$) or not ($u_i = 1$). Further, such an assignment is only allowed, if the lr-target is geographically nearer to the respective wagon keeper's productive area than its current supply location. Again, if the lr-target is unknown, the rule forbids the assignment of any foreign car to the demand. To manage this relation without an integrated map or geo-information system and avoid exhaustive memory requirements, we

11 [germ.] Lastlaufziel
12 [germ.] Wagenverwendungs- und Rückleitungs-Matrix (WuRM)

use a regional geographic clustering to incorporate this rule in the model. Nevertheless, we refer directly to the individual locations for reasons of simplicity.

2.3.3 *Cross-border Transports*

Disposing a foreign car to a demand with an lr-target outside the productive area of DB Schenker Rail Deutschland AG is not ruled as finely grained as in the domestic case. All potential lr-targets in the production area of one foreign rail transport company correspond to a single (virtual) dispo station. In this context, we need to consider the *target company* managing the productive area in which the lr-target is located. (Note that this 'location' of the lr-target is to be understood only more or less in a geographical sense for historical reasons. More than one company can generally have facilities in the same city for example and the central factor is the company in charge for the particular transport process, not the city.) Further, most rules also consider a *first transit company*, which is the first company, whose productive area is crossed by the transport of a car if it is neither DB Schenker Rail Deutschland AG nor equal to the target company. (Also note that the target company does not need to equal the wagon keeper.)

Cross-border transports can also be allowed only considering the first transit company. For example, foreign cars can often be sent to many different target companies, provided the first transit company corresponds to the wagon keeper. Therefore, each demand is annotated by the attributes e_j and z_j encoding the target company and the first transit company by a natural number. Allowed reuses are then specified as explicit triples of wagon keeper, first transit and target company.

2.3.4 *Return Transport*

If a foreign car cannot be reused (assigned to a demand), it must be returned to the wagon keeper's productive area via any or a special dispositive station at its border. Those border stations can be viewed as special demands. (Mostly border stations are considered multiple times for a number of subsequent days, such that capacity control for each day is possible.) Allowed border stations for a supply are specified by triples of wagon keeper, car type and border station. It is also possible to directly specify a return border station for individual supplies by another attribute b_i of s_i, which again encodes a predefined border station as a natural number. Note that foreign cars are never explicitly stored.

2.4 OPERATIVE STORAGE

In general, demands of empty freight cars tend to exceed supplies, especially as demands are typically known longer in advance than supplies. Still, there are season dependent surpluses of distinct types of freight cars or - like during summer or Christmas holidays, when the industrial workload is reduced - a general backlog. In such cases, freight cars have to be stored into special sidings, which are distributed throughout the production area. Due to their inherent space requirements such storage sidings are often remote, although each storage location is assigned to a dispo station like the customer sites. Storage and subsequent reuse of cars tends to be expensive both in terms of the costs to be paid for the transport and the time before a remotely stored freight car can be assigned to a demand again. (The latter increases the wagon cycle time or average idle time of empty cars which is economically similar to cost increase – time is money.)

The latter two aspects lead to a subdivision of all storage capacity into long-term storage and short-term or so-called operative storage, where long-term storage use the more remote storage sidings. Further, freight cars in long-term storage are not scheduled for regular inspection intervals. Consequently, those cars have to pass an extra inspection before a new assignment, which makes them unattractive for short-term reuse. Operative storage areas are located relatively close to dispositive stations with shunting facilities. This makes a short-time operational storage attractive, but also strongly limits space capacities.

Today surplus cars are not explicitly distributed and rather stored in the supply location at hand if there is enough capacity. If not, they are transported to locally optimal or rather seemingly good free capacities. Expert knowledge makes this process generally successful in practice, but especially in situations of extreme backlog, operational storage is a challenging task, as shunting facilities tend to be blocked by surplus supplies. Further, 'attractive' storage locations or those where supplies occur frequently may tend to operate at their capacity limit in regular situations. This reduces the efficiency of corresponding shunting facilities.

In sum, long-term storage of freight cars is too reluctant to be incorporated into a daily optimized distribution. But operative storage, as opposed to today's practice can be taken into account to some extend, regarding for example the given capacities and the global optimization of storage in terms of transport and shunting cost. Note that from the comparably great differences in needs of some transportable goods (e.g. steel coils and bulk cargo) it is evident that at least for some groups of car types the (DP) could be solved separately so far. However, in our model, we abandon separation for the sake of planning operative storage. Respecting storage capacities is only possible if every car type is present in a single optimization run.

2.4.1 *Operative Storage as Supply*

Usually supplies of freight cars are known in advance to become available at a certain place and time. By the time they become available, in the best case they are already assigned a new destination, such that they neither block the shunting facility nor have to be shunted twice or more in order to store them in a local operative storage and distribute them afterwards (see also Section 2.6).

On supply side, the assignment of a stored car and a car on the move to a demand differs mainly in terms of (additional shunting) cost. To distinguish both cases, each supply s_i is annotated by a natural number o_i representing the associated dispositive station of its current storage or zero, if the supply cars are on the move. Time stamps and (local) transport costs are provided by local fetch tour schedules like in the case of returned car supplies from customer sites.

Further, a single possible storage can be predefined for supplies. If for example a supply has to be stored at some distance from its supply location it usually passes via dispo stations, where the dispositive target of a car is technically possible to be changed. Formerly not known demands can have occurred in the meantime. A supply already assigned to a (long distance) storage can be reentered as supply at a via station to take new demands into account. However, a dispositive change is only intended to satisfy a new demand. Changing the supply's destination to another storage (with freed capacity in the meantime) due to cost optimizing, would disturb the productive process without ample benefit and is thus forbidden. Supplies with storage destination, which are possibly reassigned to demands are therefore annotated with a natural number v_i, which represents the associated dispo station of the storage destination or zero, if no such destination is set.

2.4.2 *Operative Storage as Demand*

An operative storage can also be viewed as a kind of demand, which generally allows mixed types of cars. The access to a storage can be explicitly limited to special car types by individual substitution-like rules for each individual storage according to traffic characteristics or different categories of operative storage. For example really fast accessible and moderately fast accessible storage can be limited each to those car types which are seasonably rather in demand and to those who are not. Further, cars of other wagon keepers are generally not stored, but returned, if they cannot be reused.

On supply side, already stored cars can be distinguished with respect to their type, but on demand side all types compete for the limited amount of empty track space. Consequently the capacity of a storage is not clearly defined. The base value for the storage capacity is the length of empty track in metres. To discretize this value taking the mixed car types with individual length into account, the average length of a car is computed regarding the percentage of a certain type with respect to the whole car fleet.

To avoid overflows due to averaging, a buffer is subtracted from the total rail length. Further, a buffer is needed for cars which are not especially transported into operative storage but are shunted at a respective dispositive station on their way to a demand. Here a car may have to wait for another train on its way either if it arrives too early (perhaps due to the first train possible policy, see Section 2.2.1) or if an appropriate train has no capacity left and the car has to wait for the next possible train. Such waiting times occur especially at the destination's dispositive station. The car has to wait there for the next local delivery tour passing the customer site. As the waiting time in such cases highly depends on productional data which is not provided beforehand, such delays and the corresponding amount of storage capacity cannot be taken into account precisely. From a theoretical point of view, the buffer solution also prevents an NP-complete sub-problem as shown in [12].

The remaining netto length of a storage rail is then divided by the average car length, resulting in the storage capacity. The latter can be modified according to traffic characteristics, for example, if the cars stored are usually of a type which is significantly shorter or longer than the average or if waiting times tend to higher or lower values.

Dependence on fetch and delivery tours and/or scheduled shunting activity complicates time coherent capacity administration for operative storage locations. On supply side, the given availability time at the dispositive station for stored supplies already respects corresponding fetch times. On demand side, the following case may occur: a number of stored cars originally occupies the full capacity, but is distributed to other demands while other cars are scheduled for the respective storage. Thus the capacity of the storage before and after the complete distribution process is never violated. During this process, the next fetch tour, which frees capacity in the respective storage may possibly be scheduled after the next delivery tour which already relies on the freed capacity. Thus we introduce a certain time expansion for each operative storage (see the respective definitions in Chapter 4 and the basic model definition in Chapter 5).

2.5 PRIORITIZATION OF CUSTOMERS/DEMANDS

Customers or demands can currently be awarded different priorities. Marketing reasons, but also model inherent reasons can be relevant factors to employ a prioritization. An example for the latter is that remote customer sites with respect to supply locations suffer a permanent disadvantage compared to demands which are located near to usual supply sources. A geographic location which results in relatively high transport cost for any assigned empty car leads to permanent neglect in a model with cost-oriented objective like the one currently used and our own model. As a measure against thus created unfair distribution, a demand or customer site is made more attractive in terms of costs by reducing the kilometre based pure transport costs with a negative cost term. Hence, the geographic disadvantage is compensated to a variable extent.

By similar means, a trade-off can be implemented between demands which are close to their due date and demands which are known longer in advance. Within a moderately long time horizon, demands with a rather short interval to their due date compete for the same supplies as demands which can possibly (and likely) be also satisfied with upcoming supplies for the following day. Again, 'less attractive' demands in terms of cost will only be considered in the distribution model after other demands are satisfied. By this time, due dates may be expired and a demand remains unsatisfied. A time dependant cost term with a negative cost effect inversely proportional to the remaining time to satisfy the demand leads to a more reasonable policy in terms of due dates. This is especially useful for the consideration of a longer time horizon. However, we currently do not apply cost terms in this second case, as such effects are already considered for the selection of input data from the set of all known demands.

Both of the above examples use variable prioritization cost terms w_j per demand d_j. Their effect is relative to pure or modified costs of other demands and they do not guarantee an actual preference. We call this cost-induced prioritization 'weak priority'. Weak priorities have already been employed in the current DB Schenker Rail Deutschland AG model to some extent.

With respect to some technical or procedural requirements, weak prioritization is not sufficient to ensure side constraints. An example for such a requirement is the so-called 'consent' of a number of cars to a customer. Usually a number of cars is consented to a customer (by telephone, fax, etc.) if required, after a distribution is established and before the production process is completed. In today's daily distribution, a consented car can only fail to satisfy a demand due to unforeseen car damage or transportation failure. After a number of cars is consented or 'promised' to be assigned to a demand, leaving this demand less satisfied in the final assignment is not allowed, regardless of costs. Therefore, weak prioritization cannot be used here without stretching the artificial cost terms beyond reasonable scales and thus risking side effects.

In the context of a dynamic distribution, it is possible to establish a tentative solution and keep a supply in the optimization process during its planning interval (see Section 2.6). The corresponding demand of the

(a) Frankfurt, control center (DB4788) by Mario Vedder

(b) Wagon examiner (DB4383) by Michael Neuhaus

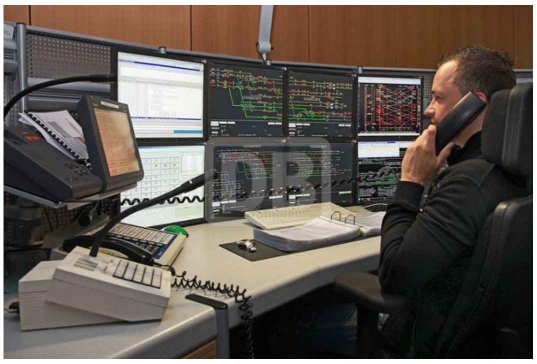

(c) Duisburg, control center (DB14067) by Jürgen Brefort

Figure 6: Planning and production process depend on smooth interaction between automated and manual steps. All pictures taken from https://mediathek.deutschebahn.com by courtesy of Deutsche Bahn AG.

tentative solution is also kept. Changes in the input data, for example a new demand with less associated costs, can lead to a cheaper solution with a deficient demand. The latter solution is preferable in terms of cost minimization. However, if the tentative number of assigned cars was consented, a solution with less cars allocated to that demand is not acceptable, whereas the allocation of the same number of cars in a cheaper way is feasible.

Hence, we also introduce a 'strong' prioritization in three levels and annotate each demand d_j with its level of strong priority $p_j \in \{0, 1, 2\}$. The priority levels essentially cluster the demands for the subsequent (re-)optimization process according to their level of variability and thus their optimization potential. The latter can this way be exploited while the same level of customer service and reliability is maintained.

2.6 DAILY DYNAMICS OF THE FREIGHT CAR DISTRIBUTION

As described in Section 1.1, the current work flow at DB Schenker Rail Deutschland AG is organized on the basis of a single daily distribution.

The distribution is automatically optimized on an aggregated level and separately for seven main type groups. A considerable amount of manual pre-processing (data acquisition) and post-processing (individual final assignments) is required. The computation is also influenced by certain parameters, which are selected by fleet managers to trim the solution to the general relation between total supply and demand like (critical) over-/undersupply.

Note that the generated solution accounts only for transport between dispositive stations (due to clustering) and provides for each dispo station the number of available cars (local supplies and supplies received from other dispo stations) and the number of local available cars to be sent to other dispo stations. The total amount of available cars is then manually assigned to the actual local demands in the *final assignment*.[13] An individual assignment composes of two or three 'parts': The transport from the actual returning customer or operational storage to the associated dispo station, possibly the transport to another dispo station and finally the transport to an actual customer site. The first and second part are mostly combined in one consistent assignment. Upcoming supplies (late returns, returns from foreign companies, etc.) and demands or cancellations (of demands or defective cars) are managed manually in the remaining part of the day.

Apart from reducing manual workload and exploiting potential optimization marges, the following aspect is the most relevant reason for DB Schenker Rail Deutschland AG to aim at a 'permanent' distribution: The manual handling during the large optimization gap of 24 hours can lead to temporal storage of a fresh supply car. If the corresponding new assignment arrives too late, which means after dissociation of the fetch train, no subsequent transport can take place without reclassification of the whole storage siding. This can take considerable time, which is idle time for the car. A car which is automatically and directly assigned a new destination, can be classified accordingly during the train dissociation without temporal storage and waiting times, provided it becomes known to the system early enough. Consequently, idle times of empty cars are decreased resulting in reduced waggon cycle times and thus important economic savings.

With multiple optimization runs per day, planning intervals of supply and demand can be exploited. The latter can be specified more precisely due to the introduction of time stamps to replace phases. A benefit of the phases is the enhanced opportunity for supply and demand aggregation by clustering. While clustering leads to data reduction, it also has drawbacks, like the above mentioned necessity of manual final assignments and the implicit split of assignments in up to three parts. Each part requires fault resistance in data exchange and manual control and may cause temporal storage, additional shunting cost and increasing idle times.

13 [germ.] Feinverteilung

The effect of (re)optimization during planning intervals is that supplies can be reassigned to upcoming demands and a possibly more cost-effective solution can be found. Today, demands seldom occur scattered over the day, because DB Schenker Rail Deutschland AG's customer politics requires announcements and cancellations of demands early enough with regard to the work flow. Apart from cost-reduction due to reoptimization, also customer convenience can be increased by a less restrictive policy. To avoid idle times in the work flow caused by multiple optimizations per day, we develop a fast and robust reoptimization concept which also contributes to a heuristic solution of the computationally hard aspect of heterogeneous substitution.

3 PRELIMINARIES

This chapter collects basic definitions and notations with regard to graphs, algorithms, complexity and linear programming. It is designed as a reference and the reader is encouraged to skip the section until needed.

3.1 GRAPHS AND NETWORKS

All given definitions, notations and properties are in line with [2].

A *directed graph* $G = (V, A)$ consists of a set V of vertices or nodes and a set A of arcs whose elements are ordered pairs of distinct nodes. A directed network is a directed graph whose nodes and/or arcs have associated numeral values (typically, costs, capacities and/or supplies and demands). Let n denote the number of vertices and m the number of arcs in G.

We define an *undirected graph* in the same manner as a directed graph except that the arcs are unordered pairs of nodes.

A directed arc (i, j) has two *endpoints*, i and j. We refer to i as the *tail* and j as the *head* of the arc, which *emanates* from i and *terminates* at j. An arc (i, j) is *incident* to its endpoints, which are *adjacent* due to $(i, j) \in A$. Arc (i, j) is an *outgoing* arc of node i and an *incoming* arc of node j. The *in degree* of node i is the number of incoming arcs, the *out degree* is the number of outgoing arcs and their sum is the *degree* of i.

Multiarcs are two or more arcs with the same tail and head nodes and a *loop* is an arc whose head and tail node are the same.

A graph $G' = (V', A')$ is a *sub graph* of $G = (V, A)$ if $V' \subseteq V$ and $A' \subseteq A$. We say that G' is the *sub graph induced* by V' if G' contains each arc of A with both endpoints in V'. G' is a *spanning sub graph* if $V' = V$ and $A' \subseteq A$.

A *walk* in a directed graph is a sub graph consisting of a sequence of nodes and arcs $i_1 - a_1 - \cdots - i_{r-1} - a_{r-1} - i_r$ such that for all $1 \leqslant k \leqslant r$ either $a_k = (i_k, i_{k+1}) \in A$ or $a_k = (i_{k+1}, i_k) \in A$. We also refer to walks by sequences of node or arcs omitting the arcs resp. nodes in between. A *directed walk* is a walk with arcs $a_k = (i_k, i_{k+1}) \in A$ for any consecutive i_k, i_{k+1}.

A *(directed) path* is a *(directed) walk* without node repetition and a *(directed) cycle* is a (directed) path $i_1 - \cdots - i_r$ together with the additional arc ($(i_1, i_r) \in A$ or) $(i_r, i_1) \in A$. A graph is *acyclic* if it contains no cycles.

Two nodes i and j are *(strongly) connected* if G contains at least one (directed) path between i and j and G is connected if every pair of vertices is connected, otherwise G is *disconnected*. The maximal connected sub graphs of a disconnected graph are called its *components*.

A graph is *weighted* if there is a weight function $c : A \to \mathbb{R}$ and/or $w : V \to \mathbb{R}$ defined on its arcs and/or nodes. The *length* of a path (cycle) is the number of its edges, if the graph is unweighted. Otherwise the length of a path is the sum of the weights on its arcs and/or nodes. The distance $\mathtt{dist}(i, j)$ of two nodes i and j in a graph is the minimum over the length of all paths between i and j and ∞ if i and j are in different (strongly) connected components.

A *cut* is a partition of the node set V in two parts, S and $\bar{S} = V \setminus S$. Each cut defines a set of arcs that have one endpoint in S and the other in $\bar{S}$. An $s - t$*—cut* with respect to two distinguished nodes s and t is a cut such that $s \in S$ and $t \in \bar{S}$.

A *tree* is a connected acyclic graph and a *forest* is a collection of trees. A *rooted tree* has a special node R, the root. A subtree $\mathtt{st}(r)$, rooted at r of a rooted tree is an induced subgraph of all nodes i, such that a path from R to i passes r. The *height* $h(T)$ of a rooted tree T, is the maximum length of a path from its root to another node in the tree.

A *bipartite* graph is a graph whose node set V can be partitioned into V_1, V_2, such that for each arc $(i, j) \in A$ either $i \in V_1$ and $j \in V_2$ or vice versa. A bipartite graph contains no odd length cycle and each acyclic graph (for example trees and forests) is bipartite as well as each graph containing only even cycles.

A graph $G = (V, A)$ is called *Eulerian* [35] if there exists a cycle containing each arc of A exactly once. Such a (directed) eulerian cycle exists if and only if G is (strongly) connected and (the in- and out degree of each node are the same, therefore) the degree of all nodes of V are even.

Let $N = (V, A)$ be a directed network with three associated functions:

- *balance function* $b : V \to \mathbb{R}$
- *arc capacities* $c : A \to \mathbb{R}^{>0}$
- *arc cost* $w : A \to \mathbb{R}^{>0}$.

Then the *minimum cost flow* problem in N can be stated as follows:

$$\text{Minimize} \sum_{(i,j)\in A} f(i,j)w(i,j) \tag{3.1}$$

subject to

$$\forall i \in V \ : \ \sum_{j:(i,j)\in A} f(i,j) - \sum_{j:(j,i)\in A} f(j,i) = b(i) \tag{3.2}$$

$$\forall (i,j) \in A \ : \ 0 \leqslant f(i,j) \leqslant c(i,j) \tag{3.3}$$

Each vertex $i \in V$ with $b(i) > 0$ is called a *source* with *excess* or *supply* $b(i)$ and each vertex $j \in V$ with $b(j) < 0$ is called a *sink* with *deficit* or *demand* $b(j)$. We define the value $|f|$ of the flow f in $N = (V, A)$ as:

$$|f| = \sum_{v \in V, b(v) < 0} f(u, v).$$

Let $N = (V, A)$ be a directed network with three associated functions as above and an additional *multiplicator function*:

- *balance function* $b : V \to \mathbb{R}$
- *arc capacities* $c : A \to \mathbb{R}^{>0}$
- *arc cost* $w : A \to \mathbb{R}^{>0}$
- *multiplicator function*: $\mu : A \to \mathbb{R}^{>0}$.

The *generalized minimum cost flow* problem in N can be stated as follows:

$$\begin{aligned} & \text{Minimize} \sum_{(i,j) \in A} f(i,j) w(i,j) && (3.4)\\ & \text{subject to} && \\ \forall i \in V \quad : \quad & \sum_{j:(i,j) \in A} f(i,j) - \sum_{j:(j,i) \in A} \mu(j,i) f(j,i) = b(i) && (3.5)\\ \forall (i,j) \in A \quad : \quad & 0 \leqslant f(i,j) \leqslant c(i,j) && (3.6) \end{aligned}$$

Note that a generalized flow network with unit multipliers corresponds to a classical flow network and that there are also more instances of the generalized minimum cost flow problem that can be expressed in terms of a modified classical flow network.

We further define the *multiplier of a generalized walk, path and cycle* as the product of the arcs multipliers along it. A *unit or break even cycle* is a cycle with multiplier 1, a *gainy (lossy) cycle* has multiplier greater (less) than 1.

Given an undirected graph $G = (V, E)$, $n = |V|, m = |E|$ a *matching* $M(G)$ on the graph is a subset of the set of Edges ($M(G) \subseteq E$), such that no two edges share a common endpoint. A vertex $v \in V$ is matched in $M(G)$, if there is an edge $e = (u, v) \in M(G)$. A matching $M(G)$ is (inclusion-wise) maximal, if no edge $E \setminus M(G)$ can be added to $M(G)$, such that the resulting edge set is still a matching. In contrast, we call a matching maximum, if there is no matching $M'(G)$ on G with higher cardinality.

A matching $M(G)$ is a perfect matching if the (disjoint) union of the endpoints of edges in $M(G)$ covers V, or simply if every vertex $v \in V$ is matched. The cardinality of a perfect matching on a graph G with n nodes

Table 1: Landau Symbols.

Notation	$x \to \infty$
$f \in \mathcal{O}(g)$	$\exists\, c > 0\; \exists\, x_0\; \forall\, x > x_0 : \lvert f(x)\rvert \leqslant c \cdot \lvert g(x)\rvert$
$f \in o(g)$	$\forall\, c > 0\; \exists\, x_0\; \forall\, x > x_0 : \lvert f(x)\rvert < c \cdot \lvert g(x)\rvert$
$f \in \Omega(g)$	$\exists\, c > 0\; \exists\, x_0\; \forall\, x > x_0 : \lvert f(x)\rvert \geqslant c \cdot \lvert g(x)\rvert$
$f \in \omega(g)$	$\forall\, c > 0\; \exists\, x_0\; \forall\, x > x_0 : \lvert f(x)\rvert > c \cdot \lvert g(x)\rvert$
$f \in \Theta(g)$	$\exists\, c_0 > 0\; \exists\, c_1 > 0\; \exists\, x_0\; \forall\, x > x_0 :$
	$c_0 \cdot \lvert g(x)\rvert \leqslant \lvert f(x)\rvert \leqslant c_1 \cdot \lvert g(x)\rvert$

is $M(G) = \frac{n}{2}$. We call a matching (problem) bipartite, if G is bipartite. In the case of unit edge weights, maximality of a matching refers to the cardinality of the matching. Given a weight function $w : E \to \mathbb{N}$, we can also search for a minimal/minimum/maximal/maximum weight (perfect) matching on G. All variants are polynomial time solvable.

3.2 ALGORITHMS, COMPLEXITY AND APPROXIMATION

The definitions and notations of this section are in line with [2] and [24]. To denote the running time of an algorithm, we use Landau symbols (also suggestively called 'big-O' notation), which is formally defined in Table 1 and was introduced in [85].

We refer to an algorithm as *pseudo-polynomial*, if its running time is bounded by a polynomial in n, m and upper bounds U, C, M, B on the capacity, cost, multiplier and balance values. (Let F generally denote an upper bound on all four values U, C, M, B.) An algorithm is *polynomial* if the bound on its running time contains logarithmic terms in F and an algorithm running polynomial in n and m in the worst case is called *strongly polynomial*.

Each recognition ('yes-no'-) problem which can be decided by a (strongly) polynomial time algorithm is contained in the problem class P. The problem class NP consists of all such problems, for which the verification of a 'yes'-instance takes polynomial time given the solution. The latter is also true for problems in P, such that $P \subseteq NP$. The question if equality holds for P and NP is a long standing open problem.

The class of NP-complete problems contains those problems of NP, which are as hard to solve as any other problem in NP. Formally a problem L is NP-complete, if it is in NP and NP-hard, which means there is a polynomial time reduction $f_{poly}(L')$, such that for any instance x' of any problem $L' \in NP$: $x = f(x') \in L \Rightarrow x' \in L'$ and $x = f(x') \notin L \Rightarrow x' \notin L'$. Given such a reduction f_{poly}, a polynomial time algorithm for problem L can be applied to obtain a polynomial time solution to any problem L' in NP, which is the key idea of complexity classification of problems and NP-completeness. (Note that in case L' is NP-complete itself, the reduction $f_{poly}(L')$ suffices to show NP-completeness of L.) A collection of NP-complete problems can be found in [47].

Optimization problems such as 'minimize $f(x)$ under a number of constraints' can be viewed as a series of decision problems of the form 'given a number of constraints, is the minimum of f(x) smaller than k' for a problem dependent range of k. Thus we can also employ the above classification for optimization problems or derive analogous classes for optimization problems.

For computationally hard optimization problems, it is desirable to approximate the optimal solution within a constant (multiplicative) factor by a polynomial time algorithm, especially if such algorithms are unknown for the optimal solution (as up to now for all NP-complete problems). We define the *(worst case) approximation ratio* as $\rho = max\{\frac{C^*}{C}, \frac{C}{C^*}\}$, where C^* is the cost of an optimal solution and C the solution of the approximation algorithm. The ratio is maximized over all possible problem instances. (Note that the first term applies to maximization and the second term applies to minimization problems, such that $\rho \geqslant 1$ measures the multiplicative deviation of the approximation with regard to the optimum in the worst case.)

We call a function β-*fractional* or of *bounded fractionality* β for some $\beta \in \mathbb{N}$ if all its values can be expressed as integer multiples of $\frac{1}{\beta}$.

3.3 LINEAR PROGRAMMING, DUALITY

The definitions and notations of this section are in line with [18]. The generic form of a *linear program (LP)* is as follows:

$$\begin{aligned} \text{maximize } c^T x& \\ \text{s.t.}& \\ Ax \leqslant b& \\ x \geqslant 0& \end{aligned}$$

The task of solving this LP is maximizing a linear objective function $z = c^T x$ with real valued vectors $c, x \in \mathbb{R}^n$, such that m constraints on the n non-negative variables (entries x_j of x) of the form $\sum_{j=1}^{n} a_{ij}x_j \leqslant b_i$ ($a_{ij}, 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n$ entries of A and $b_i, 1 \leqslant i \leqslant m$ entries of b) are satisfied. The first algorithm for solving LPs was the *simplex method*, developed by Dantzig in 1963 [26]. Meanwhile, sophisticated implementations and improvements of the simplex method, as well as other specialized and general purpose solution methods for LPs have been developed.

An important concept in linear programming (study and solution of LPs) is duality. The *dual problem* to the above stated LP (which, in this context is called the *primal problem*) is:

$$\begin{aligned} \text{minimize } b^T y \\ \text{s.t.} \\ A^T y \geqslant c \\ y \geqslant 0 \end{aligned}$$

The m dual variables $y_i, 1 \leqslant i \leqslant m$ (entries of $y \in \mathbb{R}^m$) correspond to the constraints of the primal LP and the n constraints $\sum_{i=1}^{n} a_{ij} x_j \geqslant c_j$ ($a_{ij}, 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m$ entries of A^T and $c_j, 1 \leqslant j \leqslant n$ entries of c) of the dual LP correspond to the variables $x_j, 1 \leqslant j \leqslant n$ of the primal LP.

The above defined (generalized) minimum cost flow problem is a special case of LP, as both the objective function and the flow conservation and capacity constraints are linear functions/inequalities. We can reformulate the constraints in matrix form, employing the *node arc incidence matrix* $M \in \mathbb{R}^{|V|} \times \mathbb{R}^{|A|}$ of the respective network $N = (V, A)$ with

$$m_{ij} = \begin{cases} -1, \text{arc } j \in A \text{ originates at node } i \in V \\ \mu(j), \text{arc } j \in A \text{ ends at node } i \in V \\ 0, \text{otherwise} \end{cases}$$

A (node arc incidence) matrix is called *total unimodular* if the determinant of all its non-singular submatrices is ± 1. An LP with a total unimodular constraint matrix has an integral optimum. This concept is generalized to *k-regularity* of matrices in [7, 8] which guarantees k-fractionatal LP solutions (see Chapters 7, B).

We briefly introduced LPs here, as solutions to various flow problems are obtained by LP solution techniques and combinatorial algorithms partly employ optimality criteria deduced from duality theory of LPs. For further reading consult [18, 2].

4 THE FORMAL DISTRIBUTION PROBLEM

In this Chapter, we review our application oriented description and discussion of the *distribution problem* (DP) from Chapter 2 and formalize it accordingly in the following Section 4.1. In Section 4.2, we show that (DP) is NP-complete in its general setting.

4.1 THE DISTRIBUTION PROBLEM (DP)

Supplies and demands are defined as tuples of attributes, whose range and meaning are explained in detail in Section 2.1.

Definition 1 (Supply) *A supply* s_i *is a tuple*

$$s_i = (l_i, t_i, f_i, c_i, n_i, g_i, u_i, b_i, o_i, v_i), \textit{ with}$$

- $l_i \in \mathbb{N}_0$ *location of the associated dispositive station*
- $t_i \in \mathbb{N}_0$ *(supply) type of the available freight car(s)*
- $f_i \in \mathbb{N}_0$ *wagon keeper*
- $c_i \in \mathbb{N}_0$ *time stamp which defines the time of earliest availability*
- $n_i \in \mathbb{N}_0$ *number of available cars*
- $g_i \in \mathbb{N}_0$ *local transport costs*
- $u_i \in \{0, 1\}$ *specifies if* s_i *can be reused in the productive area* $(u_i = 1)$ *or not* $(u_i = 0)$
- $b_i \in \mathbb{N}_0$ *specifies the single allowed border station (or zero, if the supply has no predefined border station)*
- $o_i \in \mathbb{N}_0$ *specifies the location of the operative storage (or zero, if the supply is not stored)*
- $v_i \in \mathbb{N}_0$ *specifies the location of the predefined operative storage (or zero, if the supply has no predefined storage)*

Definition 2 (Demand) *A demand* d_j *is a tuple*

$$d_j = (l_j, t_j, f_j, c_j, n_j, g_j, a_j, r_j, e_j, z_j, w_j, p_j), \textit{ with}$$

- $l_j \in \mathbb{N}_0$ *location of the associated dispositive station*
- $t_j \in \mathbb{N}_0$ *(demand) car type of the demanded freight car(s)*
- $f_j \in \mathbb{N}_0$ *demanded wagon keeper or zero if none is specifically demanded*
- $c_j \in \mathbb{N}_0$ *time stamp which defines the time of latest possible arrival*
- $n_j \in \mathbb{N}_0$ *number of demanded cars*
- $g_j \in \mathbb{N}_0$ *local transport costs*
- $a_j \in \{0, 1\}$ *specifies if the target location (of the loaded run) is located within the own productive area* ($a_j = 1$) *or not* ($a_j = 0$)
- $r_j \in \mathbb{N}_0$ *target location (of the loaded run) or zero if unknown*
- $e_j \in \mathbb{N}_0$ *first transit company (of the loaded run) or zero if domestic destination*
- $z_j \in \mathbb{N}_0$ *target company (of the loaded run) or zero if domestic destination*
- $w_j \in \mathbb{N}_0$ *cost reduction term (weak prioritization)*
- $p_j \in \{0, 1, 2\}$ *strong prioritization level*

We call a demand d_j *preferred* over another demand d_j' if $p_j > p_j'$. Two additional demand-like types of input data are defined in the following. The border station represents the (artificial or doubled) dispositive station associated with a 'border crossing' between the own and a foreign production area. Border stations represent the return targets for foreign freight cars. The operative storage represents the associated dispositive station to an operative storage track from the demand point of view. Border stations or operative storage are never preferred over demands.

Definition 3 (Border Station) *A border station* b_l *is a tuple*

$$b_l = (l_l, c_l, c_l', n_l, g_l),$$

where l_l, n_l, g_l *are defined as in Definition 2 and* $c_l, c_l' \in \mathbb{N}$ *define a time interval* $[c_l, c_l'[$ *during which the border station can receive* n_l *foreign cars.*

Definition 4 (Early/Late Operative Storage) *An early/late operative storage* o_k'/o_k *is a tuple*

$$o_k' = (l_k, 0, n_k', g_k),$$

$$o_k = (l_k, c_k, n_k, g_k),$$

where l_k, g_k, n_k *are defined as in Definition 2 and* $c_k \in \mathbb{N}$ *is a time stamp indicating the next fetch tour (or zero).*

Note that the meaning of the time stamp c_k differs between demand, border station and operative storage. As motivated in Chapter 2, we consider an 'early' and a 'late' capacity status of each operative storage. The early status holds until the next fetch tour, indicated by c_k and the storage capacity n'_k until then is reduced by those cars in the current optimization process, which are stored at l_k. Even if the latter are sent to somewhere else by the current optimization run, they occupy this capacity until c_k. Therefore the sum of cars assigned to l_k which arrive until c_k must not exceed n'_k, whereas the sum of cars assigned to l_k after c_k is only limited by n_k (see Definition 28).

In the following, we frequently distinguish the data records representing an instance of (DP) in *input data*, which comprises data in the above defined form and *master data*, which is represented as follows. Master data generally provides the constraints of the (DP) and is given by sets of tuples, which for example explicitly represent all available freight train connections or allowed substitutions. We add the definition of an appropriate relation between supply and demand, border station or operative storage, such that the input data fulfills the relation if and only if a corresponding tuple of the set can be found.

Definition 5 (Timetable) *A timetable is a set $\mathcal{T}$ of tuples θ_{sd} ('connections')*

$$\theta_{sd} = (l_s, l_d, c_s, c_d, c_{sd}), \text{ where}$$

- $l_s \in \mathbb{N}_0$ *is the location of the supply*
- $l_d \in \mathbb{N}_0$ *is the location of the demand*
- $c_s \in \mathbb{N}_0$ *is the latest time, a freight car can join the (first train of the) connection*
- $c_d \in \mathbb{N}_0$ *is the earliest time a freight car can leave the (last train of the) connection*
- $c_{sd} \in \mathbb{N}_0$ *is the transport cost for the connection*

We specify the in-time relation corresponding to a timetable $\mathcal{T}$ between supplies and demands, border stations and storage respectively due to the different influence of their time stamps on the relation.

Definition 6 (In-Time: Supply, Demand) *A pair of supply s_i and demand d_j is* in time $\mathrm{it}(s_i, d_j)$ *(with respect to a timetable $\mathcal{T}$), if*

$$\exists \theta_{sd} = (l_s, l_d, c_s, c_d, c_{sd}) \in \mathcal{T},$$

such that $l_i = l_s$, $l_j = l_d$, $c_i \leqslant c_j$, $c_i \leqslant c_s$, $c_j \geqslant c_d$. For each pair of supply and demand with $\mathrm{it}(s_i, d_j)$, we set θ_{sd} to such a connection with smallest c_s according to the first train possible policy.

Definition 7 (In-Time: Supply, Border Station) *A pair of supply* s_i *and border station* b_l *is* in time $it(s_i, b_l)$ *(with respect to a timetable* $\mathcal{T}$*), if*

$$\exists\theta_{sb} = (l_s, l_b, c_s, c_b, c_{sb}) \in \mathcal{T},$$

such that $l_i = l_s$, $l_l = l_b$, $c_i \leqslant c_s$ *and* $c_l \leqslant c_b < c_l'$ *or* $c_l \leqslant c_i < c_l'$ *if* $c_b = 0$. *For each pair of supply and border station with* $it(s_i, b_l)$, *we set* θ_{sb} *to such a connection with smallest* c_s *according to the first train possible policy.*

Definition 8 (In-Time: Supply, Operative Storage) *A pair of supply* s_i *and operative storage* o_k *is* in time $it(s_i, o_k)$ *(with respect to a timetable* $\mathcal{T}$*), if*

$$\exists\theta_{so} = (l_s, l_o, c_s, c_o, c_{so}) \in \mathcal{T},$$

such that $l_i = l_s$, $l_k = l_o$, $c_i \leqslant c_s$ *and* $c_o \geqslant c_k$ *or* $c_k = 0$. *For each pair of supply and storage with* $it(s_i, o_k)$, *we set* θ_{so} *to such a connection with smallest* c_s *according to the first train possible policy.*

Note that $\theta \in \mathcal{T}$ is independent of real freight trains, but corresponds to the current route ('Leitweg') between a pair of dispositive stations (l_A, l_B) and can be composed of more than one train. For each known location l, we expect $\theta_l = (l, \infty, l, 0, c_l)$ to be in the timetable. (A known location l is a dispositive station occurring in a supply, demand, border station or storage.) Thus local allocation of supply s_i and demand d_j at the same dispositive station $l = l_i = l_j$ is always in-time if $c_i \leqslant c_j$ and analogously for border stations and (early) operative storage. The cost term c_l then represents the shunting cost at l which resembles the total real transportation cost between s_i and d_j (b_l, o_k/o_k'). Those costs can neither be taken into account for local transportation cost at supplies or demands (border stations, operative storage) as for locations $l_i \neq l_j$ the transportation cost already include this term (for the supply dispo station) and thus the shunting costs would be counted twice in this case.

Definition 9 (Substitution) *A substitution is a set* $\mathcal{S}$ *of product pairs*

$$\sigma_{sd} = (n_s t_s, n_d t_d),$$

where t_s *is a car type occurring as supply (supply type) and* t_d *is a car type occurring as demand (demand type), such that* n_s *cars of type* t_s *satisfy a demand of* n_d *cars of type* t_d.

Definition 10 (S-Allowance) *A supply* s_i *is* s-allowed *(with respect to a substitution* $\mathcal{S}$*) for a demand* d_j *if*

$$\exists\sigma_{sd} = (n_s t_s, n_d t_d) \in \mathcal{S}, \textit{such that } t_i = t_s \textit{ and } t_j = t_d.$$

Note that $\sigma_{sd} \in \mathcal{S}$ is unique for each pair of supply and demand type. In contrast to timetable and in-time relation, the allowance relation is only defined for supplies and demands to which type substitution refers in the classical sense. Nevertheless, in the following a number of additional allowance relations between supply and demand have to be defined according to the various additional constraints in Chapter 2, as well as allowance between supply and border stations or operative storage respectively. All of them rely on their own set of substitution-like rules instead of the substitution set $\mathcal{S}$ itself.

Definition 11 (Domestic Rule) *A domestic rule is a set $\mathcal{I}$ of triples*

$$\iota = (f, l_s, l_t), \textit{ where}$$

- $f \in \mathbb{N}_0$ *specifies a wagon keeper*
- $l_s \in \mathbb{N}_0$ *is a supply location*
- $l_t \in \mathbb{N}_0$ *is the loaded run destination of a demand*

Definition 12 (I-Allowance) *A supply s_i is* i-allowed *(with respect to a domestic rule $\mathcal{I}$) for a demand d_j if $u_i = 1$, $a_j = 1$ and*

$$\exists \iota = (f, l_s, l_t) \in \mathcal{I}, \textit{ such that}$$

- $f = f_i = f_j$ *or* $f = f_i, f_j = 0$
- $l_i = l_s$
- $r_j = l_t$

Definition 13 (Foreign Rule) *A foreign rule is a set $\mathcal{F}$ of triples*

$$\nu = (f, e, z), \textit{ where}$$

- $f \in \mathbb{N}_0$ *specifies a wagon keeper*
- $e \in \mathbb{N}_0$ *is a first transit company (of the loaded run destination) of a demand*
- $z \in \mathbb{N}_0$ *is a target company (of the loaded run destination) of a demand*

Definition 14 (F-Allowance) *A supply s_i is* f-allowed *(with respect to a foreign rule $\mathcal{F}$) for a demand d_j if $a_j = 0$ and*

$$\exists \nu = (f, e, z) \in \mathcal{F}, \textit{ such that}$$

- $f = f_i = f_j$ *or* $f = f_i, f_j = 0$
- $e = e_j$

- $z = z_j$

Definition 15 (Border Rule) *A border rule is a set $\mathcal{B}$ of triples*

$$\beta = (b, f, t), \text{ where}$$

- $b \in \mathbb{N}_0$ *specifies a border station*
- $f \in \mathbb{N}_0$ *specifies a wagon keeper*
- $t \in \mathbb{N}_0$ *specifies a car type*

Definition 16 (B-Allowance) *A supply* s_i *is* b-allowed *(with respect to a border rule $\mathcal{B}$) for a border station* b_l *if*

- $b_i = b_l$
  or
- $b_i = 0$ *and* $\exists b = (b, f, t) \in \mathcal{B}$*, such that*
  - $b = b_j$
  - $f = f_i$ *or* $f = 0$
  - $t = t_i$

Definition 17 (Storage Rule) *A storage rule is a set $\mathcal{O}$ of triples*

$$\omega = (o, f, t), \text{ where}$$

- $o \in \mathbb{N}_0$ *specifies an operative storage*
- $f \in \mathbb{N}_0$ *specifies a wagon keeper*
- $t \in \mathbb{N}_0$ *specifies a car type*

Definition 18 (O-Allowance) *A supply* s_i *is* o-allowed *(with respect to a storage rule $\mathcal{O}$) for an operational storage* o_j *if*

- $v_i = o_j$
  or
- $v_i = 0$ *and* $\exists o = (o, f, t) \in \mathcal{O}$*, such that*
  - $o = o_j$
  - $f = f_i$ *or* $f = 0$
  - $t = t_i$ *or* $t = 0$

So far, we formally defined the input of a (DP) instance, which can now can be stated as:

Definition 19 (Instance of the (DP)) *An instance of the (DP)*

$$I = \{S, D, O, B, \mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$$

is a tuple of possibly empty sets of supplies S*, demands* D*, operative storage* O*, border stations* B *and possibly empty sets* $\mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}$*.*

We call an instance of the (DP) *homogeneous*, if $n_s = n_d = 1$ for all $\sigma_{sd} \in \mathcal{S}$ and heterogeneous otherwise. A M-heterogeneous instance with $M \subset \mathbb{N}$ is an instance such that $n_s, n_d \in M$ for all $\sigma_{sd} \in \mathcal{S}$ and for $M = \{1, m \neq 1\}$ we also speak of an m-heterogeneous instance. Remember that $\sigma = (n_s t_s, n_d t_d) \in \mathcal{S}$ means that n_s cars of type t_s can substitute n_d cars of type t_d. Thus we are generally interested in the ratio or 'exchange-rate' of n_s and n_d. Let T_S (T_D) be the set of car types t_s (t_d) such that $\sigma_{sd} \in \mathcal{S}$ for some car type t_d (t_s) with respect to an instance I.

Definition 20 (Relative Valency) *The* relative valency $\nu(t_s, t_d)$ *of the supply car type* $t_s \in T_S$ *to the demand car type* $t_d \in T_D$ *is:*

$$\nu(t_s, t_d) = \frac{n_d}{n_s}.$$

The relative valency of t_d *to* t_s *is defined as the reciprocal.*

Definition 21 (Total Valency) *The* total valency *of a car type* $t \in T_S \cup T_D$ *is only defined if there is a value* $\nu(t)$*, such that:*

$$\forall \sigma_{sd} \in \mathcal{S}, t_s = t : \nu(t_s, t_d) = \nu(t).$$

Then $\nu(t)$ *is the total valency of* t*. We call an instance* I *of the (DP) total, if each supply type* $t \in T_S$ *has a total valency* $\nu(t)$*.*

The solution of an instance I of the (DP) is the assignment of all supplies $s_i \in S$ to demands $d_j \in D$, operative storage $o_k \in O$ or border stations $b_l \in B$. (We frequently summarize the latter as *pseudo demands* $\hat{d}_j \in D \cup O \cup B$.) The global allocation is called a *distribution*, which consists of individual *assignments*. As neither supplies nor (pseudo) demands are restricted to single cars, we allow a supply to be allocated to several (pseudo) demands and (pseudo) demands can be assigned several assignments of cars. Therefore, in addition to the respective supply and (pseudo) demand each assignment requires the number of actually assigned cars. We formalize a distribution of cars from a supply to a (pseudo) demand as follows.

Definition 22 (Assignment) *An assignment* δ_{ij} *(with respect to* I*) is a tuple*

$$d_{ij} = (s_i, \hat{d}_j, n_{ij})$$

with $s_i \in S$, $\hat{d}_j \in D \cup O \cup B$ *and* $n_{ij} \in \mathbb{R}$ *is the number of cars sent from* s_i *to* $\hat{d}_j$*.*

Definition 23 (Distribution) *A distribution (with respect to* I*) is a set* $\mathcal{D}$ *of assignments* δ *(with respect to* I*).*

An assignment is feasible (with respect to a (DP) instance), if the assignment satisfies all constraints given by the sets $\mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}$ of the instance. We introduce the notion of *matching* supplies and (pseudo) demands to simplify notation when we formalize the feasibility of assignments/distributions.

Definition 24 (Match of Supply, Demand) *A supply* s_i matches *a demand* d_j *(with respect to* I*) if* $\mathrm{it}(s_i, d_j)$*,* s_i *is s-allowed for* d_j *and either* s_i *is i-allowed for* d_j *or* s_i *is f-allowed for* d_j.

Definition 25 (Match of Supply, Border Station) *A supply* s_i matches *a border station* b_l *(with respect to* I*) if and only if* $\mathrm{it}(s_i, b_l)$ *and* s_i *is b-allowed for* b_l.

Definition 26 (Match of Supply, Storage) *A supply* s_i matches *an early/late operative storage* o'_k/o_k *(with respect to* I*) if and only if* $\mathrm{it}(s_i, o'_k)$ *or* $\mathrm{it}(s_i, o_k)$ *respectively and* s_i *is o-allowed for* o'_k/o_k.

Obviously, feasible assignments δ_{ij} and distributions $\mathcal{D}$ also require integrality of the number of cars n_{ij}. Especially with respect to the heterogeneous (DP), we will also consider assignments/distributions for which this is not true. The latter can be seen as solutions to a (LP)-relaxation of the respective (DP) instances.

Definition 27 ((LP)-feasible Assignment) *An assignment* $\delta_{ij} = (s_i, \hat{d}_j, n_{ij})$ *is (LP)-feasible with respect to* I *if and only if* s_i *matches* $\hat{d}_j$*,* $n_{ij} \in \mathbb{R}$*,* $n_i \geqslant n_{ij}$ *and* $n_j \leqslant \nu(t_i, t_j) n_{ij}$.

Definition 28 ((LP)-feasible Distribution) *A distribution* $\mathcal{D}$ *is (LP)-feasible with respect to* I *if and only if each assignment* $\delta \in \mathcal{D}$ *is (LP)-feasible and*

- $\forall s_i \in S : \sum_{\delta_{ij} \in \mathcal{D}} n_{ij} = n_i$
- $\forall d_j \in D : \sum_{\delta_{ij}) \in \mathcal{D}} \nu(t_i, t_j) n_{ij} \leqslant n_j$
- $\forall o_k \in O : \sum_{\delta_{ik} \in \mathcal{D}} n_{ik} \leqslant n_k$
- $\forall o'_k \in O : \sum_{\delta_{ik} \in \mathcal{D}, c_d < c_k} n_{ik} \leqslant n'_k$,
  where $\theta_{ik} = (l_s = l_i, l_d = l_k, c_s, c_d, c_{sd}) \in \mathcal{T}$ *is the first possible connection for assignments* (s_i, o_k, n_{ik}) *and* $n'_k = n_k - \sum_{s_i \in S : o_i = o_k} n_i$.
- $\forall b_l \in S : \sum_{\delta_{il} \in \mathcal{D}} n_{il} \leqslant n_l$

- $\forall d_j \in D$ *such that* $\sum_{\delta_{ij})\in\mathcal{D}} \nu(t_i, t_d)n_{ij} < n_j$:
  There does not exist $(s_i, d_j', n_{ij'}) \in \mathrm{mathcalD}$ *such that* d_j *is preferred over* d_j'
  and match(s_i, d_j*).*

We refer to the last constraint as *strong priority order*. We call a demand d_j with $\nu(d_j) = \sum_{\delta_{ij}\in\mathcal{D}} \nu(t_i, t_j)n_{ij} - n_j$ *undersatisfied* (by $\mathcal{D}$) if $\nu(d_j) < 0$ and *oversatisfied* (by $\mathcal{D}$) if $\nu(d_j) > 0$.

An assignment δ_{ij} is *feasible*, if it is (LP)-feasible and $n_{ij} \in \mathbb{N}$, and a distribution $\mathcal{D}$ is *feasible*, if it is (LP)-feasible and solely contains feasible assignments.

Observe that a distribution which causes undersatisfied demands is feasible, but a distribution which causes oversatisfied demands is not. This imbalance stems from the fact that supplies may be too short to satisfy all demands and still a distribution satisfying as much demand as possible with smallest possible cost shall be found. On the other hand, even if not all supply can be allocated to actual demands, storage capacities can be assumed to be sufficient such that each car is assigned. Traditionally there were undersatisfied demands as well as unassigned supplies after carrying out the best distribution, as supply was not distributed actively into storage. In the implementation practice, supply can still remain unallocated indicating a serious problem in the input data, which is to be detected and reported. For theoretical considerations we will adopt the above assumption of sufficient storage. Furthermore, each supply s_i has to be in time and o-allowed for appropriate amount of storage. Otherwise, we call s_i *isolated*.

The cost of an assignment/distribution is partly, but not completely determined by the transport cost:

Definition 29 (Cost of an Assignment/Distribution) *Let* $\theta_{sd} \in \mathcal{T}$ *be the connection, such that* $\mathrm{it}(s_i, \hat{d}_j)$. *Then the cost* $c(\delta_{ij})$ *of the assignment is defined as*

- *the sum of the transport and the local transport costs, if* $\hat{d}_j \in O \cup B$*:*

$$c(\delta_{ij}) = n_{ij}c_{ij} = n_{ij}(c_{sd} + g_i + \hat{g}_j)$$

- *the sum of the transport and the local transport costs and the difference between the maximum weak prioritization cost term and the weak prioritization cost term of the demand, if* $d_j \in D$*:*

$$c(\delta_{ij}) = n_{ij}c_{ij} = n_{ij}(c_{sd} + g_i + g_j + (\max_{d_j\in D} w_j - w_j))$$

The cost $c(\mathcal{D})$ *of a distribution* $\mathcal{D}$ *is the sum of the cost of each individual assignment:*

$$c(\mathcal{D}) = \sum_{\delta\in\mathcal{D}} c(\delta).$$

Although the effect of the weak prioritization is a cost reduction for assignments, we do not interpret it directly as a negative cost term to further ensure non-negativity of all assignment costs. With the above definitions at hand, we can now fully specify the distribution problem.

Definition 30 (Distribution Problem (DP)) *Given an instance* I *of the (DP), find a feasible distribution* $\mathcal{D} = \mathcal{D}(I)$ *with minimum cost* $c(\mathcal{D})$.

In the context of computational complexity, we consider the following decision variant of the distribution problem. (The solution to the decision problem is a 'yes/no' answer.)

Definition 31 (Decision Distribution Problem (DDP)) *Given an instance* I *of the (DP) and a (non-negative, integral) cost value* c_I, *is there a feasible distribution* $\mathcal{D} = \mathcal{D}(I)$ *with minimum cost* $c(\mathcal{D}) \leqslant c_I$?

Note that a solution to the (DP) can theoretically be obtained by solving a series of (DDP)s as follows: Determine the minimum c_I, such that the answer to the decision problem is 'yes' by binary search on the interval of possible values c_I. In our case (with non-negative costs), the natural lower bound on c_I is zero and a (naive) upper bound is given by the sum of the costs of all feasible assignments $\delta = (s_i, \hat{d}_j, n_{ij})$ with maximum n_{ij}.

Further, we consider a special type of approximation to the (DP). For this, we call a 1-to-1-exchange of a car of type t_a with a car of type t_b an *upgrade* with respect to a demand d_j of type t_c if $\sigma_{ac}, \sigma_{bc} \in \mathcal{S}$ and $\nu(t_a, t_c) < \nu(t_b, t_c)$. Such an exchange is called a *downgrade* if $\sigma_{ac}, \sigma_{bc} \in \mathcal{S}$ and $\nu(t_a, t_c) > \nu(t_b, t_c)$. Note that an upgrade or downgrade is a 1-to-1-exchange of cars, independent of relative or total valencies.

Definition 32 (ν-Upgrade) *We call a distribution* $\mathcal{D}$ *ν-upgrade(d) for an instance* I *of the (DP), if it is feasible for* I' *with* $n'_j = n_j + \nu$ *for all demands* $d'_j \in D'$ *of* I' *(and all other data equal to* I*) and after a downgrade of a single car for each demand* $d_j \in D$ *oversatisfied by* $\mathcal{D}$ *with respect to* I *we obtain a feasible distribution for* I.

Observe that downgrading does not increase the cost. An optimal ν-upgraded distribution may well be applicable in practice as well as an optimal distribution, as it only provides some customers with a single car of higher quality than ordered without charging extra-cost.

4.2 COMPUTATIONAL COMPLEXITY OF THE (DP)

We prove that the decision distribution problem as in Definition 31 is NP-complete by reduction of a variant of the well-known satisfiability problem [47]. Consider the following SAT type problem, which was shown to be NP-complete in [114]:

[3V2L3SAT] Given a Boolean formula

$$\alpha = C_1 \wedge \cdots \wedge C_n, C_i = l_{i1} \vee l_{i2} \vee l_{i3},$$

$$l_{ij} \in \{v_k, \neg v_k | 1 \leqslant k \leqslant m\} \cup \{0\}$$

where each variable v_k can only occur 3 times in total and maximal 2 times as one of the corresponding literals. Decide whether there is a truth assignment satisfying α.

The reduction of 3SAT to 3V2L3SAT introduces k new variables $x_1, \ldots, x_k$ for k occurrences of a variable x and exchange the ith occurrence of x by x_i. Then the clauses $(x_i \vee \neg x_{i+1}), 1 \leqslant i < k$, and $(x_k \vee \neg x_1)$ are added to the original formula, which ensure an equal truth assignment on all x_i. To reduce 3V2L3SAT to the (DP), we construct an instance I_α, given a 3V2L3SAT formula α with m variables in n clauses, such that the assignment of all supply of I_α is possible with cost smaller or equal to $c_I = 0$ if and only if α is satisfiable.

I_α is a total 2-heterogeneous instance, which is defined as follows. Let O, B and consequently $\mathcal{O}, \mathcal{B}$ be empty sets. For the moment, we set a number of supply attributes to zero:

$$s_i = (l_i, t_i, 0, 0, n_i, 0, 0, 0, 0, 0).$$

We abbreviate such a supply by $s_i = (l_i, t_i, 0, 0, n_i)$. Analogously we set the following demand attributes to zero:

$$d_j = (l_j, t_j, 0, 0, n_j, 0, 0, 0, 0, 0, 0, 0).$$

We abbreviate such a demand by $d_j = (l_j, t_j, 0, 0, n_j)$.

The set S composes of two subsets S_c and S_v with supplies that correspond to clauses and variables in α. For each clause c_i in α the set S_c contains one supply s_i and for each variable in α the set S_v contains three supplies s_j^1, s_j^2, s_j^3, which are defined as follows:

- $S_c = \{s_i = (l_i, t_1, 0, 0, 1) | 1 \leqslant i \leqslant n\}$
- $S_v =$

$$\begin{aligned} & \{s_k^1 = (l_k^1, t_1, 0, 0, 2) | 1 \leqslant k \leqslant m\} \\ \cup \ & \{s_k^2 = (l_k^2, t_2, 0, 0, 1) | 1 \leqslant k \leqslant m\} \\ \cup \ & \{s_k^3 = (l_k^3, t_1, 0, 0, 2) | 1 \leqslant k \leqslant m\} \end{aligned}$$

We set $S = S_c \cup S_v$. Further, four demands are specified for each variable in x_j in α, two corresponding to the (up to two) occurrences of x_j as a literal and the other two to the occurrences of $\neg x_j$, respectively:

$$\begin{aligned} D &= \\ & \{d_j^1 = (l_j^1, t_3, 0, 0, 1) | 1 \leqslant k \leqslant m\} \cup \{d_j^2 = (l_j^2, t_3, 0, 0, 1) | 1 \leqslant k \leqslant m\} \\ \cup & \{\bar{d}_j^1 = (\bar{l}_j^1, t_3, 0, 0, 1) | 1 \leqslant k \leqslant m\} \cup \{\bar{d}_j^2 = (\bar{l}_j^2, t_3, 0, 0, 1) | 1 \leqslant k \leqslant m\}. \end{aligned}$$

We define the set of substitution rules as

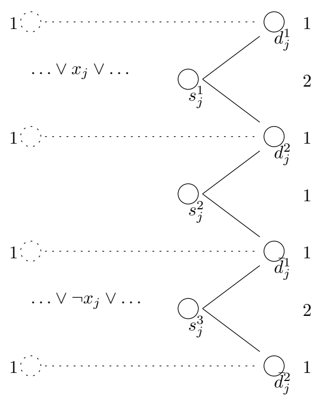
$$\mathcal{S} = \{(2t_1, t_3), (t_2, t_3)\}.$$

(As all occurring demands are of type t_3 we can also enlarge the set by the 'natural' extension $\{(t_1, t_1), (t_2, t_2)\}$ avoiding the impression of degeneration. On the other hand I_α is then not generically total, such that the result is weakend.) Note that thus any supply $s_i \in S$ is s-allowed for each demand $d_j \in D$. Further, we define the timetable $\mathcal{T}$ as follows:

$$\begin{aligned} \mathcal{T} &= \{(l_i, 0, l_j^1, \infty, 0) | \text{if } c_i \text{ is the first clause containing } x_j\} \\ &\cup \{(l_i, 0, l_j^2, \infty, 0) | \text{if } c_i \text{ is the second clause containing } x_j\} \\ &\cup \{(l_i, 0, \bar{l}_j^1, \infty, 0) | \text{if } c_i \text{ is the first clause containing } \neg x_j\} \\ &\cup \{(l_i, 0, \bar{l}_j^2, \infty, 0) | \text{if } c_i \text{ is the second clause containing } \neg x_j\} \\ &\cup \{(l_k^1, 0, l_j^1, \infty, 0), (l_k^1, 0, l_j^2, \infty, 0) | k = j\} \\ &\cup \{(l_k^2, 0, l_j^2, \infty, 0), (l_k^2, 0, \bar{l}_j^1, \infty, 0) | k = j\} \\ &\cup \{(l_k^3, 0, \bar{l}_j^1, \infty, 0), (l_k^3, 0, \bar{l}_j^2, \infty, 0) | k = j\} \\ &\cup \{(l_a, 0, l_b, \infty, M) | \text{for each other combination of locations}\} \end{aligned}$$

Note that all s_i and d_j are in time with respect to $\mathcal{T}$, while only some corresponding connections have associated cost zero. We set the domestic rule $\mathcal{I}$ to the empty set. As all demands d_j carry the attribute $a_j = 0$ this has no effect as all demands have foreign load run targets. We define the foreign rule as $\mathcal{F} = \{(0, 0, 0)\}$, such that each supply in S matches each demand in D.

We can visualize matching supplies and demands corresponding to a single variable x_j in α as shown in Figure 7: Each supply and each demand corresponds to a solid vertex. Solid lines connect vertices corresponding to matching supplies and demands with associated costs zero and the numbers at the vertices represent the number of available and demanded cars respectively. Further, the up to three clauses in α containing either x_j or $\neg x_j$ are indicated by the dotted vertices in correspondence to associated supplies. The dotted lines also represent connections with associated cost zero. Figure 7 shows four dotted vertices, as S contains up to two supplies



1
$\ldots \vee x_j \vee \ldots$
$\ldots \vee \neg x_j \vee \ldots$
s_j^1
s_j^2
s_j^3
d_j^1
d_j^2
$\bar{d}_j^1$
$\bar{d}_j^2$
2

Figure 7: Matching supply and demand associated with a variable x_j and its occurrence in clauses of α.

associated with a clause containing x_j (upper part of the figure) and up to two supplies associated with a clause containing $\neg x_j$ (lower part of the figure). Still the total number of such supplies in S_c is three.

Lemma 33 *The total supply $e_v = \sum_{s_i \in S_v} n_k = 5m$ of S_v can only be distributed with cost zero by either of the following distributions for each variable x_j:*

1. *Send two cars of s_k^1 to d_j^1, one car of s_j^2 to d_j^2 and the two cars of s_k^3 either both to $\bar{d}_j^1$ or $\bar{d}_j^2$ or one to each of the latter.*
2. *Send two cars of s_k^3 to $\bar{d}_j^2$, one car of s_j^2 to $\bar{d}_j^1$ and the two cars of s_k^1 either both to d_j^1 or d_j^2 or one to each of the latter.*

Proof: The one car of s_k^2 can either be assigned to d_j^2 or $\bar{d}_j^1$ with zero cost and satisfies either of them completely. Otherwise it cannot be distributed with zero cost. Consequently, either s_k^1 is completely assigned to d_j^1 or s_k^3 is completely allocated to $\bar{d}_j^2$ respectively, as otherwise the supply cannot be distributed with zero cost. The respective other supply can (but need not) be distributed to the respective other demands with zero cost. □

Note that in either case, the total demand of four demands associated with a variable x_j is not satisfied. We associate the only two possible distributions with zero cost per variable with its two possible truth values. The first distribution variant, satisfying both demands d_j, corresponds to 'x_j is false'. The second variant, satisfying both demands $\bar{d}_j$, corresponds to 'x_j is true'. This notion is chosen due to the following observation: The first variant allows no supply c_i matching to a demand d_j with zero cost to be assigned to it, whereas the second allows both of the possibly two matching supplies to be distributed with zero cost, each to either of the demands. The latter distribution corresponds to the truth assignment of x_j satisfying the clause c_i.

The same argumentation holds for clauses containing $\neg x_j$ and the exchanged roles of both distribution variants. Given that any supply corresponding to a clause c_i can only be distributed with zero cost to one of the demands corresponding to a literal which occurs in c_i, a distribution, which allocates the total supply of $e = e_v + e_c = 5m + n$ with zero cost corresponds to a satisfying truth assignment. We formally show:

Theorem 34 *Let* $I_\alpha = \{S, D, O, B, \mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$*, as defined above. The formula* α *is satisfiable if and only if all supply* $e = 5m + n$ *can be distributed with zero cost.*

Proof: Suppose α is satisfiable. Distribute e_v for each x_j by the variant of Lemma 33, which corresponds to the satisfying truth assignment of x_j. Thus e_v is completely allocated at zero costs and only demands corresponding to true literals are not yet completely satisfied. As the truth assignment satisfies α, it satisfies every clause and consequently each corresponding supply c_i is at least a match with a connection of zero associated cost for one not yet satisfies demand. Send c_i to the latter demand, which corresponds to a true literal. Thus the additional unit supply for each clause $e_c = n$ is also disposable at zero cost. On the other hand, suppose α is not satisfiable. Each distribution of e_v with zero cost according to the variants in Lemma 33 corresponds to a consistent truth assignment and therefore blocks at least one clause from all matching demands with associated connections of zero cost, such that at most $5m + n - 1$ supply can be distributed at zero cost. If on the other hand, we allow any other distribution of e_v, by Lemma 33, not all supply e_v can be assigned at zero costs and any complete distribution of the total supply $5m + n$ needs at least cost of $M > c_I$. □

As we can check in polynomial time for a given distribution if it allocates the total supply at zero costs and does not violate any constraints, it follows:

Corollary 35 *The (DDP) is* NP*-complete for total 2-heterogeneous instances.*

Note that by omitting connections with cost M in the timetable, the upper construction allows to prove the existence of a feasible distribution if and only if the formula α is satisfiable. Thus the DDP is strongly NP-complete, which means that the complexity of the problem does not depend on the values of the cost function, but remains hard for unit costs. We chose the variant with distinguished costs in order to keep the prove closely related to the optimization problem.

5 A NETWORK MODEL FOR THE HOMOGENEOUS DP

In this chapter, we derive a classical network model N_I for homogeneous instances I of the (DP) (Section 5.1). We argue that for this setting an integral minimum cost flow $f^*(N_I)$ translates into a solution to the (DP) instance I disregarding strong priorities (Section 5.2). We obtain such a flow $f^*(N_I)$ in polynomial time [39], as all cost, capacity and balance values are integral or can be scaled to integers if necessary.

The minimum cost flow problem is a long known and well studied graph theoretical problem (see [2] for a broad survey). In Section 5.3 we briefly survey flow algorithms and motivate our choice of the *Successive Shortest Path* algorithm to solve the minimum cost flow problem on instances N_I. We conclude this chapter by a modification to the *Successive Shortest Path* algorithm which guarantees strong priority order.

5.1 A CLASSICAL FLOW NETWORK

Let $I = \{S, D, O, B, \mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$ be a homogeneous instance of the (DP). We construct a network model $N_I = (V_I, A_I)$ and define capacities/costs on arcs as well as a balance function on nodes as follows (see Figure 8).

The node set V_I of the network composes of *supply nodes* i and *(pseudo) demand nodes* j corresponding to the supplies $s_i \in S$ and (pseudo) demands $\hat{d}_j \in D \cup O \cup B$ respectively. For each node k corresponding to an operative storage o_k, we add a second node k' to V_I. The latter also corresponds to the storage o_k except that the timestamp c'_k is set to zero.

Further, a source node s and three sink nodes t_0, t_1, t_2 corresponding to the three strong priority levels belong to V_I. The source and sink nodes are referred to as *model nodes*.

The arc set A_I can intuitively be decomposed into three 'layers' of the network: the supply layer A_S, the transit layer A_T and the demand layer A_D. The set A_S contains all arcs (s, i) with cost $c_{si} = 0$ and capacity $u_{si} = n_i$, the number of freight cars available for s_i. The set A_D contains all arcs (j, t_p) with $p = p_j \in \{0, 1, 2\}$ according to the strong priority level of $d_j \in D$ or t_0 if $\hat{d}_j \in O \cup B$. The cost c_{jt_p} is also set to zero for the arcs of A_D and the capacity $u_{jt_p} = n_j$ equals the number of demanded cars of d_j.

The arc set A_T, referred to as *transit arcs*, comprises all arcs (i, j) between supply nodes i and (pseudo) demand nodes j, such that s_i matches $\hat{d}_j$. The cost c_{ij} is the cost of a unit assignment $\delta_{ij} = (s_i, \hat{d}_j, 1)$ and the capacity u_{ij} is set to infinity. Additionally A_T contains arcs of the type (k', k), correspond-

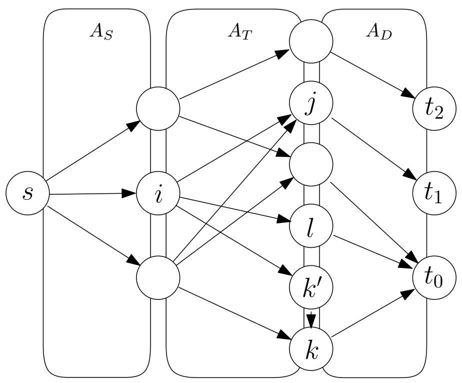


Figure 8: The network N_I contains a source s and three sinks $t_p, p = 1, 2, 3$, a node i, j, l for each $s_i \in S$, $d_j \in D$, $b_l \in B$ and two nodes k', k for each $o_k \in O$. Source s is connected to all i by arcs from A_S (capacity n_i, cost 0) and each node j, l, k is connected via arcs of A_D (capacity n_j, n_l, n_k, cost 0) to one of the sinks. There are also arcs $(k', k) \in A_D$ (capacity n'_k, cost 0). The arcs set A_T (capacity ∞, cost c_{ij}, c_{il}, c_{ik}) connects nodes i to nodes j, l, k', k according to the appropriate matching relation.

ing to operative storage $o_k \in O$. The cost $c_{k'k}$ are zero and the capacity $u_{k'k}$ is given by the rest capacity $n'_k = n_k - \sum_{s_i \in S: o_i = o_k} n_i$ of the early storage stage.

Finally, we define the excess at s as $b(s) = \sum_{(s,i) \in A_I} u_{si}$; the excess at the sinks $t_p, p \in \{0, 1, 2\}$ is defined as $b(t_p) = -\sum_{(j,t_p) \in A_I} u_{jt_p}$.

Definition 36 (Distribution Network N_I) *Let*

$$I = \{S, D, O, B, \mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$$

be a homogeneous instance of the (DP). Then the distribution network $N_I = (V_I, A_I)$ *with capacity* $u : A_I \to \mathbb{N}$, *cost* $c : A_I \to \mathbb{N}$ *and balance* $b : V_I \to \mathbb{N}$ *is defined as described above.*

5.2 MINIMUM COST FLOW AND DISTRIBUTION

We show that, if a feasible distribution for I exists, an integral minimum cost flow $f^* = f^*(N_I)$ in N_I provides a solution to the (DP) on I disregarding strong priorities. The situation that no feasible distribution exists corresponds to the existence of unassigned supply. With sufficient overall storage capacity and non-isolated supplies, we can always find a feasible distribution for I. Regarding the flow model, we assume that a maximum flow in N_I has been precomputed and the source and sink balances have been adjusted such that a minimum cost flow f^* exists with

$$|f^*| = b(s) = \sum_{p \in \{0,1,2\}} -b(t_p).$$

For each feasible integral flow f in N_I, we can identify each positive flow f_{ij} on a transit arc $a = (i, j)$ with a so-called derived assignment δ^f_{ij}. These assignments can be combined to a derived distribution $\mathcal{D}^f$.

Definition 37 (Derived Assignment/Distribution) *Let f be an integral feasible flow in N_I associated to a homogeneous (DP) instance I. Then the derived assignment from the flow f_{ij} on $a = (i, j) \in A_T$ is:*

$$d^f_{ij} = \begin{cases} \delta^f_{ij} = (s_i, \hat{d}_j{}', n_{ij} = f_{ij}) \textit{ with } \hat{d}_j{}' = o_k \textit{ if } j = k' \\ \delta^f_{ij} = (s_i, \hat{d}_j, n_{ij} = f_{ij}), \textit{ otherwise.} \end{cases}$$

The derived distribution $\mathcal{D}^f$ is the set of all derived assignments with $n_{ij} > 0$:

$$\mathcal{D}^f = \{d^f_{ij} | f_{ij} > 0\}.$$

The identification of freight cars with units of flow obviously requires integrality of f. Moreover, we have to show that derived assignments/distributions from integral feasible flows are generally feasible. We do so by the following two Lemmas 38, 39.

Lemma 38 (Feasible Assignment δ^f_{ij}) *Let f be a feasible integral flow in N_I. Then a derived assignment δ^f_{ij} is feasible with respect to I.*

Proof: By Definition 27, an assignment is feasible if and only if s_i matches $\hat{d}_j$ or $\hat{d}_j{}'$ respectively, $n_{ij} \leqslant n_i$, $n_{ij} \leqslant \nu(t_i, t_j)n_j$ and $n_{ij} \in \mathbb{N}$. Note that $\nu(t_i, t_j) = 1$, as I is homogeneous. As f is an integral feasible flow in N_I, by network construction $a = (i, j) \in A_T$ and thus $f_{ij} > 0$ only if s_i matches $\hat{d}_j$ or $\hat{d}_j{}'$ respectively with respect to I and $n_{ij} = f_{ij} \in \mathbb{N}$. Further, $n_{ij} = f_{ij} \leqslant f_{s,i} \leqslant u(s, i) = n_i$ because (s, i) is the only incoming arc to i. We distinguish four cases for the pseudo demands $\hat{d}_j, \hat{d}_j{}'$, respectively:

1. Let $\hat{d}_j = d_j \in D$. The arc (j, t_p) is the only outgoing arc from j in N_I and $f_{ij} \leqslant f_{jt_p} \leqslant u(jt_p) = n_j, t_p = p_j$ from which follows $\nu(t_i, t_j)n_{ij} \leqslant n_j$, as $\forall t_i, t_j : \nu(t_i, t_j) = 1$ for the homogeneous instance I.

2. Let $\hat{d}_j = o_k \in O$. The arc (k, t_0) is the only outgoing arc from k in N_I and $f_{ik} \leqslant f_{kt_0} \leqslant u(k, t_0) = n_k$ from which follows $n_{ij} \leqslant n_k$.

3. Let $\hat{d}_j{}' = o_k \in O$. The arc (k, k') is the only outgoing arc from k in N_I and $f_{ik} \leqslant f_{kk'} \leqslant f_{kt_0} \leqslant u(k, t_0) = n_k$, as again (k, t_0) is the only outgoing arc from k in N_I. It follows $n_{ij} \leqslant n_k$.

4. Let $\hat{d}_j = b_l \in B$. The arc (l, t_0) is the only outgoing arc from l in N_I and $f_{il} \leqslant f_{lt_0} \leqslant u(l, t_0) = n_l$ from which follows $n_{ij} \leqslant n_l$.

Thus δ^f_{ij} is feasible with respect to I. □

Lemma 39 (Feasible Distribution $\mathcal{D}^f$) *Let* f *be a feasible integral flow in* N_I. *Then a derived distribution* $\mathcal{D}^f$ *is feasible with respect to* I *disregarding strong priorities.*

Proof: By Definition 28, a distribution $\mathcal{D}$ is feasible (disregarding strong priority) if all assignments $\delta_{ij} = (s_i, \hat{d}_j, n_{ij}) \in \mathcal{D}$ are feasible, the sum of allocated cars from each supply s_i equals n_i and the sum of cars assigned to each (pseudo) demand $\hat{d}_j$ (weighted with the relative valency in case of demands) does not exceed n_j. All assignments $\delta \in \mathcal{D}$ are feasible by Lemma 38. Further, the capacities of the single incoming and outgoing arcs at nodes i and j, k, k' and l respectively together with the capacity and flow conservation constraints ensure:

$$\sum_{\delta_{ij} \in \mathcal{D}^f} n_{ij} = \sum_{(i,j) \in A_T} f_{ij} \leqslant u(s,i) = n_i \qquad \forall s_i \in S$$

$$\sum_{\delta_{ij} \in \mathcal{D}^f} \nu(t_i, t_j) n_{ij} = \sum_{(i,j) \in A_T} f_{ij} \leqslant u(j, t_{p_j}) = n_j \qquad \forall d_j \in D$$

$$\sum_{\delta_{ik} \in \mathcal{D}^f} n_{ik} = \sum_{(i,k) \in A_T} f_{ik} + \sum_{(i,k') \in A_T} f_{ik'} \leqslant u(k, t_0) = n_k \qquad \forall o_k \in O$$

$$\sum_{\delta_{ik} \in \mathcal{D}^f, c_d < c_k} n_{ik} = \sum_{(i,k') \in A_T} f_{ik'} \leqslant u(k', k) = n'_k \qquad \forall o_k \in O$$

$$\sum_{\delta_{il} \in \mathcal{D}^f} n_{il} = \sum_{(i,l) \in A_T} f_{il} \leqslant u(l, t_0) = n_l \qquad \forall b_l \in B$$

As f is feasible, we have $|f| = b(s) = \sum_{(s,i) \in A_S} u(s,i) = \sum_{s_i \in S} n_i$ and all flow from source s to sinks $t_p, p = 0, 1, 2,$ passes a unique node i through (s,i) with $u(s,i) = n_i$. Therefore all supply is distributed and $\mathcal{D}^f$ is feasible with respect to I disregarding strong priorities. □

Conversely, we can also define a derived flow $f^{\mathcal{D}}$ in N_I from a feasible distribution $\mathcal{D}$ with respect to I:

Definition 40 (Derived Flow) *Let $\mathcal{D} = \mathcal{D}(I)$ be a feasible distribution with respect to* I. *Then we define a derived flow* $f^{\mathcal{D}}(u,v)$ *in* N_I *on arcs* $a = (u,v) \in A_I$ *as:*

$$f^{\mathcal{D}}(u,v) = \begin{cases} n_i, \textit{ for } u = s, v = i \\ n_{ij}, \textit{ for } u = i, v = j, \textit{ if } \delta_{ij} \in \mathcal{D} \\ n_{il}, \textit{ for } u = i, v = l, \textit{ if } \delta_{il} \in \mathcal{D} \\ n_{ik}, \textit{ for } u = i, v = k, \textit{ if } \delta_{ik} \in \mathcal{D}, c_o \geqslant c_k \\ n_{ik}, \textit{ for } u = i, v = k', \textit{ if } \delta_{ik} \in \mathcal{D}, c_o < c_k \\ \sum_{\delta_{ij} \in \mathcal{D}, j=u} v(t_i, t_j) n_{ij}, \textit{ for } u = j, v = t_{p_j} \\ \sum_{\delta_{il} \in \mathcal{D}, l=u} n_{il}, \textit{ for } u = l, v = t_0 \\ \sum_{\delta_{ik} \in \mathcal{D}, k=u} n_{ik}, \textit{ for } u = k, v = t_0 \\ \sum_{\delta_{ik} \in \mathcal{D}, c_d < c_k, k'=u} n_{ij}, \textit{ for } u = k', v = k. \end{cases}$$

With the help of following Lemma 41, we show the equivalence of feasible integral flows in N_I and feasible distributions for I (disregarding strong priority).

Lemma 41 (Feasible Flow $f^{\mathcal{D}}$) *Let $\mathcal{D} = \mathcal{D}(I)$ be a feasible distribution with respect to* I *and* $f^{\mathcal{D}}$ *a derived flow from* $\mathcal{D}$. *Then* $f^{\mathcal{D}}$ *is an integral feasible flow in* N_I.

Proof: We check capacity constraints on each arc $a \in A_I$ and flow conservation constraints on each node $v \in V_I$ for $f^{\mathcal{D}}$. Arcs $(i,j) \in A_T$ possess infinite capacity and capacity constraints obviously hold for $f^{\mathcal{D}}$ on those arcs. Due to the feasibility of $\mathcal{D}$, each supply is completely distributed, such that $n_i = f^{\mathcal{D}}(s,i) \leqslant u(s,i) = n_i$. Hence, capacity constraints hold for arcs $(s,i) \in A_S$ along with node balance constraints for the source s and all nodes $i, s_i \in S$. For arcs $(j, t_p) \in A_D, p \in \{0,1,2\}$, the flow $f^{\mathcal{D}}(j, t_p)$ is the sum of all incoming flows. Thus node balance constraints hold for all nodes $j, \hat{d}_j \in D \cup O \cup B$ and due to the feasibility of $\mathcal{D}$, the respective sums do not exceed $u(j, t_p) = n_j$. The flow on arcs $(k, k'), o_k \in O$, is the sum of cars which arrive early at operative storage o_k, such that the node balance constraints also hold for nodes $k', o_k \in O$. As $\mathcal{D}$ is feasible, the sum of early arrivals does not exceed the rest capacity of o_k and $f^{\mathcal{D}}(k,k') \leqslant u(k,k')$. Since all node balance and capacity constraints hold for $f^{\mathcal{D}}$, it is a feasible flow in N_I. □

Lemmas 39 and 41 show the equality of integral feasible flows in N_I and feasible distributions with respect to I (disregarding strong priority order). The appropriate definition of costs provides:

Theorem 42 (Minimum Cost Flow and Optimal Distribution) *Let f^* be an integral minimum cost flow in N_I, then the derived distribution $\mathcal{D}^*$ is an optimal distribution for I, disregarding strong priorities.*

Proof: By Lemma 39, $\mathcal{D}^*$ is a feasible distribution for I, disregarding strong priorities. Let $c(f^*)$ denote the cost of f^*, then:

$$c(f^*) = \sum_{(i,j)\in A_I} f_{ij}c(i,j) = \sum_{(i,j)\in A_T} f_{ij}c(i,j) = \sum_{\delta_{ij}\in\mathcal{D}} n_{ij}c_{ij} = c(\mathcal{D}^*).$$

Assume $\mathcal{D}'$ is a feasible distribution for I disregarding strong priorities with cost $c(\mathcal{D}) < c(\mathcal{D}^*)$. Then by Lemma 41, there exists a feasible integral flow $f^{\mathcal{D}}$ with cost $c(f^{\mathcal{D}}) = c(\mathcal{D}) < c(f^*)$ contradicting the minimality of f^*. Thus $\mathcal{D}^*$ is an optimal distribution for I, disregarding strong priorities. □

So far, our derived distributions disregard the strong priority order. How to guarantee the latter algorithmically is discussed at the end of the next section.

5.3 SPECIAL MINIMUM COST FLOWS

In Section 5.2 we showed that a minimum cost flow solution to the derived network model N_I delivers a minimum cost distribution for I, disregarding strong priorities. In this section we motivate and explain the application of the *Successive Shortest Path* algorithm to obtain an integral minimum cost flow f^*, such that $\mathcal{D}^*$ is an optimal distribution for I and respects strong priorities. With the use of the *Successive Shortest Path* algorithm we also develop a reoptimization approach (see Chapter 6) and do not need to precompute a maximum flow. We start with a short survey of network flow algorithms concentrating on minimum cost flow algorithms.

5.3.1 *Minimum Cost Flow Algorithms*

In context of network models, there is a variety of closely related flow problems to be answered, such as the question for a maximum flow/circulation, minimum cost flow/circulation, (un)capacitated transhipment problems and also generalized variants with gains and losses (or multipliers), which in some cases are also equivalent to the traditional flow models (see Section 7.1, [51, 115]).

As the above mentioned network problems are special cases of linear programming, historically the simplex method was the first solution method which gave an exponential upper bound (avoiding cycling) on the problem solution in terms of running time. Later on, ellipsoid [82] and interior point [80] methods showed that the problems can also be solved in pseudo-polynomial time. The underlying graph theoretic structure encouraged the development of LP solvers specialized to network problems like the *network simplex* [26] and modifications to obtain strongly polynomial running time for maximum flow [56] and minimum cost flow [99, 57].

The design of purely combinatorial algorithms started in the nineteen-sixties with the *augmenting paths* [39] and *out of kilter* algorithm [92] for maximum flow and minimum cost flow respectively and is continuously improved in terms of running time and problem variants until today. The above mentioned pioneer algorithms for the maximum flow and minimum cost flow problems originally provided pseudo-polynomial running time, i.e. depended linearly on the maximum balance, capacity and cost values. The *augmenting paths* algorithm was improved to a strongly polynomial version independently by Dinic and Edmonds/Karp [30, 33]. The latter also provides the first polynomial, although not strongly polynomial, minimum cost flow algorithm and introduces 'capacity scaling'. Since then the idea of scaling had many applications to flow algorithms for example like cost scaling [106, 14, 54] and double scaling combining both [4]. The first strongly polynomial algorithm for the minimum cost circulation variant was presented by Tardos [111], settling a challenge posed by [33].

Most combinatorial minimum cost flow algorithms such as the *cycle-canceling* [83], the *augmenting paths* and *Successive Shortest Path* algorithm [73, 15, 75] and the *out of kilter algorithm* maintain either feasibility or (cost) minimality and strive to obtain the missing. The partial 'solution' or pseudo-flow is then improved iteratively until the appropriate optimality criterion ([2]) is fulfilled and the flow is feasible. Optimality conditions, such as the absence of negative cost cycles in case of the *cycle cancelling* approach or the existence of a node potential which generates only non-negative reduced costs on the arcs, refer to a residual network built during the iterations. Further, such conditions base strongly on the characteristic of the network flow problem as a special case of linear programming and duality. (See [2], Chapter 9, for detailed derivation of different optimality criteria.)

For example the network simplex can be seen as a cycle canceling algorithm. Generically, cycle cancelling does not provide an order in which negative cost cycles should be cancelled. Thus, as in the case of the network simplex, there are actually examples for instances without pseudo-polynomial bounds on the running time. On the other hand, the choice of canceling a negative cycle with maximum improvement (for the objective function) or minimum mean cost guarantees pseudo-polynomial running time. The minimum mean cost cycle is preferable as identifying the maximum improvement cycle states an NP-complete problem.

General approaches to minimum cost flow problems are strongly connected to the shortest paths problem and the maximal flow problem. Both problems (depending on the choice of the minimum cost flow algorithm) may be applied as an iteration step on the residual network to obtain a minimum cost flow solution.

5.3.2 *Successive Shortest Path Algorithm*

We briefly recapitulate the *Successive Shortest Path* algorithm (see Algorithm 1): The algorithm starts from zero flow, which satisfies capacity and flow conservation constraints, except at sources ($b(e) > 0$) and sinks ($b(e) < 0$). It is therefore not a feasible (primary) solution. The pseudo-flow is iteratively transferred into a flow by augmenting the maximum possible number of flow units along a shortest path π_{ed} from a source e to a sink d in the residual network. We start from the original network, which is updated according to flow augmentations as follows: For each arc $a = (u, v)$ with a positive flow $f(a) > 0$, an arc $\bar{a} = (v, u)$ with capacity $u(\bar{a}) = f(a)$, cost $c(\bar{a}) = -c(a)$ and flow $f(\bar{a}) = 0$ is added. If $f(a)$ equals $u(a)$, then a is removed from the network. While the shortest path π_{ed} is determined, each vertex v is labeled with its distance $d(e, v)$ from e and we update its (initially zero) potential to $p'(v) = p(v) - d(e, v) + d(e, d)$ if v was permanently labeled. Otherwise, the potential $p(v)$ remains unchanged. The reduced cost $c_r(a) = c(a) - p(u) + p(v)$ of an arc $a = (u, v)$ measures its cost relatively to the shortest path distance of u and v. As each iteration maintains capacity constraints and the node potentials p satisfy the reduced cost-optimality condition, a minimum cost flow solution is obtained, as soon as all excesses and deficits are balanced.

Furthermore, the reduced cost-optimality ensures non-negative arc weights. Since also the initial arc costs are non-negative, we can apply *Dijkstra's algorithm* [29] for the shortest path computations. With the Fibonacci heap data structure, a running time of $O(m + n \log n)$ is achieved [2]. With Dijkstra's Algorithm as a subroutine, the overall running time of the *Successive Shortest Path* algorithm is therefore $O(U(m + n \log n))$, where U is the minimum of the sum of all supplies and the sum of all demands. This value U bounds the number of iterations, because each iteration reduces a supply and a demand at least by one, which also accounts for the integrality of the resulting minimum cost flow. So far, the algorithm is pseudo polynomial as it depends (linearly, not logarithmically) on U. Applying known scaling techniques [54, 2] (on balances, capacities, costs or combined) would lead to polynomial running time. For our application, the pseudo-polynomial implementation has proved to be adequate, as the number of augmentations showed to be roughly determined by the number of aggregated supplies (see Chapter A).

In general, we do not limit the number of sources and sinks in a network, but require the sum of excesses (positive balance values at sources) and deficits (negative balances at sinks) to equal each other. Moreover, sources and sinks are assumed to be connected by paths with sufficient total capacity such that excesses and deficits can be balanced. Although in our application these requirements might be violated, for theoretical analysis and description we will always assume that a maximal flow has been precomputed as mentioned above and total supply and demand at source s and sinks $t_p, p \in \{0, 1, 2\}$, has been adapted. Using an appropriate version of the *Successive Shortest Path* algorithm, a preliminary maximal flow computation is not necessary: We reformulate the algorithm such that in case not all given supply and demand can be balanced, it fails to detect a shortest (or any) path from a source to a sink and terminates. The maintained optimality of the pseudo-flow nevertheless provides us with the minimum cost solution for a maximum flow. In our case the latter corresponds to the distribution of all supplied cars.

Algorithm 1 *Successive Shortest Path*

```
1: $\forall v \in V : p(v) = 0$
2: $\forall a = (u, v) \in A : f(a) = 0, c_r(a) = c(a), u_r(a) = u(a)$
3: $E = \{v, b(v) > 0\}, D = \{v, b(v) < 0\}$
4: while $E, D \neq \emptyset$ do
5:     Determine shortest path $\pi^*_{ed}$ and $d(e, d)$ for $e \in E, d \in D$ with respect
       to reduced costs $c_r$ in the residual network $G(f)$
6:     Update $p(v) = p(v) - d(e, v) + d(e, d)$ for each permanently labeled
       $v \in V$
7:     $\delta := \min\{b(e), -b(d), \min_{a=(uv)\in\pi^*_{ed}} u_r(a)\}$
8:     Augment $\delta$ units of flow along $\pi^*_{ed}$
9:     $\forall a = (u, v) \in A :$
10:    $f(a), c_r(a) = c(a) - p(u) + p(v), u_r(a) = u(a) - f(a)$
11:    Update $G(f), E, D$
12: end while
```

Thus a partial solution is always feasible and (with respect to the amount of already distributed cars) also an optimal solution. Due to this robustness, a preliminary solution can be used in practice if the optimization is interrupted or delayed. We are also able to detect and output isolated supplies and undersatisfied demands in a single run. Further, we benefit from the permanent feasibility of a partial solution in developing a reoptimization strategy (see Chapter 6).

Finally, the use of the *Successive Shortest Path* algorithm ensures the strong prioritization order of demand satisfaction, as we explain in the next section.

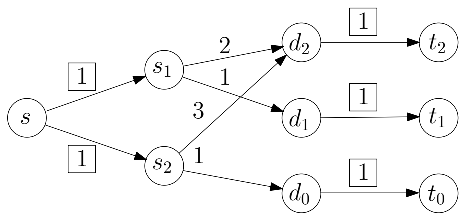


Figure 9: Network N_I for I with two unit supplies s_1, s_2 and three unit demands d_1, d_2, d_3 of three different strong priorities.

5.3.3 *Strong Priority Order*

Algorithm 1 allows arbitrary sets of source and sink nodes, whereas a network N_I associated to a (DP) instance I always contains one super source s and (up to) three sink nodes t_0, t_1, t_2. The sinks are associated with the strong priority levels in ascending order. The generic *Successive Shortest Path* algorithm would always pick the pair $s, t_p, p = 0, 1, 2$, with the shortest reduced cost distance in the current residual network and proceed. Our implementation given in Algorithm 2 always picks the pairs (s, t_2), (s, t_1) and (s, t_0) in descending order.

Algorithm 2 *Successive Shortest Path with predefined Order*

```
1:  $\forall v \in V : p(v) = 0$
2:  $\forall a = (u, v) \in A : f(a) = 0, c_r(a) = c(a), u_r(a) = u(a)$
3:  $p = 2, t = t_p$
4:  **while** $p \geqslant 0$ **do**
5:      Determine shortest path $\pi^*_{st}$ and $d(s, t)$
6:      Update $p(v) = p(v) - d(s, v) + d(s, t)$ for each permanently labeled
        $v \in V$
7:      $\delta := \min\{b(s), -b(t), \min_{a=(uv)\in\pi^*_{st}} u_r(a)\}$
8:      Augment $\delta$ units of flow along $\pi^*_{st}$
9:      $\forall a = (u, v) \in A :$
10:     $f(a), c_r(a) = c(a) - p(u) + p(v), u_r(a) = u(a) - f(a)$
11:     Update $G(f)$
12:     **if** $\pi_{st} \notin G(f)$ **then**
13:         $p - -, t = t_p$
14:     **end if**
15: **end while**
```

Selecting the source and sink pairs by definition and independently of the shortest path distances may lead to suboptimal solutions in terms of costs for pseudo-flows during the iterations. Consider the example in Figure 9,

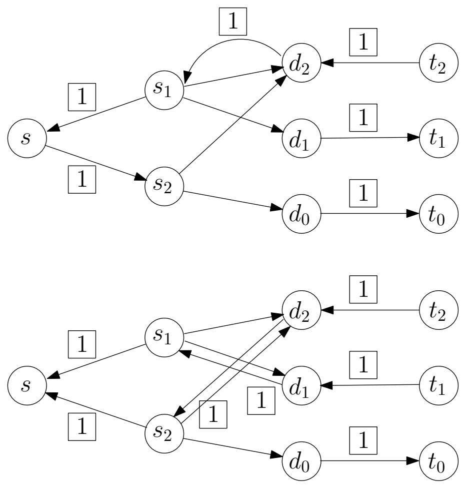


Figure 10: Residual network for N_I after two iterations of Algorithm 2: The derived distribution obeys strong priority order.

where numbers in boxes are capacities and numbers attached to the arcs are costs. Recall that model arcs have cost zero and transit arcs have unlimited capacity. The balance at source s (sinks t_0, t_1, t_2) is the sum of capacities of outgoing (incoming) arcs. The optimal solution obviously sends one unit to t_0 via d_0 and one unit to t_1 via d_1 with total cost 2.

This corresponds to a distribution that leaves a demand with maximal strong priority unsatisfied although the network allows for another (more expensive) flow solution. After two iterations with predefined source-sink-order, the *Successive Shortest Path* algorithm finds the solution as shown in Figure 10. The latter corresponds to a more expensive distribution, which on the other hand fully respects strong priority order.

Note that we cannot guarantee higher priority demands to be satisfied although lower priority demands also receive cars: Without arc (s_2, d_2), the optimal solution would satisfy demand d_0 and d_2 instead of d_1 and d_2. Nevertheless, the latter is a feasible distribution with respect to the strong priority order, as d_1 and d_2 cannot both be satisfied any longer.

On the one hand, if supply equals (or exceeds) demand and we consider the three source-sink-pairs in a single computation, we still obtain a cost-optimal solution: Independent of the order of flow augmentations, finally all demands are satisfied and the residual arcs allow for any feasible rerouting of the flow in the transit layer, such that the overall minimum cost flow is achieved. (Note that this cannot be achieved by solving three consecutive minimum cost flow problems – one single-source-single-sink computation for each source-sink-pair, where the network depends on the outcome of each previous computation. Here the effect of predefined source-sink-pairs cannot be undone by rerouting of flow, as we set up different (residual) networks.)

On the other hand, in practice (pseudo) demand usually exceeds supplies. But flow which is routed to the higher priority sinks in earlier iterations can still be rerouted to decrease the overall costs in Algorithm 2. Yet, the (residual) network structure only allows such reroutings that keep the total amount of flow into the higher priority sink unchanged. Thus we obtain a feasible flow, which distributes the maximal amount of supply and provides minimum costs while respecting strong priority order:

Lemma 43 ($\mathcal{D}^f$ with Strong Priority Order) *Let* f *in* N_I *be determined by Algorithm* 2 *and* $\mathcal{D}^f$ *be the derived distribution from* f. *Then* $\mathcal{D}^f$ *is a feasible distribution for* I.

Proof: The distribution $\mathcal{D}^f$ is feasible disregarding strong priority by Lemma 39. We show that $\mathcal{D}^f$ respects strong priority order: Assume that $d_j \in D$ is undersatisfied with $\nu(d_j)$ by $\mathcal{D}^f$ and preferred over $d_j' \in D$, where $d_{ij'} = (s_i, d_j', n_{ij'}) \in \mathcal{D}^f$ and s_i matches d_j as well as d_j'.

Then f contains flow $f(s,i) \geqslant n_{ij'}$, $f(i,j')$, $f(j' t_p') \geqslant n_{ij'}$ on the path s, i, j', t_p'. Further, arc $(j, t_p), p' < p$, has rest capacity $u_r(j, t_p)$. Hence, before sending flow along the path s, i, j', t_p' there still was a path s, i, j, t_p, between s and t_p such that Algorithm 2 would have sent flow along this path at first. As each later redirection of the flow from t_p to t_p' in the residual network also requires a path between s and t_p to exist, but the considered source sink pair switched from (s, t_p) to $(s, t_p'), p' < p$, this is a contradiction to f being a flow computed by Algorithm 2. □

By the same arguments as in Lemma 41, we can construct a feasible flow $f^{\mathcal{D}}$ in N_I from each feasible distribution $\mathcal{D}$ for I with $c(f^{\mathcal{D}}) = c(\mathcal{D})$. This is especially true for those distributions respecting the strong priority order. Thus, $\mathcal{D}^f$ is also of minimum cost:

Corollary 44 (Optimal Distribution $\mathcal{D}^f$) *Let* f *be a flow in* N_I *computed by Algorithm* 2 *and* $\mathcal{D}^f$ *be the derived distribution from* f. *Then* $\mathcal{D}^f$ *is an optimal distribution for* I.

6 NETWORK-BASED REOPTIMIZATION

One of the major challenges in freight car distribution is the problem's online nature. That is, minor changes on the given data basis can occur over time. The latter being due to new supply cars emerging unforeseen (for example at border stations), damage of supply cars, cancellations or short-termed changes of demands or simply due to the rolling time horizon. On the other hand, also system inherent procedures like the final distribution of a supply to a demand results in changes of the network as supply and demand (partly) drop out of the optimization process.

In a static context, the data of a given problem instance is fixed before the optimization process starts. The optimal solution is then carried out completely and a new problem instance is created on the basis of new data without memory of the previous optimization. In our application context, we want to abandon the static view and thereby increase the optimization potential. All the more, as usually a distribution needs not to be processed immediately after an optimization run. Depending on the time for which a supply is known in advance, the assignment to a demand can be held back during its planning interval (see 2.6), such that upcoming other supplies and demands possibly leading to a globally more cost-efficient solution are taken into account. Generally, a supply is known just before its transport from the actual returning customer's location is started. With regard to the highly clustered association with distribution stations the corresponding distance between customer locations and dispo stations leave a considerable time interval for planning. The distribution for the supply can change within this planning interval, up to a certain time window before the supply actually reaches the dispo station, because only there the productive process changes due to the distribution destination.

Assignments, which are found to be optimal within the planning interval are called 'tentative' assignments. They also contribute to a rather stable data volume, as their associated supply and demand possibly stay in the optimization process (or the network respectively) for a number of runs until the end of the planning interval is reached. (This obviously depends on the frequency of reoptimization runs.) Thus the data on the one hand changes due to the above described (node based) incidences, which involve supplies and demands separately. On the other hand, changes occur to supply and demand simultaneously due to former tentative assignments which become 'fixed', as the considered supply leaves its planning interval.

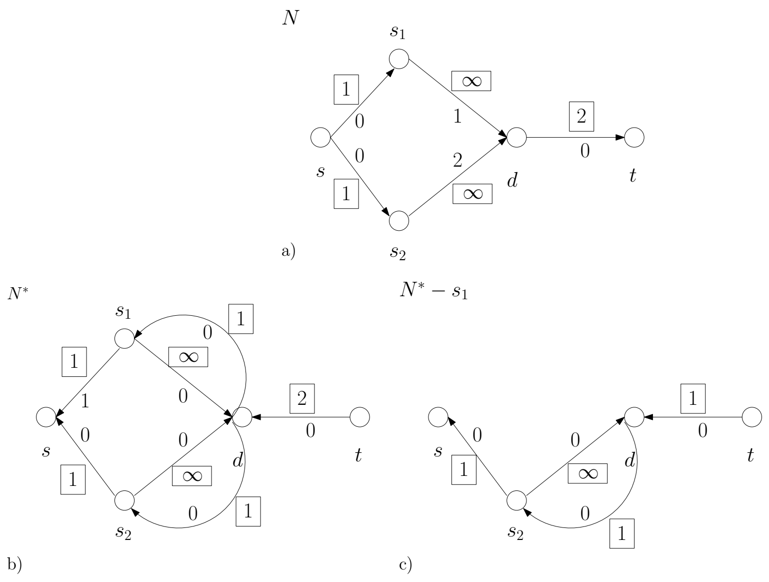


Figure 11: Network (a) and residual network with optimal flow, before (b) and after (c) 'tentative' assignment of s_1 to d is 'fixed' and removed from the network.

Such changes - if according to the current flow solution - do not affect the feasibility and/or optimality of the remaining solution: supply, demand, flow and capacity (of the residual arcs) can be adjusted. Consider the simple example in Figure 11. Both supplies are tentatively allocated to the only demand in the (only) optimal solution. As soon as supply s_1 leaves the planning interval, the distribution of s_1 to d is fixed and the corresponding flow, including residual capacities and arcs, are deleted from the residual network as well as the corresponding supply node. (Note that the reduction of residual capacity on the reverse arc (t, d_j) corresponds to decreasing the number of demanded cars in d, because the order is already partly satisfied.) The remaining network directly corresponds to the residual network of an optimal flow given only s_2 and the reduced demand d.

Still also manual assignments can be given priority over the optimized solution or are carried out during the computation. The latter need not be consistent with 'tentative' assignments. These cases are interpreted as changes of the supplies and demands, which we address by our reoptimization procedure.

As we expect the changes to be minor relative to the whole current data volume, our strategy is to keep the current residual network intact and 'repair' it to associate with an optimal, feasible solution to the changed data basis again. We also assume that data changes only affect input data and not master data. The latter is agreeable in case of substitution rules, as requirements of good transport do not suddenly change and thus do not allow for different substitutions. With regard to the timetable, the optimization process is also detached from an up-to-date timetable, which can be affected by sudden changes such as track closures due to accidents or weather conditions, because we rely on itineraries (or 'Leitwege') rather than actual trains. Further, with a change in the timetable or rule sets, the whole residual network has to be checked for feasibility (especially the transit arcs) and possibly be repaired. Thus running time advantages over a computation 'from scratch' cannot be expected in case of a network-wide rearrangement. With this assumption, we observe that changes in the input data can always be modelled by the following three operations on the residual network:

1. Node addition
2. Node deletion
3. Change of node attributes

The third operation can generally be simulated by deletion of the original node and the addition of a node with changed attributes. For frequently occurring changes, especially a changed number of cars, our implementation supports faster direct changes. Here, we concentrate on the first two cases, which both require a two-stage change of the residual network. One stage keeps the optimality of the current flow solution while adding or deleting the node and the other stage returns the now possible pseudo-flow again into a flow by additional iterations of the *Successive Shortest Path* algorithm (in the variant of Algorithm 2). The detailed procedure and order of the stages depend on whether it is a supply or demand node which is added or deleted.

In the following subsections, we will establish both stages for addition and deletion of single supply and demand nodes and afterwards combine the iterative stages for distinct groups of supply and demand nodes with respect to the operation carried out. The following modular reformulation of the *Successive Shortest Path* algorithm enables a compact representation of the reoptimization procedures. (Note that Algorithm 3 also predefines the order of source-sink-pairs according to Algorithm 2.)

Algorithm 3 *Dispositive Successive Shortest Path*

```
Input: $N = (V, A), s, t_2, t_1, t_0$
Output: $f^*, N(f^*) = (V, A^*)$
1: $\forall v \in V : p(v) = 0$
2: $\forall a = (u, v) \in A : f(a) = 0, c_r(a) = c(a), u_r(a) = u(a)$
3: Iterate$(N, s, t_2)$
4: Iterate$(N, s, t_1)$
5: Iterate$(N, s, t_0)$
```

Algorithm 4 *Iterate Successive Shortest Path*

```
Input: $N(f) = (V^f, A^f), s, t$
Output: $f^t, N^t(f^t) = (A^t, V)$
1: while $\exists \pi^*_{st}$ do
2:     Determine shortest path $\pi^*_{st}$ and $d(s,t)$ w.r.t. $c_r$ in $N(f)$
3:     Update $p(v) = p(v) - d(s,v) + d(s,t)$ for each permanently labeled
       $v \in V$
4:     $\delta := \min\{b(s), -b(t), \min_{a \in \pi^*_{st}} u_r(a)\}$
5:     Augment $\delta$ units of flow along $\pi^*_{st}$, obtaining $f'(a)$
6:     $\forall a = (u,v) \in A^f : c_r(a) = c(a) - p(u) + p(v)$
7:     $\forall a = (u,v) \in A^f : u_r(a) = u(a) - f'(a)$
8:     Update $N(f)$ to $N(f')$
9: end while
```

6.1 ADDITION OF NODES

When we build a network N_I for a given (DP) instance I from scratch, each supply (demand) node is connected to source s (one of sinks $t_p, p \in \{0,1,2\}$) via an incoming (outgoing) model arc and connected to all matching demand (supply) nodes via outgoing (incoming) transit arcs. After the network setup, the algorithm starts with zero flow and a zero potential function, which (due to non-negative arc costs) satisfies the reduced cost-optimality criterion as restated in Theorem 45.

Theorem 45 *[2] A feasible minimum cost flow solution f^* is optimal if and only if some set of node potentials p satisfies the reduced cost-optimality conditions:*

$$\forall a = (i,j) \in A : c^p(a) = c(a) - p(i) + p(j) \geqslant 0.$$

Each minimum cost flow solution f^* delivered by Algorithm 3 also delivers a corresponding potential function by p.

Given a residual network $N^* = N(f^*) = (V, A^*)$ to the minimum cost flow f^*, we can add a new supply or demand node v to V as in the network construction. Further, we want to add the same model and transit arcs A_v as in the original construction and obtain $N' = (V' = V \cup \{v\}, A' = A \cup A_v)$. A minimum cost flow on N' again delivers an optimal distribution solution. To maintain reduced cost optimality, for each arc $a = (u,v) \in A_v$ ($a = (v,u) \in A_v$): $c^p(a) = c(a) - p(u) + p(v) \geqslant 0$ ($c^p(a) = c(a) - p(v) + p(u) \geqslant 0$) must hold.

Although a final optimal and maximal feasible pseudo-flow solution in N' possibly requires the rerouting of flow throughout the network, we want to keep the initial change in the network structure local. Thus, to avoid network-wide distance recalculations and potential adjustments in the first step, we fix the potential of all 'old' nodes $u \in V$ to $p(u)$ and set $p(v)$ according to the respective arc conditions. Further, we generally start with flow $f'(a) = f^*(a)$ for $a \in A$ and zero flow $f'(a_v) = 0$ on the additional 'new' arcs $a_v \in A_v$.

As the new nodes by the desired network construction generally have incoming as well as outgoing arcs, the potential cannot always be chosen according to reduced cost-optimality. Yet, both in the case of new supply or demand, it suffices to reverse the corresponding new model arc in the residual network by sending the maximum flow along it and temporarily creating imbalances at super source, super sink or data nodes.

6.1.1 *Additional Supply*

Adding supply node i to N^*, we need to create an arc (s, i) with non-negative (reduced) cost and capacity n_i, the number of supplied cars. On the other hand, we create arcs $a = (i, j)$ for every matching (pseudo) demand and denote the set of additional transit arcs by A_i. This leads to the following reduced cost constraints:

1. $a = (s, i) : c^p(a) = c(a) - p(s) + p(i) = p(i) - p(s) \geqslant 0$
2. $\forall a = (i, j) \in A_i : c^p(a) = c(a) - p(i) + p(j) \geqslant 0$

The constraints of the second type will always be satisfied by setting

$$p(i) = \min_{a=(i,j)\in A_i}\{c(a) + p(j)\}.$$

This choice on the other hand needs not satisfy the first constraint. We distinguish exactly these two cases: If the first constraint is satisfied, we set $A_i = A_i \cup \{(s, i)\}$ and $A = A \cup A_i$ with reduced cost set appropriately to $p(i)$ and capacity n_i. The new flow f' composes of f^* and zero flow on the new arcs, as defined above. As the total supply is increased by n_i, which again creates excess at s, we iterate the *Successive Shortest Path* algorithm with input N' and the pairs s, t_2, s, t_1 and s, t_0, to achieve a maximal feasible pseudo-flow.

In case the first constraint is violated by the choice of $p(i)$, we add the arc $\bar{a} = (i, s)$ to A_i instead of $a = (s, i)$. The addition of the reverse arc $\bar{a}$ can be interpreted as sending n_i flow units from s to i. This creates an excess of $b(i) = n_i$, a (temporal) demand $b(s) = n_i$ and saturates a, i.e. $u_r(a) = 0$ and a drops out of the residual network N'. The resulting flow f' differs from the above definition only on a as $f'(a) = n_i$. The reduced cost constraint for (i, s) must now be satisfied and we also apply iterations of the *Successive Shortest Path* algorithm with input N' and the pairs s_i, t_2, s_i, t_1, s_i, t_0 and s_i, s to make the pseudo-flow f' maximal feasible.

Algorithm 5 *Add Supply*

```
Input: $N^* = (V, A), f^*, s, s_i, t_2, t_1, t_0$
Output: $N' = (V', A'), f'$
 1: $\forall a \in A : f'(a) = f^*(a)$
 2: $A_i = \{(i, j) | s_i$ matches $\hat{d}_j\}$
 3: $\forall a \in A_i : f'(a) = 0$
 4: $\forall v \in V : p(v) = p(v)$
 5: $V' = V \cup \{s\}, A' = A \cup A_i$
 6: $p(i) = \min_{a=(i,j) \in A_i} \{c(a) + p(j)\}$
 7: if $p(s) \leqslant p(i)$ then
 8:     $a = (s, i), c_r(a) = p(i) - p(s), u_r(a) = n_i, f'(a) = 0$
 9:     $A' = A' \cup \{a\}, b(s) = b(s) + n_i, N' = (V', A')$
10:     Iterate($N', s, t_2$)
11:     Iterate($N', s, t_1$)
12:     Iterate($N', s, t_0$)
13: else
14:     $\bar{a} = (i, s), c_r(\bar{a}) = p(s) - p(i), u_r(\bar{a}) = n_i, f'(\bar{a}) = 0$
15:     $A' = A' \cup \{\bar{a}\}, b(i) = n_i, b'(s) = -n_i, N' = (V', A')$
16:     Iterate($N', i, t_2$)
17:     Iterate($N', i, t_1$)
18:     Iterate($N', i, t_0$)
19:     Iterate($N', i, s$)
20: end if
```

Note that if f^* was a feasible flow, i.e. all excesses and deficits were balanced, Algorithm 5 only finds a path from i to s in the *'else'*-case. On the other hand, for our application we can assume the existence of rest demand in the form of operative storage.

Lemma 46 (Additional Supply) *Algorithm 5 finds a minimum cost flow on* $N' = (V \cup \{i\}, A \cup A_i)$, *given* $N^* = (V^*, A^*)$ *for a minimum cost flow* f^* *in* $N = (V, A)$ *in* $O(n_i)$ *Successive Shortest Paths iterations after preparation* $O(m)$, *such that the strong priority order of demand satisfaction is kept.*

Proof: Algorithm 5 establishes a pseudo-flow f' obeying the reduced cost criterion and the subsequent iterations achieve a feasible flow assuming balanced total supply and demand and appropriate paths in the network (or a maximal feasible pseudo-flow in practice). Further, due to the predefined iteration order of source and sink pairs Corollary 44 holds. □

In context of linear programming duality, the potential function on the nodes is a set of values for dual variables of the minimum cost flow formulation. They can thus be interpreted to be the price of a flow unit at the corresponding node. (Note that we defined the node potential to be the accumulated negative distances.) Thus the cases $p(i) \geqslant p(s)$ and $p(i) < p(s)$, which lead to the existence of a or $\bar{a}$ respectively can also be nicely interpreted in terms of how desirable a distribution of the new car supply is.

On the one hand, if $p(i) \geqslant p(s)$, the flow from s through s_i to a demand is not more desirable than all previously sent flows, as its price (measured by $p(s_i)$) is higher than all paths used before (measured by $p(s)$). If the demand is already completely satisfied or there is no way from s_i to a still unsatisfied demand, the flow f^* will not change as we already computed the optimal solution in absence of s_i. Even if it is possible to satisfy more demand with the additional supply s_i, exactly f^* would have occurred as a pseudo-flow after some iterations of the *Successive Shortest Path* algorithm, knowing about s_i in advance. The algorithm did 'nothing wrong' during the computation of f^* in N^* regarding N' due to the lack of information and the subsequent iterations merely complete the computation of the *Successive Shortest Path* algorithm given N'.

On the other hand, if $p(i) < p(s)$, the price to send a flow unit or a car via s_i is lower than the price for some paths already used. The algorithm - with respect to the unknown N' while computing f^* in N^* - formerly chose the 'wrong path' due to the lack of information. To correct this wrong path, we first try to satisfy additional demand n_i and thereby reroute the flow which was 'too expensive'. If there is not enough demand or path capacity left to do so, we return the remaining excess of s_i to s with the last iteration. Hereby we either choose a path via another supply node s_k and a backwards arc (k, s) or flow is sent back to s directly via the reverse arc $\bar{a}$.

In the former case we exchange s_k in a distribution by s_i, which means we reroute flow along some path containing s_i and s_k. Due to the network structure, the latter must use a path including reversed arcs. With the definition of the residual network, the (non reduced) cost of reversed arcs are the negative cost of the arcs and thus the cheapest return path to s equals to returning the most expensive former supply. According to the potential $p(i) < p(s)$ and by picking the shortest path first, the latter case can only occur after sending at least one unit of flow from i to s via another supply node k. On the other hand, this path does not need to have capacity of n_i and a rest excess must probably be returned to s via $\bar{a}$. Thus, the structure of the potential function on the one hand assures that to achieve optimality such a rerouting is necessary, but on the other hand does not provide how much flow has to be rerouted.

Based on the network in Figure 12, we give a small example for the '*if*'-case (see Figure 13) as well as the '*else*'-case (see Figure 14).

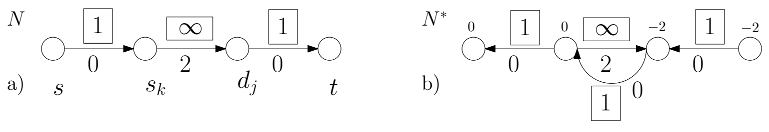


Figure 12: Network (a) and residual network (b) with optimal flow.

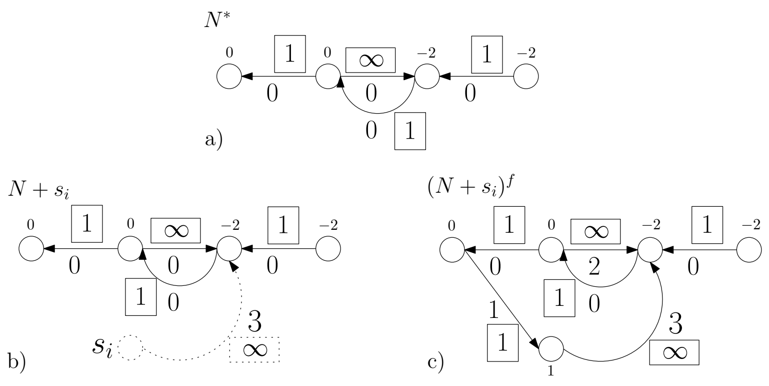


Figure 13: Add supply node s_i with $p(s) \geqslant p(i)$. a) Residual network with optimal flow. b) Additional node s_i and arc set A_i (in gray). c) Residual network with optimal flow after addition of s_i.

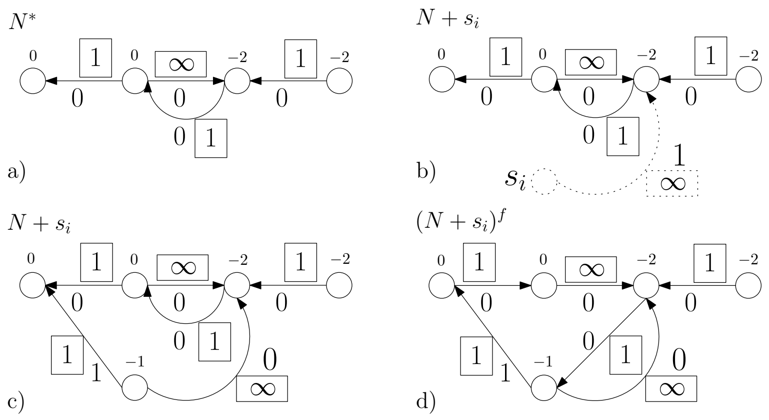


Figure 14: Add supply node s_i with $p(s) < p(i)$. a) Residual network with optimal flow. b) Additional node s_i and arc set A_i (in gray). c) Additional backwards arc (s_i, s). d) Residual network with optimal flow after addition of s_i.

6.1.2 *Additional Demand*

We start by describing the addition of a demand node d_j to N^* in detail. As pseudo demand nodes are handled similarly, we define additional steps or simplifications respectively afterwards. Further, we assume $t = t_p, p \in \{0, 1, 2\}$ to be the sink with appropriate priority $p = p_j$. We create arcs $A_j = \{a = (i, j)\}$ for every matching supply i and need an additional model arc (d_j, t) with zero (reduced) cost and capacity n_j, the number of ordered cars.

This leads to the following reduced cost constraints:

1. $a = (j, t) : c^p(a) = c(a) - p(j) + p(t) = p(t) - p(j) \geqslant 0$
2. $\forall a = (i, j) \in A_j : c^p(a) = c(a) - p(i) + p(j) \geqslant 0$

The constraints of the second type will always be satisfied by setting

$$p(j) = max\{-min_{a=(i,j)\in A_j}\{c(a) - p(i)\}, 0\}.$$

This choice again needs not satisfy the first constraint and we consider the two cases: If the first constraint is satisfied, we add $a = (j, t)$ with $c(a) = 0, u(a) = n_j$ to A_j and set $A' = A^* \cup A_j$. All reduced cost are set appropriate to the above defined potential $p(j)$. The new flow f' composes of f^* and zero flow on the new arcs. As the total demand is increased by n_i, which again creates deficit at t, we iterate the *Successive Shortest Path* algorithm with input $N(f')$ and the pairs s, t, and $t_k, k < p, k \in \{0, 1, 2\}$.

Flow can only be routed from s to t, if there was a rest excess at s due to supply exceeding demands or lack of sufficiently capacitated paths to the sinks. Further, rerouting flow from a sink t_k to t is enabled by temporarily creating an artificial excess at t_k. In case that there is no rest supply of unassigned cars at s, this is necessary to ensure the strong prioritization ordering of satisfied demands. The latter is not mirrored in the (reduced) cost structure and therefore not taken care of automatically in the following rerouting process. If flow is successfully routed from t_k to t it is done in the 'cheapest' way and creates a new or increased rest demand at sink t_k in favour of additional satisfaction of demands with stronger priority.

If the first constraint cannot be satisfied by the choice of $p(j)$, we push n_j units of flow along $a = (j, t)$ creating temporal demand $b(j) = n_j$ at j, excess $b(t) = n_j$ at t and the reverse arc $\bar{a} = (t, j)$ with appropriate capacity and reduced costs. Subsequently we iterate the *Successive Shortest Path* algorithm with input $N(f')$ and the pairs s, j, and $t_k, j, 0 \leqslant k \leqslant 2$ as before. We summarize the different steps in Algorithm 6.

Algorithm 6 *Add Demand*

```
Input: $N^* = (V, A), f^*, s, d_j, t_2, t_1, t_0$
Output: $N' = (V', A'), f'$
 1: $\forall a \in A : f'(a) = f^*(a)$
 2: $A_j = \{(i, j) | s_i \text{ matches } d_j\}$
 3: $\forall a_j \in A_j : f'(a_j) = 0$
 4: $\forall v \in V : p(v) = p(v)$
 5: $V' = V \cup \{j\}, A' = A \cup A_j$
 6: $p(j) = max\{-min_{a=(i,j) \in A_j}\{c(a) - p(i)\}, 0\}$
 7: $t = t_l, l = p_j$
 8: if $p(j) \leqslant p(t)$ then
 9:     $a = (j, t), c_r(a) = p(i) - p(s), u_r(a) = n_i, f'(a) = 0$
10:     $A' = A' \cup \{a\}, b(t) = b(t) - n_j, N' = (V', A')$
11:     for $k = l$ to $k = 0$ do
12:         Iterate($N', s, t_k$)
13:     end for
14:     for $k = 0$ to $k = l$ do
15:         if $b(j) < 0$ then
16:             $b(t_k) = b(j)$
17:             Iterate($N', t_k, j$)
18:         end if
19:     end for
20: else
21:     $\bar{a} = (t, j), c_r(\bar{a}) = p(t) - p(j), u_r(\bar{a}) = n_j, f'(\bar{a}) = 0$
22:     $A' = A' \cup \{\bar{a}\}, b(j) = -n_i, N' = (V', A')$
23:     Iterate($N', s, j$)
24:     for $k = 0$ to $k = l$ do
25:         if $b(j) < 0$ then
26:             $b(t_k) = b(j)$
27:             Iterate($N', t_k, j$)
28:         end if
29:     end for
30: end if
```

Lemma 47 (Additional Demand) *Algorithm 6 finds a minimum cost flow on* $N' = (V \cup \{j\}, A \cup A_j)$*, given* $N^* = (V^*, A^*)$ *for a minimum cost flow* f^* *in* $N = (V, A)$ *in* $O(n_j)$ *Successive Shortest Paths iterations after preparation* $O(m)$*, such that the strong priority order of demand satisfaction is kept.*

Proof: Algorithm 6 establishes a pseudo-flow f' obeying the reduced cost criterion and the subsequent iterations achieve a feasible flow assuming balanced total supply and demand and appropriate paths in the network (or

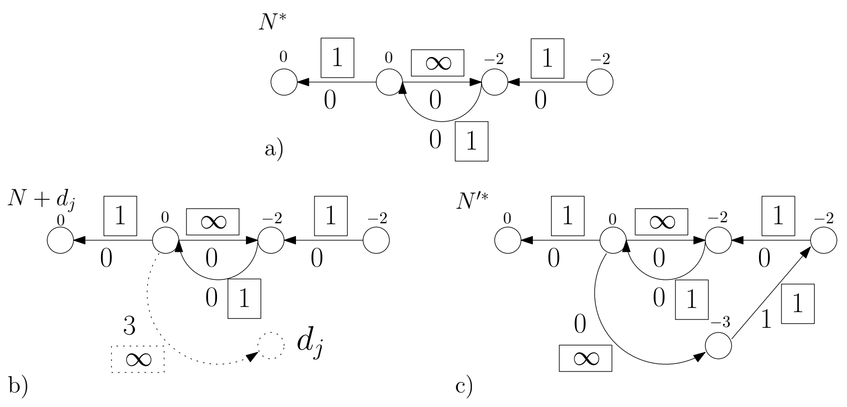


Figure 15: Add demand node d_j with $p(t) \geqslant p(j)$. a) Residual network with optimal flow. b) Additional node d_j and arc set A_j (in gray). c) Residual network with optimal flow after addition of d_j.

a maximal feasible pseudo-flow in practice). Further, due to the predefined iteration order of source and sink pairs, Corollary 44 holds. □

The pricing interpretation based on the potential function and the cases $p(t) \geqslant p(j)$ or $p(t) < p(j)$ also holds for demands. In the first case, the demand remains unsatisfied if there is no rest supply and no flow can be rerouted from sinks with lower priority. Routing of a rest supply would increase the flow value, which legitimates a cost increase, while rerouting flow from the same priority sink would not increase the flow value and will not decrease the total cost, given the reduced cost on a. On the other hand, if $\bar{a}$ is added to the network we can still route additional flow directly from s to d_j or via a sink, which both increases the total flow value. Or we reroute flow from an appropriate (lower or equal priority) sink and thus decrease the total cost while keeping flow value and priority order.

Figures 15 and 16 give examples for both cases with a single sink, based on the optimal network flow of Figure 12. Note that we start from a feasible flow f^* and do not give examples with rest excesses.

Figure 17 gives an example with a lower and a higher priority sink. For the sake of simplicity, we keep the original network as in Figure 12 and let only the additional demand be of higher priority than the given demand. This creates a special case, where also the higher priority super sink is added. Thus, the potential of the sink can be set appropriately, such that only the '*if*'-case occurs.

Although this case might be degenerated, it nicely shows that flow is rerouted in such a case from a lower priority sink to the higher priority sink independent of the actual cost of assigning supply to the new demand d_j.

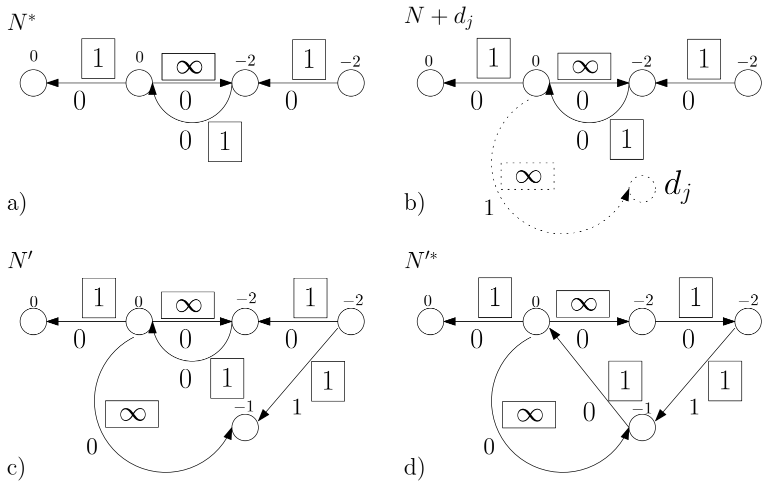


Figure 16: Add demand node d_j with $p(t) < p(j)$. a) Residual network with optimal flow. b) Additional node d_j and arc set A_j (in gray). c) Additional backwards arc (t, d_j). d) Residual network with optimal flow after addition of d_j.

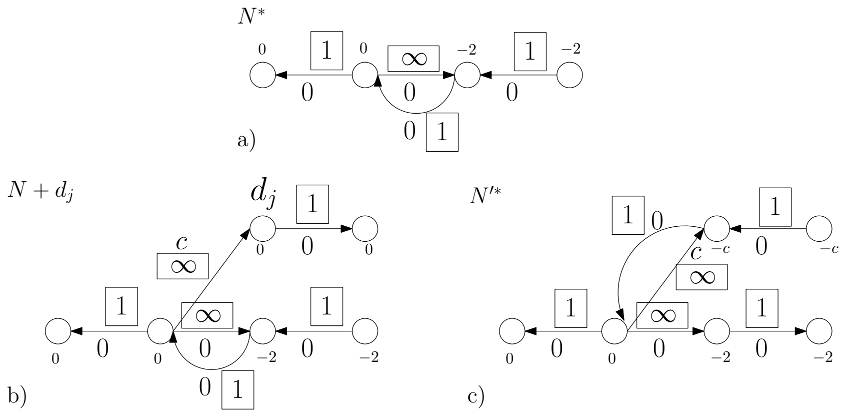


Figure 17: Add demand node with higher strong priority. a) Residual network with optimal flow. b) Additional nodes d_j, t_1, arc (d_j, t_1) and arc set A_j. c) Residual network with optimal flow after addition of d_j.

6.2 DELETION OF NODES

The deletion of a single supply or demand node from the network N^* affects both optimality and feasibility of the given solution f^*. Each supply s_i contributes to the total excess at the super source by n_i, which equals the capacity of arc (s, i). To balance the full excess of s all such arcs are saturated, as they are the only outgoing arcs of s. Thus, f^* sends n_i flow via i and matching demand nodes j to the corresponding sink t. The deletion of i and its incident arcs would leave a deficit at j and f^* is infeasible, whereas a subsequent return of an appropriate amount of flow from t to j may be suboptimal. The latter is true if (t, j) is not (one of) the shortest path(s) between t and j in N^* without A_i. The case of a demand deletion is analogue.

If f^* is a maximal feasible pseudo-flow and there are unassigned supplies (which would indicate a serious problem in the application), i can luckily be corresponding to such a rest supply at s. Or with the assumption that (pseudo) demands exceed supplies, a demand d_j to be deleted is not satisfied. In both cases, the simple deletion of the respective node and all incident arcs a in N^* does not affect feasibility or optimality of f^* as $f^*(a) = 0$ on all such arcs. Our strategy is therefore to transform the more likely case when a node deletion affects the flow f^* into the case where all incident arcs of the node have zero flow. To obtain a flow f' with the desired property, we again manipulate the residual network N' locally and reapply some iterations of the *Successive Shortest Path* algorithm.

Note that often a mixture of the desired case and the case where reoptimization is needed occurs with pseudo-flows involved. This holds for example for a partly satisfied demand in the current optimal distribution which is then cancelled by the customer. In the following, we assume that n_i (n_j) is already reduced by the portion of supply (demand) which can be deleted from N^* without affecting f^*. This is essentially achieved by removing the arc (s, i) or (j, t), which then exists due to non-zero rest capacity.

6.2.1 *Deletion of Supply*

Given a supply node i to be deleted from N^*, we remove the reverse arc (i, s) from A^*. As i originally does not possess positive excess due to network construction, the outgoing flow creates a deficit at i and f^* is infeasible, but still cost-optimal with respect to p and reduced costs c^p. To obtain a feasible flow f', we have to route flow back to i, such that no incident arc carries flow. Then i and A_i can be deleted from $N(f')$ without affecting feasibility or optimality of f'. We reroute the flow out of i back to i in the

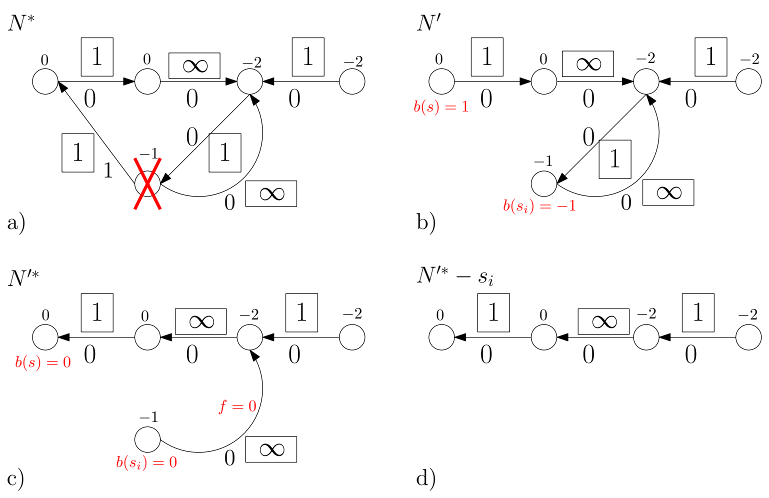


Figure 18: Remove supply node with rest excess at s. a) Residual network with optimal flow and rest excess $b(s) = 1$. b) Removed arc (s_i, s) and temporal artificial sink at s_i with $b(s_i) = -1$. c) Rest excess of s and artificial sink s_i are balanced. d) Node s_i and arc set A_i can be removed without influencing the optimality of the flow.

most cost-efficient way applying iterations of the *Successive Shortest Path* algorithm as in Algorithm 7.

In the first '*Iterate*'-call, the arc (s, i) does not provide a path from s to i — it exists no longer. Thus, due to the network structure each path from s to i mus contain another supply node k, corresponding to a so far unassigned supply s_k of cars which can cover the cancellation of supply s_i (see Figure 18). If on the other hand no rest excess exists, an appropriate amount flow is routed back from the sinks in ascending priority order by turning them into temporal sources (see Figure 19). We always obtain a feasible flow as

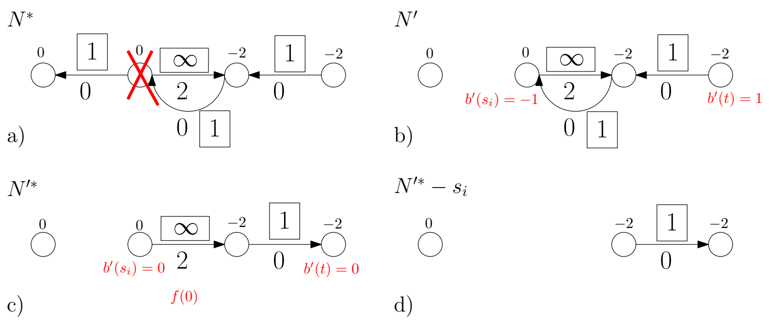


Figure 19: Remove supply node without rest excess at s. a) Residual network with optimal flow. b) Removed arc (s_i, s) and temporal artificial sink at s_i with $b(s_i) = -1$. c) Artificial sink s_i is balanced by augmenting flow back from the sink t. d) Node s_i and arc set A_i can be removed without influencing the optimality of the flow.

the deficit of s_i is exactly the amount of flow which was already routed via i to one of the sinks and can of course be routed back completely.

Lemma 48 (Removed Supply) *Algorithm 7 finds a minimum cost flow on* $N' = (V \setminus \{i\}, A \setminus A_i)$*, given* $N^* = (V^*, A^*)$ *for a minimum cost flow* f^* *in* $N = (V, A)$ *in* $O(n_i)$ *Successive Shortest Paths iterations after preparation* $O(m)$*, such that the strong priority order of demand satisfaction is kept.*

Proof: Algorithm 7 establishes a pseudo-flow f' obeying the reduced cost criterion and the subsequent iterations achieve a feasible flow assuming balanced total supply and demand and appropriate paths in the network (or a maximal feasible pseudo-flow in practice). Further, due to the predefined order of source and sink pairs Corollary 44 holds.

□

Algorithm 7 *Remove Supply*

```
Input: $N^* = (V, A), f^*, s, i, t_2, t_1, t_0$
Output: $N' = (V', A'), f'$
 1: $A' = A \setminus \{(i, s)\}$, $N' = (V', A')$
 2: $b(i) = -n_i$
 3: Iterate($N', s, i$)
 4: $b'(t_0) = -b(i)$
 5: Iterate($N', t_2, i$)
 6: $b'(t_1) = -b(i)$
 7: Iterate($N', t_1, i$)
 8: $b'(t_2) = -b(i)$
 9: Iterate($N', t_0, i$)
10: $A_i = \{a = (i, j) | a \in A'\}, A' = A' \setminus A_i$
11: $V' = V \setminus \{i\}$
```

6.2.2 *Deletion of Demand*

The deletion of a demand node j requires flow to be either rerouted to one or more sinks via other demand nodes (assuming that demands exceed supplies and both are connected with paths of sufficient capacity) or rerouted to source s. The latter generates rest excess (unassigned cars). In both cases, we remove the arc (t, j) from N^* and create n_j excess at j, such that f^* remains cost-optimal, but becomes infeasible. Subsequent iterations of the *Successive Shortest Path* algorithm then reestablish feasibility.

Feasibility is always obtained, because $b(d_j)$ equals the sum of all flow previously routed from s via j to the appropriate sink. Thus the residual

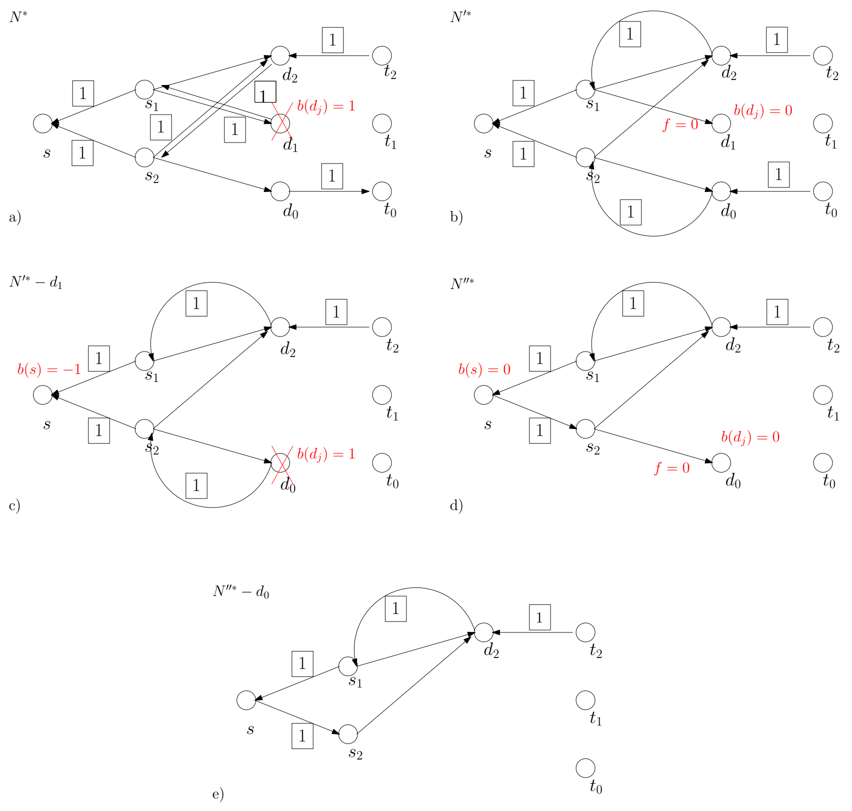


Figure 20: Remove demand node with (d_1) and without (d_0) rest demand. a) Residual network with (former) optimal flow, removed arc (t_1, d_1) and temporal artificial source at d_1 with $b(d_1) = 1$. b) Rest demand at t_0 and artificial source d_1 are balanced. Node d_1 and arc set A_1 can be removed without influencing the optimality of the flow. c) Residual network with (former) optimal flow, removed arc (t_0, d_0) and temporal artificial source at d_0 with $b(d_1) = 1$. d) Artificial source d_0 is balanced by augmenting flow back to the source s creating a rest excess. Node d_0 and arc set A_0 can be removed without influencing the optimality of the flow.

network at least contains backward paths of sufficient capacity from j to s. Further, if there are unsatisfied demands in N^* the prior calls of *'Iterate'* reroute flow via them to the sinks in strong priority order, if appropriate paths exists.

We see an example for both cases in Figure 20, where d_1 and d_0 respectively are deleted from the example in Figure 9 and Figure 10 respectively.

Lemma 49 (Removed Demand) *Algorithm 8 finds a minimum cost flow on* $N' = (V \setminus \{j\}, A \setminus A_j)$, *given* $N^* = (V^*, A^*)$ *for a minimum cost flow* f^* *in* $N = (V, A)$ *in* $O(n_j)$ *Successive Shortest Paths iterations after preparation* $O(m)$, *such that the strong priority order of demand satisfaction is kept.*

Proof: Algorithm 8 establishes a pseudo-flow f' obeying the reduced cost criterion and the subsequent iterations achieve a feasible flow assuming balanced total supply and demand and appropriate paths in the network (or a maximal feasible pseudo-flow in practice). Further, due to the predefined iteration order of source and sink pairs Corollary 44 holds. □

Algorithm 8 *Remove Demand*

```
Input: $N^* = (V, A), f^*, s, j, t_2, t_1, t_0$
Output: $N' = (V', A'), f'$
1: $A' = A \setminus \{(t, j)\}, N' = (V', A')$
2: $b(j) = n_j$
3: Iterate($N', j, t_0$)
4: Iterate($N', j, t_1$)
5: Iterate($N', j, t_2$)
6: $b'(s) = -b(j)$
7: Iterate($N', j, s$)
8: $A_j = \{a = (i, j) | a \in A'\}, A' = A' \setminus A_j$
9: $V' = V \setminus \{j\}$
```

6.3 PSEUDO DEMANDS

Pseudo demands are nodes corresponding to border stations and storage. With respect to the network model and consequently the reoptimization procedures they are treated very similar to demand nodes. For each border station $b_l \in B$, the corresponding node l is always connected to t_0 by the arc (l, t_0) due to the construction of N_I and we can simplify the Algorithms 6 and 8 by removing the superfluous *Iterate*-calls for higher priority sinks. (We do not list the resulting shortened algorithms explicitly.)

Due to time expansion, for each operational storage $o_k \in O$ there are two corresponding nodes k', k and the arc (k', k) in N_I. Thus the Addition of o_k requires to add both nodes, the arc (k', k) and the arc sets

$$A'_k = \{(i, k') | s_i \text{ matches } o_k \text{ with } \theta_{ik} = (l_i, c \geqslant c_i, l_k, c' \leqslant c_k, r_{ik})\}$$

and

$$A_k = \{(i, k) | s_i \text{ matches } o_k \text{ with } \theta_{ik} = (l_i, c \geqslant c_i, l_k, c' > c_k, r_{ik})\}.$$

The potential $p(k')$ is defined similar to the demand potential and $p(k)$ is defined as follows:

$$p(k) = \max\{-\min_{a=(i,k) \in A_k \cup \{(k',k)\}} \{c(a) - p(i)\}, 0\}.$$

This choice of potentials ensures positive reduced cost on $A'_k \cup A_k \cup \{(k', k)\}$, but not necessarily on (k, t_0). If the latter has negative reduced cost we send n_k units of flow from k to t_0 as before, such that the arc drops out of the network. We define f' as $f'(a) = f^*(a), a \in A^*$ and $f'(a) = 0, a \in A'_k \cup A_k \cup \{(k', k)\}$ in the first case and only change $f'(k,t) = n_k$ in the second case. Thus we again start from an infeasible, but cost optimal flow f' and establish feasibility by iterating the *Successive Shortest Path* algorithm as in Algorithm 9. (Like in the case of border nodes, superfluous *Iterate*-calls are removed here.)

Algorithm 9 *Add Operative Storage*

```
Input: $N^* = (V, A), f^*, s, k, t_0$
Output: $N' = (V', A'), f'$
 1: $\forall a \in A : f'(a) = f^*(a)$
 2: $V' = V \cup \{k', k\}$
 3: $A'_k = \{(i, k') | s_i$ matches $o_k$ with $\theta_{ik} = (l_i, l_k, c, c' \leqslant c_k, r_{ik})\}$
 4: $A_k = \{(i, k) | s_i$ matches $o_k$ with $\theta_{ik} = (l_i, l_k, c, c' > c_k, r_{ik})\}$
 5: $\forall a \in A'_k \cup A_k : f'(a) = 0$
 6: $A' = A \cup A'_k \cup A_k$
 7: $p(k') = \max\{-\min_{a=(i,k') \in A'_k}\{c(a) - p(i)\}, 0\}$.
 8: $p(k) = \max\{-\min_{a=(i,k) \in A_k \cup \{(k',k)\}}\{c(a) - p(i)\}, 0\}$.
 9: if $p(k) \leqslant p(t_0)$ then
10:     $a = (k, t), c_r(a) = p(t) - p(k), u_r(a) = n_k, f'(a) = 0$
11:     $A' = A' \cup \{a\}, b(t) = b(t) - n_k$, $N' = (V', A')$
12:     Iterate($N', s, t_0$)
13: else
14:     $\bar{a} = (t_0, k), c_r(\bar{a}) = p(k) - p(t), u_r(\bar{a}) = n_k, f'(\bar{a}) = 0$
15:     $A' = A' \cup \{\bar{a}\}, b(k) = -n_k$, $N' = (V', A')$
16:     Iterate($N', s, k$)
17:     if $b(k) < 0$ then
18:         $b(t_0) = -b(k)$
19:         Iterate($N', t_0, k$)
20:     end if
21: end if
```

Note that due to the minor changes or specifications, Lemma 47 also holds for all kinds of pseudo-demands with regard to the reduced algorithm for border stations and Algorithm 9. Deletion of pseudo demands, especially storage nodes, also requires only minor modifications of Algorithm 8 as in Algorithm 10 for which Lemma 49 also holds.

Algorithm 10 *Remove Operative Storage*

```
Input: $N^* = (V, A), f^*, s, k, t_0$
Output: $N' = (V', A'), f'$
1: $A' = A \setminus \{(t_0, k)\}$, $N' = (V', A')$
2: $b(k) = n_k$
3: Iterate($N', k, t_0$)
4: $b'(s) = -b(k)$
5: Iterate($N', k, s$)
6: $A'_k = \{a = (i, k') | a \in A'\}, A_k = \{a = (i, k) | a \in A'\}$
7: $A' = A' \setminus (A'_k \cup A_k \cup \{(k', k)\})$
8: $V' = V \setminus \{k', k\}$
```

6.4 NODE CHANGES

In practice changes of node attributes mostly affect the number of cars in a supply s_i or demand d_j, which correspond directly to the capacities on the model arcs (s, s_i) or (d_j, t) respectively. The necessary changes in the (residual) network and the (optimal) flow can be viewed as generalizations of node addition and deletion.

If n_i of supply node s_i is increased to n'_i, either $(s, i) \in A^*$ and we increase its capacity by $n'_i - n_i$ or only $(i, s) \in A^*$. Here we must again distinguish the *if*-case $(p(i) \geqslant p(s))$ and *else*-case $(p(i) < p(s))$ of Algorithm 5 according to the (already given) potential $p(i)$. In the former case the arc (s, i) is added with capacity $n'_i - n_i$ and appropriate (reduced) cost and the additional supply is only allocated if there is rest demand, as rerouting flow cannot decrease the total cost. In the latter case arc (i, s) is added by sending $n'_i - n_i$ units of flow along (s, i), which then drops out of N^*. The resulting temporal excess at i is either routed to one or more sinks if there is rest demand, which increases the flow value and keeps optimality. Or flow is rerouted from i to s via another (most expensive) supply node to regain the optimal flow solution given the additional excess at i without increasing the flow value. Increasing a demand follows analogously along the lines of Algorithm 6.

Decreasing the number of supplied or demanded cars by n only requires the following changes in Algorithms 7 and 8: The respective model arc (i, s) or (t, j) is not deleted, but its capacity is appropriately decreased by n and the temporal imbalances at i or j respectively are set to n instead of n_i or n_j respectively.

6.5 BATCH REOPTIMIZATION

In the previous sections we covered the addition and deletion as well as increase and decrease in the number of cars with respect to a single supply or pseudo demand node. As the addition of a new node requires all potentials of matching nodes to be already fixed, this operation can not be carried out simultaneously for supply and demand nodes. Further, all four operations possibly require the introduction of temporal imbalances at source, sink or data nodes (*else*-cases), which can result in undesired flows if the operations are mixed.

We can simultaneously add (increase, remove, decrease) all currently changed supply (demand) nodes. We therefore prepare the local network modifications for all changed supply (demand) nodes as in Algorithm 5 or 7 (6 or 8) and call the redefined *Iterate* procedure 11, which handles a set of sources and sinks rather than a single node in each case. The *Iterate* procedure is thus either called with a single source and a set of sinks or vice versa. In each case, the respective model node is the singleton and thus the predefinition of source-sink-pairs still suffices to ensure Corollary 44. On the other hand, out of the set of changed supply or demand nodes, the order of flow augmentation is determined by the *Successive Shortest Path* algorithm and thus ensures optimality of the resulting (maximal) feasible flow solution.

Algorithm 11 *Set-Iterate Successive Shortest Path*

```
Input: $N(f) = (V^f, A^f), S, T$
Output: $f^t, N^t(f^t) = (A^t, V)$
1: while $\exists \pi^*_{st}, s \in S, t \in T$ do
2:     Determine shortest path $\pi^*_{st}$ and $d(s,t)$ w.r.t. $c_r$ in $N(f)$
3:     Update $p(v) = p(v) - d(s,v) + d(s,t)$ for each permanently labeled
       $v \in V$
4:     $\delta := \min\{b(s), -b(t), \min_{a \in \pi^*_{st}} u_r(a)\}$
5:     Augment $\delta$ units of flow along $\pi^*_{st}$, obtaining $f'(a)$
6:     $\forall a = (u,v) \in A^f : c_r(a) = c(a) - p(u) + p(v)$
7:     $\forall a = (u,v) \in A^f : u_r(a) = u(a) - f'(a)$
8:     Update $N(f)$ to $N(f')$
9: end while
```

By Lemmas 46, 47, 48 and 49 follows directly:

Theorem 50 (Node-based Reoptimization) *Given a set S of supply nodes s_i, which are added, deleted, increased or decreased in the number of cars by n_i and a set D of pseudo demand nodes $\hat{d}_j$, which are added, deleted, increased or decreased in* *the number of cars by* n_j, *the reoptimization Algorithms 5, 6, 7, 8 with application of Procedure 11 find a minimum cost flow on the network* N' *altered by* S *and* D, *given* $N^* = (V^*, A^*)$ *for a minimum cost flow* f^* *in the original network* $N = (V, A)$ *in* $O(n_S + n_D)$ *Successive Shortest Path iterations with* $n_S = \sum_{s_i \in S} n_i$ *and* $n_D = \sum_{\hat{d}_j \in D} n_j$ *after preparation of time* $O(m)$, *such that the strong priority order of demand satisfaction is kept.*

7 A NETWORK MODEL FOR THE HETEROGENEOUS DP

In this chapter, we introduce a generalized network model N_I^g for heterogeneous instances I of the (DP) (Section 7.1) and discuss the complexity of the generalized integral minimum cost flow problem with respect to such networks N_I^g. The latter obviously remains NP-complete, given the complexity result of Corollary 35.

Total instances I can be transformed to homogeneous instances I' (see Section 7.2), such that an integral minimum cost flow f^* in the classical network $N_{I'}^g$ can be obtained in polynomial time. A more general transformation from special generalized networks to classical networks is known from literature [51].

We show that the flow f^* provides an optimal (LP)-feasible derived distribution $\mathcal{D}^*$ of bounded fractionality for the original instance I. By characterization of *totalizable* instances of the (DP), this approach is extended to all instances from the application, for which we obtain optimal (LP)-feasible derived distributions of bounded fractionality $\beta = 2$. Those distributions can be obtained directly from N_I^g by the modified *Successive Shortest Path* algorithm for generalized minimum cost flows we present in Section 7.4 after a brief review of literature on generalized flow problems in Section 7.3.

The half-integral optimal (LP)-feasible distributions are the basis of polynomial time approximations and good heuristics for the heterogeneous (DP) in practice, which we present in Chapter 8.

7.1 A GENERALIZED FLOW NETWORK

We construct a generalized network model N_I^g similar to the network model N_I for homogeneous instances in Chapter 5. Let $I = \{S, D, O, B, \mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$ be a heterogeneous instance of the (DP). Then $N_I^g = (V_I, A_I)$ and capacity $u^g = u$, cost $c^g = c$ and balance function $b^g = b$ are the same as before. Additionally, we define a multiplicator function $\mu : A_I \to \mathbb{Q}$ on the arcs of N_I^g. As we show in Section B.1, there is always an equivalent generalized network employing node multipliers. We set the multipliers $\mu(s, i) = 1$ and $\mu(j, t_p) = 1, p \in \{0, 1, 2\}$. For all arcs $(i, j) \in A_T$, the multiplier $\mu(i, j)$ is set to $\nu(t_i, t_j)$, the relative valency of car type t_i to car type t_j.

Definition 51 (Generalized Distribution Network N_I^g) *Let*

$$I = \{S, D, O, B, \mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$$

be a heterogeneous instance of the (DP). Then the distribution network $N_I^g = (V_I, A_I)$ *with capacity* $u^g : A_I \to \mathbb{N}$*, cost* $c^g : A_I \to \mathbb{N}$*, multiplier* $\mu : A_I \to \mathbb{N}$ *and balance* $b^g : V_I \to \mathbb{N}$ *is defined as described above.*

An integral feasible flow f in N_I^g provides a feasible derived distribution $\mathcal{D}^f(I)$ (see Definition 37) disregarding the strong priorities by analogous arguments as in Lemma 38 and 39 (Chapter 5). With Definition 40 and analogous arguments as in Lemma 41, we obtain the following corollary from Theorem 42:

Corollary 52 (Generalized Flow-Distribution-Equivalence) *For each feasible flow* f *in* N_I^g *there is an (LP)-feasible derived distribution* $\mathcal{D}^f$ *(disregarding strong priorities) with respect to* I *and vice versa. The distribution* $\mathcal{D}^f$ *is feasible if and only if* f *is integral and optimal if and only if* f *is an integral minimum cost flow in* N_I^g*.*

So far, if a feasible distribution exists an integral solution to the generalized minimum cost flow problem on N_I^g provides an optimal distribution to any heterogeneous instance I of the (DP). Unfortunately, integrality is computationally harder to guarantee in the general case. The complexity of the integral maximal generalized flow problem (a special case of the minimum cost flow problem with cost zero) is long known to be NP-complete for general networks [107]. It still holds for the rather special N_I^g arising from instances I of the (DP). This is not surprising, given the NP-completeness result for the heterogeneous (DP) (see Section 4.2) and the equivalence between integral feasible flows and feasible distributions (disregarding strong priority order).

An argument for the computational complexity result from the field of linear programming is that introducing flow multipliers into a network generally destroys total unimodularity of the corresponding node-arc incidence matrix. Therefore, without explicit integrality constraints the optimum needs not be integral, even if all input values are integral. Figure 21 shows an instance of the problem without any feasible integral solution.

Hence, the existence of a feasible distribution does not only require sufficient storage and only non-isolated supplies. Consider an instance I and the precomputed maximal generalized flow f_{max} in N_I^g as the one we use to adjust $b(s)$ and $b(t_p), p \in \{0, 1, 2\}$. If $\sum_{a \in A_S} f_{max}(a)$ is not integral, no feasible solution exists for I, although there is an (LP)-feasible solution.

We also investigated the complexity of the integral generalized maximum resp. minimum cost flow problem independent of the (DP) in other special networks (see Appendix B). For example, in bipartite generalized networks $N = (V, A)$ with one source and two sinks and a multiplicator function $\mu : A \to K \subset \mathbb{N}$ the problem remains NP-complete (see Section B.2). If $K = \{1, k\}$ and the structure of N is regular in the sense that for each node

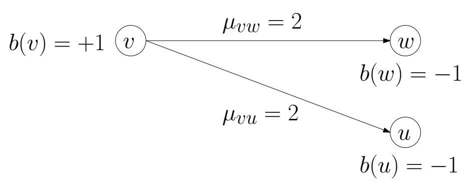


Figure 21: The only valid flow is fractional.

the multipliers on all outgoing arcs are identical, we obtain minimum cost generalized flows of bounded fractionality $\beta = k$ in polynomial time (see Section B.3).

On the one hand, such networks occur for total k-heterogeneous instances of the (DP). On the other hand, such networks have a so-called k-regular [8] node-arc incidence matrix as we prove in Section B.4. Therefore, the constraint matrix of an LP modeling the minimum cost flow in such a network is k-regular as well. With the results of [8], this provides an alternative proof for the bounded fractionality of a solution (without explicit integrality constraint) in this case.

Due to the complexity of the (DP) and practical runtime requirements we want to temporarily drop the integrality constraint on the generalized flow problem in N_I^g. In the following section, we characterize those instances I for which we then obtain minimum cost flows f of bounded fractionality in N_I^g in polynomial time. The derived distribution $\mathcal{D}^f$ is always an (LP)-feasible distribution of bounded fractionality with respect to I. For instances from the application we thus obtain half-integral derived distribution, which are rounded to feasibility in Chapter 8.

7.2 BOUNDED FRACTIONALITY

For a generalized network, a minimum cost flow of bounded fractionality β does not need to exist. This is especially true if we want β to be independent of the network size, namely $n = |V|$ and $m = |A|$. We give an example with $\beta = \frac{1}{2^n}$ in a network $N = (V, A)$ with $3n$ nodes in Figure 22: At each level l, the maximum possible flow is carried from source s_l to sink t_l of the respective level. We start from level 1, where no other way of balancing t_1 is possible. The rest excess of s_l must then be transferred to t_{l+1}. There it causes a fractional rest deficit of $\frac{1}{2^l}$ which is balanced by $\frac{1}{2^{l+1}}$ flow out of s_{l+1} due to the flow multiplier of 2.

The remaining excess of source s_{l+1} is then $1 - \frac{1}{2^{l+1}}$ and so on. For this part of the graph, we need $2n+1$ vertices to create a flow of $\frac{1}{2^n}$ units. We create a path of $n-1$ vertices connecting $s_{\frac{n}{3}}$ and $t_{\frac{n}{3}+1}$ on the last level with an arc multiplier of 2 on each arc. Thus the remaining excess is completely consumed by the last integral deficit. In this way we ensure a valid flow

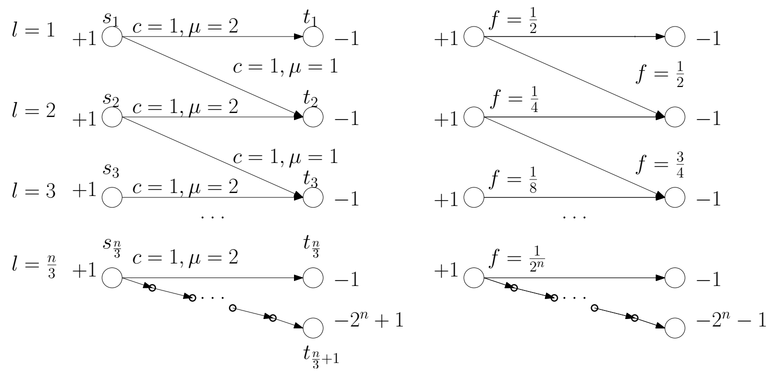


Figure 22: An example of arbitrary small fractions of flow.

solution and the integrality of all demands and excesses by exclusive use of multipliers 1 and 2. The same construction can be applied for any fixed multiplier k. The fractionality of a maximum flow solution can only be bounded in the number of vertices. Additional arcs with high costs modify the example to apply also to the minimum cost flow problem and show that we cannot obtain a flow of bounded fractionality without additional constraints.

In the following we show how to obtain (LP)-feasible distributions of bounded fractionality with minimum costs for total instances I of the (DP) in polynomial time. We briefly call such distributions optimal fractional distributions for I. Further, we extend the considered set of instances by characterization of totalizable instances. The latter include the instances from our application.

7.2.1 *Bounded Fractionality of total Instances*

Let $I = \{S, D, O, B, \mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$ be a total instance such that p is the least common multiple of the nominators and q is the least common multiple of the denominators of total valencies $\nu(t_i), s_i \in S$. For the moment let $O = B = \emptyset$. Then we define an associated instance I' as follows.

Definition 53 (Instance I')

$$I' = \{S', D, O, B, \mathcal{T}, \mathcal{S}', \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$$

with

$$S' = \{(l_i, t_i, c_i, \nu(t_i) \cdot n_i, g_i, f_i, u_i, b_i, o_i, \nu_i) | s_i \in S\}$$

and

$$\mathcal{S}' = \{(t_s, t_d) | \sigma_{sd} \in \mathcal{S}\}.$$

Obviously, I' is a homogeneous instance, thus we apply the results of Chapter 5, but modify the cost function of N'_I to $c'(i,j) = \nu(t_i)c(i,j)$ for $(i,j) \in A_T$ and $c'(u,v) = 0$ for $(u,v) \in A_I \setminus A_T$. As the number of supplied cars not necessarily remains integral, we scale the capacity and balance function by q to $u_q(v,w) = qu(v,w)$ and $b_q(v) = qb(uv)$ and obtain the network $N'_I(q)$ with integral values. We thus obtain an integral minimum cost flow f^*_q in $N'_I(q)$ in polynomial time.

Lemma 54 (Equivalence of f' and f_q) *Let $f' = \frac{1}{q}f_q$. Then f' is a q-fractional feasible flow in N'_I if and only if f_q is an integral feasible flow in $N'_I(q)$.*

Proof: Let f_q be an integral feasible flow in $N'_I(q)$. Due to linearity of capacity and balance constraints, the flow f', obtained by scaling f_q with $\frac{1}{q}$, is feasible for N'_I, because it is obtained from $N'_I(q)$ by scaling capacity and balance by the same factor. Further, as f_q is integral, f' is q-fractional. Let f' be a q-fractional feasible flow in N'_I. Then f_q is obtained by scaling f' with q, while $N'_I(q)$ is obtained from N'_I by scaling the capacity and balance function with that particular q. Hence, f_q is feasible in $N'_I(q)$ by linearity of the scaled functions. Finally, as f' is q-fractional, f_q is an integral feasible flow in $N'_I(q)$. □

Due to the equivalence of f_q and f' as given by Lemma 54 and by linearity of the cost functions $c(f')$ and $c(f_q)$, the following holds:

Corollary 55 (Minimum Cost Flow f^*) *We obtain a q-fractional minimum cost flow $f^* = \frac{1}{q}f^*_q$ in N'_I via scaling of a minimum cost flow f^*_q in $N'_I(q)$, which can be computed in polynomial time.*

From f', we derive an optimal q-fractional distribution $\mathcal{D}(I')$ for I' by Corollary 52. Intuitively, the difference between instances I and I' is that we interpret flow units as fractions or tuples of freight cars appropriate to the valency of the supply in I', instead of identifying flow units with cars as in I. We derive an optimal pq-fractional distribution $\mathcal{D}(I)$ for I from $\mathcal{D}(I')$ by transforming the fractions or tuples of freight cars back to actual freight cars as follows.

Let I be a total heterogeneous (DP) instance with valencies $\nu(t), t \in T_S$ and I' as in Definition 53. Further, let $\mathcal{D}^*(I')$ be the optimal q-fractional distribution for I' derived from a minimum cost flow $f^* = \frac{1}{q}f^*_q$ in N'_I, with f^*_q in $N'_I(q)$ being computed by Algorithm 2 (thus obeying strong priority order).

Definition 56 (q-Derived Distribution) *We define a q-derived distribution $\mathcal{D}^*(I)$ as:*

$$\mathcal{D}^*(I) = \{\delta = (s_i, \hat{d}_j, n_{ij} = \frac{1}{\nu(t_i)} n'_{ij}) | \delta'_{ij} \in \mathcal{D}^*(I')\}.$$

Theorem 57 (Optimal pq-fractional distribution $\mathcal{D}(I)$) *The* q*-derived distribution* $\mathcal{D}^* = \mathcal{D}^*(I)$ *is an optimal* pq*-fractional distribution for* I.

Proof: Each assignment $\delta'_{ij} \in \mathcal{D}^*(I')$ is q-fractional feasible for I': s_i matches $\hat{d}_j$ with respect to I' and, by construction, also with respect to I. Further, $n_i = \frac{1}{\nu(t_i)} n'_i \geqslant n_{ij} = \frac{1}{\nu(t_i)} n'_{ij}$ follows from $n'_i \geqslant n'_{ij}$ and $n_j \geqslant \nu(t_i, t_j) \frac{1}{\nu(t_i)} n'_{ij}$ from $n_j \leqslant \nu'(t_i, t_j) n'_{ij}$ (with $\nu'(t_i, t_j) = 1$). As $\mathcal{D}^*(I')$ is q-fractional and we obtain n_{ij} from n'_{ij} by division by the reciprocal of total valencies, every assignment $\delta \in \mathcal{D}^*(I)$ is pq-fractional feasible (with p, q as defined previously). Further, with and $O = B = \emptyset$,

$$
\begin{aligned}
\nu(t_i) n_i = n'_i = \sum\nolimits_{\delta'_{ij} \in D^*(I')} n'_{ij} & & = \\
\sum\nolimits_{\delta_{ij} \in D^*(I)} \nu(t_i) n_{ij} & & \Leftrightarrow \\
n_i = \sum\nolimits_{\delta_{ij} \in D^*(I)} n_{ij} & & \forall s_i \in S \\
n_j = n'_j \geqslant \sum\nolimits_{\delta'_{ij} \in D^*(I')} \nu'(t_i, t_j) \cdot n'_{ij} & & = \\
\sum\nolimits_{\delta_{ij} \in D^*(I)} \nu(t_i) \frac{1}{\nu(t_i)} \cdot n'_{ij} & & \Leftrightarrow \\
n_j \geqslant \sum\nolimits_{\delta_{ij} \in D^*(I)} \nu(t_i) \cdot n_{ij} & & = \\
\sum\nolimits_{\delta_{ij} \in D^*(I)} \nu(t_i, t_j) \cdot n_{ij} & & \forall d_j \in D.
\end{aligned}
$$

Moreover, $\mathcal{D}^*(I')$ is minimum with respect to $c'(D(I')) = \sum_{\delta' \in D^*(I')} c'_{\delta'}$ respecting strong priority order and $c'_{\delta'} = n'_{ij} \cdot c'_{ij} = n'_{ij} \cdot \nu(t_i) \cdot c_{ij} = n_{ij} \cdot \frac{1}{\nu(t_i)} \cdot \nu(t_i) \cdot c_{ij} = n_{ij} \cdot c_{ij} = c_\delta$. Thus $D^*(I)$ is minimum with respect to cost c (respecting strong priority order). □

The two drawbacks of the above method for solving heterogeneous instances is that on the one hand, not all instances allow the definition of total valencies for all supply car types and on the other hand, the feasibility of a solution $D^*(I)$ derived from $D^*(I')$ depends on the integrality of the reciprocal of total valencies ($p = 1$) and the scaling factor ($q = 1$). On the other hand, with regard to approximation strategies, total instances have the advantage of bounded fractionality solutions which can be obtained in polynomial time.

Let I be a total instance of the (DP) with total valencies $\nu(t), t \in T_S$ and let p be the least common multiple of all nominators, q the least common multiple of all denominators of total valencies.

Corollary 58 (Bounded Fractionality of Total Instances) *We obtain an optimal pq-fractional distribution $\mathcal{D}^*(I)$ with regard to I respecting strong priorities in polynomial time.*

Especially, all total k-heterogeneous instances I_k have bounded fractionality k: The total valencies are from the set $\{1, k, \frac{1}{k}\}$, such that $p = k$ and $q = k$ and due to Corollary 58 the fractionality of $\mathcal{D}^*(I_k)$ would be bounded by k^2. This bound proves too abrasive: only assignments $d_{ij} \in \mathcal{D}^*(I_k)$ for supplies s_i with $\nu(t_i) = k$ are scaled twice by k during the computation of the q-derived distribution for I_k. Due to scaling and linearity f_q^* is a multiple of k on corresponding arcs (i, j) such that $\mathcal{D}^*(I_k)$ is in fact k-fractional. The latter is formally shown in Sections B.3 and B.4 with respect to the k-regularity of the associated networks $N'_{I_k}(q)$. Further, we explicitly show half-integrality of optimal fractional distributions for the non-total 2-heterogeneous instances which occur in the application in Section 7.2.3.

In the following section, we define and characterize totalizable instances, which include those practical instances. We prove that totalizable instances also allow pq-fractional distributions to be obtained in polynomial time.

7.2.2 *Bounded Fractionality of totalizable Instances*

Most (k-)heterogeneous instances are not generically total. For some instances, including the practical 2-heterogeneous instances, we can achieve relative valencies to be equal for each occurring supply car type by introducing an artificial total valency function w on occurring demand car types. Let therefore

$$I = \{S, D, O, B, \mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$$

be a (non-total) instance of the heterogeneous (DP) and let T_S and T_D be the sets of occurring supply and demand types $T_S = \{t = t_s | \sigma_{sd} \in \mathcal{S}, \exists s_i \in S : t_i = t_s\}$ and $T_D = \{t = t_d | \sigma_{sd} \in \mathcal{S}, \exists d_j \in D : t_j = t_d\}$.

Definition 59 (Totalizable Instance) *I is totalizable if and only if there is a function $w : T_D \to \mathbb{R}$ such that the instance*

$$I_t = \{S, D_t, O, B, \mathcal{T}, \mathcal{S}_t, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$$

with

$$D_t = \{(l_j, t_j, f_j, c_j, w(t_j) \cdot n_j, g_j, a_j, r_j, e_j, z_j, w_j, p_j) | d_j \in D\}$$

and

$$\mathcal{S}_t = \{(n_s t_s, w(t_d) \cdot n_d t_d) | \sigma_{sd} \in \mathcal{S}\}$$

is total.

To characterize the totalizable instances of the heterogeneous (DP), we represent the relative valencies of supply car types in the following way:

Definition 60 (Substitution Graph) *The substitution graph* $G_{\mathcal{S}}(I) = (V_{\mathcal{S}}, E_{\mathcal{S}})$ *of* I *consists of a node* $s \in V_S$ *for each supply car type* $t_s \in T_S$ *and a node* $d \in V_D$ *for each demand car type* $t_d \in T_D$ *and* $V_{\mathcal{S}} = V_S \cup V_D$. *Further,* $E_{\mathcal{S}} = \{(s,d) | (n_s t_s, n_d t_d) \in \mathcal{S}\}$ *and each edge* (s,d) *is annotated with the relative valency* $\nu(s,d) = \nu(t_s, t_d) = \frac{n_d}{n_s}$.

For the substitution graph $G_{\mathcal{S}}(I)$ of a non-total instance I of the (DP), there is a node s in $V_{\mathcal{S}}$ such that two incident edges (s,d), (s,d') have different values $\nu(s,d) = \frac{n_d}{n_s}$ and $\nu(s,d')\frac{n'_d}{n_s}$. To set both relative valencies to the same value, we scale n'_d, which represents a number of demanded cars of type t_d, by $w(t_d) = \frac{n_d}{n'_d}$. Scaling n_d serves to represent a number of cars by a multiple of fractions or tuples of the cars similar to the treatment of supplied cars in Definition 53.

To maintain the original relative valencies with respect to the real number of cars, we have to scale n'_d in the same way for all $\sigma_{sd'} \in \mathcal{S}$. We replace $n_d t_d$ by $w(t_d) n'_d t_d$ in all $\sigma_{sd'} \in \mathcal{S}$ and update $G_{\mathcal{S}}(I)$, if relative valencies $\nu(t'_s, t_d)$ change. Possibly new pairs of incident edges (s', d), (s', d'') with different values $\nu(s', d) = \frac{w(t_d) n_d}{n'_s}$ and $\nu(s', d'')\frac{n''_d}{n'_s}$ are created. We proceed with the definition of a total valency $w(t''_d)$ for demand car type t''_d, if and only if t''_d has not permanently received a different total valency yet. If this procedure terminates without conflicts, I is totalizable.

Let I again be a (non-total) instance of the heterogeneous (DP) with substitution rule $\mathcal{S}$. Further, let the value $\nu(p)$ of a path $p = e_1, \ldots, e_{r_p} \subseteq E_{\mathcal{S}}$ be defined as:

$$\nu(p) = \begin{cases} \frac{\nu(e_1)\nu(e_3)\ldots\nu(e_{2r-1})}{\nu(e_2)\nu(e_4)\ldots\nu(e_{2r})}, & \text{if } r_p = 2r, r \in \mathbb{N} \\ \frac{\nu(e_1)\nu(e_3)\ldots\nu(e_{2r+1})}{\nu(e_2)\nu(e_4)\ldots\nu(e_{2r})}, & \text{if } r_p = 2r+1, r \in \mathbb{N} \end{cases}$$

Note that the value $\nu(c)$ of a cycle is always determined as that of an even length path as $G_{\mathcal{S}}$ is bipartite.

Theorem 61 (Characterization of Totalizable Instances) I *is totalizable if and only if* $G_{\mathcal{S}}(I)$ *contains only cycles* c *with* $\nu(c) = 1$.

Proof: Let $(s,d), (s,d') \in E_{\mathcal{S}}$ have different values $\nu(s,d)$ and $\nu(s,d')$. Set $w(t_d) = 1$ and $w(t'_d) = \frac{\nu(s,d)}{\nu(s,d')} = \frac{n_d n_s}{n'_d n_s} = \frac{n_d}{n'_d}$, such that

$$\nu'(s,d) = \frac{n_d}{n_s} = \frac{n_d}{n'_d}\frac{n'_d}{n_s} = w(t'_d)\frac{n'_d}{n_s} = \nu'(s,d'),$$

which implies that both values are identical with respect to the modified substitution rule $\mathcal{S}_t$ with total valency $w(t'_d)$ of type t_d as in Definition 59. Consequently, all edges incident to d' receive new values $\nu'(s', d') =$

$v(t'_d)v(s'd') = \frac{v(s,d)v(s'd')}{v(s,d')}$. For incident edges (s', d'') and (s', d') with different values, $w(t''_d)$ is set to $\frac{v'(s,d)v'(s'd')}{v'(s,d')v'(s',d'')}$. Continuing the assignment of total valencies as above, $w(t'_d) = v(p)w(t_d)$ for a path p from d to d'.

As long as for any node s' for which a path from d' to s' exists, two incident edges (s', d') and (s', d'') have different values, proceed as above. If no such node s' exists, either I is total with respect to $\mathcal{S}_t$ and the values $w(t_d)$ (all total valencies for demand types, which are not yet explicitly set, are set to one) or $G_\mathcal{S}$ is not connected.

In the latter case, remove the considered connected component of $G_\mathcal{S}$ and start as above in a remaining component. Thus, starting from two incident edges $(s, d), (s, d')$ with different values and setting $w(t_d) = 1$ in each connected component, each value $w(t''_d) \in V_D$ is determined as $w(t''_d) = v(p)w(t_d)$ if p is a unique path from d to d'' or if $v(p)$ is equal for all paths from d to d''. If both is not the case, there are at least two different paths p, p' from d to d''. Let w.l.o.g. p and p' be edge disjoint. (Otherwise, we redefine d to be the last common node in V_D of p and p' and d' to be the first common node in V_D of p and p' after d.) Then $c = p\bar{p}'$ with $\bar{p}'$ denoting the reverse of p' is a cycle in $G_\mathcal{S}$ which contains d. The reverse of p' is also associated with the inverse value $v(\bar{p}') = \frac{1}{v(p')}$ and the value of the cycle c is $v(c) = v(p)v(\bar{p}') = v(p)\frac{1}{v(p')}$. Due to the assumption that $v(p) \neq v(p')$, c is a cycle with $v(c) \neq 1$. □

For a totalizable instance I of the (DP), we obtain an optimal pq-fractional distribution $\mathcal{D}^*(I_t)$ respecting strong priority order for the associated total instance I^t in polynomial time by Corollary 58 and the results of Chapter 5. Let I be a totalizable heterogeneous (DP) instance with $w : T_D \to \mathbb{N}$, and I^t as in Definition 59. Further, let $\mathcal{D}^*(I_t)$ be an optimal pq-fractional distribution for I_t.

Theorem 62 (Equivalence of $\mathcal{D}(I^t)$ and $\mathcal{D}(I)$) *$\mathcal{D}^*(I_t)$ is a pq-fractional distribution for I.*

Proof: Each assignment $\delta_{ij} = (s_i, \hat{d}_j, n_{ij}) \in \mathcal{D}^*(I_t)$ is pq-fractional feasible for I_t which means s_i matches $\hat{d}_j$, $n_i \geqslant n_{ij}$ and $v_t(t_i, t_j)n_{ij} = v(t_i, t_j)w(t_j)n_{ij} \leqslant w(t_j)n_j$ from which follows $v(t_i, t_j)n_{ij} \leqslant n_j$. As $\mathcal{S}_t$ contains for each $(n_s t_s, n_d t_d) \in \mathcal{S}$ a tuple $(n_s t_s, w(t_d)n_d t_d)$, for each assignment δ_{ij}, s_i also matches $\hat{d}_j$ with respect to I and δ_{ij} is pq-fractional feasible for I. Further, $\sum_{\delta_{ij}\in\mathcal{D}^*(I^t)} n_{ij} = n_i$ holds $\forall s_i \in S$ and

$$\begin{aligned}\sum_{\delta_{ij}\in\mathcal{D}^*(I^t)} \nu_t(t_i, t_j)n_{ij} & & = \\ \sum_{\delta_{ij}\in\mathcal{D}^*(I^t)} \nu(t_i, t_j)w(t_j)n_{ij} \leqslant w(t_j)n_j & & \Leftrightarrow \\ \sum_{\delta_{ij}\in\mathcal{D}^*(I^t)} \nu(t_i, t_j)n_{ij} \leqslant n_j & & \forall d_j \in D\end{aligned}$$

Hence (with $O = B = \emptyset$), $\mathcal{D}^*(I_t)$ is a pq-fractional distribution for I. □

Since $c(D^*(I_t)) = c(D^*(I))$ minimizes the distribution cost c respecting strong priority order, Theorem 62 generalizes Corollary 58 to totalizable instances. Let I be a (non-total) heterogeneous instance of the (DP), totalizable with $w : T_D \to \mathbb{R}$, which results in total valencies $\nu_t(t_i), s_i \in S$ for the instance I_t. Further, let p be the least common multiple of all nominators, q the least common multiple of all denominators of total valencies ν_t.

Corollary 63 (Bounded Fractionality of totalizable Instances) *We obtain an optimal pq-fractional distribution $\mathcal{D}^*(I) = \mathcal{D}^*(I_t)$ with respect to I regarding strong priority order in polynomial time.*

A more general result for equivalence of certain generalized flow problem instances and classical flow problems is given in [51]. It is shown that any generalized network problem whose node-arc-incidence matrix does not have full row rank is equivalent to a pure network problem. The prove is given via an algorithm employing row and column manipulation on the incidence matrix of the network. It would be interesting to investigate in how far the determination of the values $w(t_d)$ with respect to a given totalizable instance are equivalent to the necessary row and column operations and if the determination of $w(t_d)$ in generalized networks can be shown to be a graph-theoretic pendant to the above result.

7.2.3 *Application Instances*

In this section we show that the 2-heterogeneous (DP) instances

$$I_a = \{S, D, O, B, \mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$$

which occur in practice are totalizable. Moreover, we obtain half-integral optimal solutions $\mathcal{D}^*(I_a)$ in polynomial time. The substitution rule $\mathcal{S}$ of such instances I_a contains the following subsets:

$$\mathcal{S}|xyz = \{(t_x, t_x), (t_y, t_y), (2t_x, t_z), (t_y, t_z)\}$$

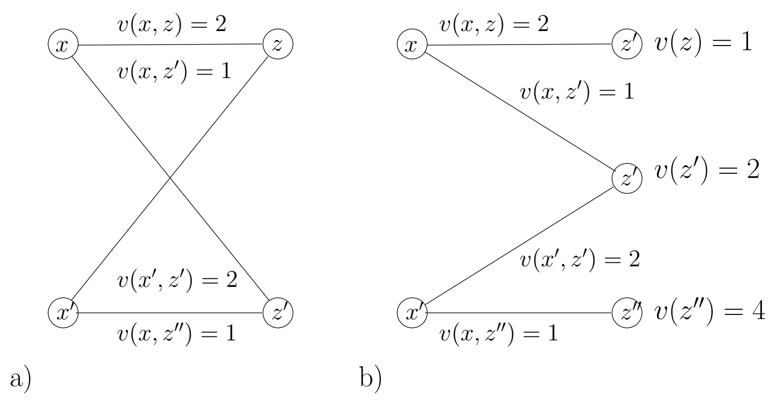


Figure 23: Subgraphs of $G_\mathcal{S}$: a) A non-totalizable instance. b) Increased bound of fractionality

and

$$\mathcal{S}'|xyz = \{(t'_x, t'_x), (t'_y, t'_y), (2t'_x, t'_z), (t'_y, t'_z)\}.$$

Any associated instance $I_r = \{\mathcal{S}, D, O, B, \mathcal{T}, \mathcal{S}_r, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$ with $\mathcal{S}_r = \mathcal{S} \setminus (\mathcal{S}|xyz \cup \mathcal{S}'|xyz)$ is homogeneous. Note that I_a is not total as $v(t_x, t_x) = 1 \neq v(t_x, t_z) = \frac{1}{2}$ and $v(t'_x, t'_x) = 1 \neq v(t'_x, t'_z) = \frac{1}{2}$, but all other involved supply car types t have total valency $v(t) = 1$. Still $G_\mathcal{S}(I_a)$ possibly contains the subgraphs depicted in Figure 23:

The graph in Figure 23 a) contains a cycle $c = (x,z)(z,x')(x',z')(z',x)$ with $v(c) = \frac{2 \cdot 2}{1 \cdot 1} = 4 \neq 1$, such that the corresponding instance is not totalizable. The graph in Figure 23 b) contains no such cycle, but $w(z'') = 4$, resulting in $v_t(t'_x, t''_z) = \frac{1}{4}$ such that $\mathcal{D}^*(I_t)$ of the corresponding totalized instance I_t can only guaranteed to be 4-fractional, so far.

However, $G_\mathcal{S}(I_a)$ does not contain the subgraphs of Figure 23, as $G_\mathcal{S}(I_a)$ is not connected, the edges corresponding to the subsets $\mathcal{S}|xyz$ and $\mathcal{S}'|xyz$ are in different connected components of $G_\mathcal{S}(I_a)$ and neither component contains a cycle c with $v(c) \neq 1$. Thus the instances I_a are totalizable by Theorem 61. Further, I_a is totalizable with $w : T_D \to \{1, 2\}$ by setting $w(t_x) = w(t'_x) = 1$, $w(t_y) = w(t'_y) = w(t_z) = w(t'_z) = 2$ and extending w to all other demand types with corresponding nodes in the same connected component of $G_\mathcal{S}(I_a)$ as in Theorem 61. Demand types t with corresponding nodes in other connected components of $G_\mathcal{S}(I_a)$ receive total valency $w(t) = 1$.

Theorem 64 ($\mathcal{D}^*(I_a)$ is half-integral) *Let I_a be a totalizable 2-heterogeneous instance of the (DP) as above. Then we obtain a half-integral optimal distribution $\mathcal{D}^*(I_a)$ in polynomial time.*

Proof: I_a is totalizable with $w : T_D \to \{1, 2\}$ by setting $w(t_x) = w(t'_x) = 1$, $w(t_y) = w(t'_y) = w(t_z) = w(t'_z) = 2$. Thus the totalized instance I_t has total valencies $v_t(t_x) = 1$ and $v_t(t_y) = v_t(t_z) = \frac{1}{2}$ such that $p = 1$, $q = 2$. Then by Corollary 63 $\mathcal{D}^*(I_a) = \mathcal{D}^*(I_t)$ is half-integral and can be obtained in polynomial time. □

7.2.4 *Operative Storage and Border Stations*

We so far assumed $O = B = \emptyset$. With respect to operative storage $o_k \in O$, the substitution analogue is the storage rule $\mathcal{O}$, which does not specify different valencies. This is natural as storage is usually mix-type and does not demand any special types due to transport requirements for certain goods. For each $o_k \in O$, the capacity n_k is computed as an average over the car length of different car types weighted with the number of those cars in the stock (see Section 2.4.2). For a totalized heterogeneous instance I_t, the capacity of a storage can be recomputed as an average over the car length of different car types t weighted not only with the number of those cars in the stock, but also with their total valency $w(t)$. Thus the recomputed storage capacity enables capacity control in the totalized solution as well as the formerly computed capacity does in the original solution for I.

In contrast, the border station capacities cannot be redefined in such a way: A border station possibly allows several distinct car types with different total valencies per time interval, but the valency has no effect on the shunting process and thus must not influence the capacity. On the other hand, in the application, so far no foreign car types are involved in heterogeneous substitution. More specifically, for a practical 2-heterogeneous instance I_a with substitution rule $\mathcal{S}$, a node corresponding to a foreign car type only occurs in a different connected component of $G_{\mathcal{S}}$ than x, x', z and z'. Thus for instances I_a with non-empty sets O and B we also obtain half-integral optimal distributions in polynomial time applying the previous results of Section 7.2 on a slightly modified instance.

7.3 ALGORITHMS FOR GENERALIZED FLOW PROBLEMS

In this section, we survey algorithms for generalized flow problems as not all instances of the (DP) can be solved via classical flow models as described in the previous sections.

Besides the traditional network flow problems, the generalized flow model has also been studied since the publication of [39]. Early results were the extension of Ford and Fulkerson's primal-dual approach for minimum cost flows by Jewell [76, 75] and the development of the generalized network simplex algorithm by Dantzig [26], which both concentrate on generalized flows as a special case of linear programming. A dual version of the latter was developed more recently by Goldfarb, Jin and Lin to solve the generalized circulation problem in polynomial time [59].

In the nineteen-sixties and -seventies, the generalized flow model was approached in a strongly application oriented context from manufacturing, transportation, communication and financial analysis [97, 98, 50], but also in some broad [15, 30, 51] or limited [74] theoretical aspects.

Especially in [115], a strong relationship between generalized maximum flow algorithms and traditional minimum cost flow algorithms is established. In particular, setting the cost function $c(i,j)$ of an arc $a = (i,j)$ to $-\log \mu(i,j)$, where $\mu(i,j)$ is the arc multiplier on a, there is a 1-to-1-correspondence between flow generating cycles in the generalized maximum flow problem with multipliers μ and negative reduced cost cycles in the traditional minimum cost flow problem with cost c.

This might be considered as a kick off into the exploration of combinatorial algorithms to solve generalized network flow problems, as it allows adaption from known algorithms to the generalized case. In the late nineteen-eighties and early -nineties, Goldberg and Tarjan for example developed new and fast combinatorial algorithms for maximal flow and minimum cost flow or circulation problems respectively [52, 53, 54]. Those algorithms also find their application in the generalized circulation problem [55] and have been further improved by Goldfarb, Jin (and Orlin) [58, 60]. This was the basis for the above mentioned dual simplex approach ([59]).

On the other hand, previous generalized maximum flow algorithms turned out to be analogues already. For example, Onaga's algorithm [97] could be interpreted as a successive shortest path approach with the appropriately defined costs and Truemper's own algorithm is similar to the primal-dual algorithm for minimum cost flows. Another parallel can be drawn between the maximum capacity augmenting path algorithm proposed by Edmonds and Karp [33] for the maximum flow problem and the most helpful cycle cancelling algorithm of Barahona and Tardos [10] or the minimum mean cycle cancelling algorithm [53] for the minimum cost circulation in an adapted network. (In the latter, an arc (t, s) with negative cost is added while all other arcs receive cost zero.) Shigeno [110] also sketches a generalized maximum flow algorithm on this basis.

Yet, in the nineteen-nineties also frameworks (e. g.[20]) were established for generalized problems, which rely on solving a (relaxed or dual) linear program as a subroutine or at least testing its feasibility. Mostly such subroutines solve generalized shortest path, transshipment or traditional cut problems, which can nicely be formulated as linear programs with two variables per inequality (TVPI). Those can be solved polynomially in integers [78, 1]. The running times for solving (TVPI) have been continuously decreased for example [19, 70]. Specially structured (LP) formulations also allow for half-integral optimal solutions obtainable in polynomial time and a thereon based 2-approximation [69, 68]. However, up to now it seems impossible for the generalized minimum cost flow problem to be formulated in that way. While most research is done into the generalized maximum flow problem, there are also possibilities to extend this work to minimum cost flows.

Different formulations for the generalized problem variants arise for example with regard to excesses and demands. In this context we have new structures in the generalized model like gainy (flow-creating) and lossy (flow-destroying) cycles. This leads to an extended decomposition theorem for generalized elementary flows ([61, 55]).

Concerning running times, since the development of the ellipsoid [82] and interior point [80] methods, solutions to various generalized network problems can provably be obtained in polynomial time as they can be modeled as linear programs. On the other hand, exploiting combinatorial structure of the problems tends to give better worst-case bounds on algorithm running times. To our best knowledge, the fastest known running time for a combinatorial approach to the generalized maximum flow problem is $(\tilde{O})(m^2 n \log B)$ due to Radzik [104] based on [103, 60]. Applying the fastest general purpose LP-algorithm combined with speed-up techniques for matrix inversion delivers a time bound of $(\tilde{O})(n^2 m^{1.5} \log nB)$ for solving the generalized circulation problem [55, 118, 79].

One of the more recent approaches by Oldham [96] exchanges a TVPI feasibility test for a combinatorial subroutine. The subroutine indicates if a chosen vertex potential is smaller or greater than the dual problem's optimal value by a Bellman-Ford [11] computation. Unfortunately, Restrepo and Williamson [105] show that Oldham's assumption of converging distance labels within n iterations is not correct via a counter-example. Yet, his definition of appropriate reduced cost and application of a shortest path algorithm as a combinatorial subroutine may retrospectively be seen as the original generalization of the successive shortest path algorithm to the minimum cost flow problem with multipliers, although Oldham only uses the shortest path computation as a relational operator and finds the generalized maximum flow via binary search.

Williamson and Restrepo also apply a path search algorithm, but detect a best generalized augmenting path with respect to a ratio between generalized reduced cost and residual capacity. The latter is important to guarantee the running time of $O(m^3 n \log(mB))$ as an exact and $O(m^2 n \log(mB/\epsilon))$ for an ϵ-approximate algorithm. Here, the actual and probably fractional optimal value is approximated, as opposed to our case, where we approximate an integral optimal solution (Chapter 8).

Regarding purely combinatorial algorithms, Wayne and Tardos [112] improved Truemper's augmenting paths algorithm [115] to polynomial running time. Further, they adopt Goldberg and Tarjan's preflow-push algorithm [54] for generalized maximum flow and also simplify the algorithm of [103]. More recently, Wayne proposed the first polynomial time

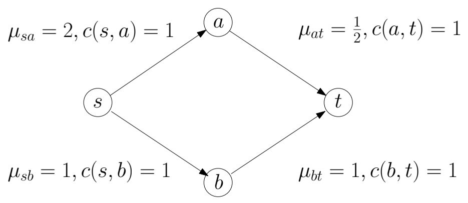

$\mu_{sa} = 2, c(s,a) = 1$
$\mu_{at} = \frac{1}{2}, c(a,t) = 1$
s
a
t
b
$\mu_{sb} = 1, c(s,b) = 1$
$\mu_{bt} = 1, c(b,t) = 1$

Figure 24: Two s-t-paths with identical arc cost cause different costs for sending one unit of flow from s to t.

combinatorial algorithm for the generalized minimum cost flow problem [119]. Previously, Fleischer and Wayne already developed fully polynomial time approximation schemes for generalized versions of maximum flow, minimum cost flow and multi-commodity flow problems [38]. For series-parallel graphs, even integral generalized maximum flows can be computed in pseudo-polynomial time [13].

7.4 MODIFIED SUCCESSIVE SHORTEST PATH ALGORITHM

In this section we present a modified version of the *Successive Shortest Path* algorithm with predefined source-sink-pairs (see Chapter 5) to solve generalized minimum cost flow instances N_I^g, which means for our purpose non-totalizable instances of the (DP). Note that we can also apply the modified *Successive Shortest Path* algorithm on total(izable) instances to avoid the transformation to a total instance with integral input values. In this case - as well as for any other instance with bounded fractionality - the modified *Successive Shortest Path* algorithm has pseudo-polynomial running time as the original algorithm. For the general case, without a fractionality bound on the flow solution, we cannot bound the running time of our modified algorithm, as arbitrary small fractions may be augmented.

To adopt the *Successive Shortest Path* algorithm for generalized flow networks, we define 'shortest paths' in the presence of multipliers. Figure 24 demonstrates why such a separate definition is necessary: The two possible paths from s to t result in costs of 2 accounting only on arc costs. Yet actually sending one unit of flow along $s-a-t$ creates two units of flow at a, which have to be passed on to t in order to avoid the creation of an imbalance at node a. This accounts to total cost of 3 for one incoming unit at t. On the other hand, sending one unit of flow along $s-b-t$ causes total cost of 2 per incoming unit at t. Let $N = (V, A)$ be a generalized network with cost function $c : A \to \mathbb{R}$ and multiplicator function $\mu : A \to \mathbb{R}$.

Definition 65 (Cost of Multiplier Paths) *The cost of a multiplier path (m-path) $\pi_{uv} = u_1 \dots u_n$ from $u = u_1$ to $v = u_n$ in N per unit of flow out of u is defined as:*

$$c'(\pi_{uv}) = \sum_{i=2}^{n} \prod_{j=1}^{i-1} \mu_{u_j u_{j+1}} c(u_{i-1}, u_i).$$

A shortest flow multiplier path π^*_{uv} in N is a path from u to v of minimum cost $c'(\pi_{uv})$.

We show that for this definition of path cost, similar to the canonical sum definition, the following Lemma holds:

Lemma 66 (Consistent Shortest m-Paths) *Every subpath $\pi_{u_i u_j}, 1 \leqslant i < j \leqslant n$ of a shortest m-path $\pi^*_{uv} = u = u_1 \dots v = u_n$ is a shortest m-path $\pi^*_{u_i u_j}$.*

Proof: Let $\pi^*_{uv} = u = u_1 \dots v = u_n, 1 \leqslant i < j \leqslant n$ be a shortest m-path and $\pi_{u_i u_j} \subseteq \pi^*_{uv}$ be a subpath, such that a shortest m-path from u_i to u_k with $c'(\pi^*_{u_i u_j}) < c'(\pi_{u_i u_j})$ exists. Then:

$$\begin{aligned} c'(\pi^*_{uv}) &= c'(u \dots u_{i-1}) + \prod_{l=1}^{i-1} \mu_{l(l+1)} \cdot c'(\pi_{u_i u_j}) \\ &\quad + \prod_{l=1}^{j} \mu_{l(l+1)} \cdot c'(u_j \dots v) \\ &> c'(u \dots u_{i-1}) + \prod_{l=1}^{i-1} \mu_{l(l+1)} \cdot c'(\pi^*_{u_i u_j}) \\ &\quad + \prod_{l=1}^{j} \mu_{l(l+1)} \cdot c'(u_j \dots v). \end{aligned}$$

This contradicts the assumption that π^*_{uv} is a shortest m-path. □

We adjust Dijkstra's algorithm to the above defined path cost introducing a node parameter m_v, which accounts for the path multiplier of the (tentative) shortest path from s to every node v (see Algorithm 12).

Algorithm 12 *Flow Multiplier Dijkstra (m-Dijkstra)*

```
Input: $G = (V, E), s \in V, c : E \to \mathbb{R}^{>0}, \mu : E \to \mathbb{R}^{>0}$
Output: $\forall v \in V : d_s(v) = |\pi^*_{sv}|$ under $c, \mu$
 1: $F = \{s\}, F^* = \emptyset, d_s(s) = 0, m_s = 1$
 2: $d_s(v) = \infty, m_v = 1, v \in V \setminus \{s\}$
 3: while $F \neq \emptyset$ do
 4:     $d_s(u) = \min_{v \in F}\{d_s(v)\}$
 5:     for all $v \in V \setminus F^* : (u, v) \in A$ do
 6:         $F = F \cup \{v\}$
 7:         if $d_s(v) > d_s(u) + m_u \cdot c(u, v)$ then
 8:             $d_s(v) = d_s(u) + m_u \cdot c(u, v)$
 9:             $m_v = m_u \cdot \mu_{uv}$
10:         end if
11:     end for
12:     $F = F \setminus \{u\}, F^* = F^* \cup \{u\}$
13: end while
14: return $d_s$
```

Lemma 67 (Correctness of m-Dijkstra) *Algorithm 12 maintains the following invariants (over the while loop, ll. 3–13):*

1. $\forall v \in F^* : d_s(v) = c'(\pi_{sv}), m_v = \prod_{(u,w) \in \pi_{sv}} \mu_{uw}$
2. $\forall v \in V \setminus F^* : d_s(v) = \min_{x \in F^*}\{c'(\pi_{sx}) + m_x \cdot c(x, v)\}$

Proof: After the initialization both invariants hold: $d_s(s) = c'(\pi_{ss}) = 0$ and $m_s = 1$. The first pass through the while loop updates all the distances of the source's neighbors to $d_s(v) = m_s c(s, v)$, such that the second invariant is valid. For each additional pass through the while loop, one vertex u enters F^*. Let $\pi^*_{su} = s \dots v_x x y v_y \dots u$ be a shortest m-path from s to u and x the last vertex on the path in F^* before the current pass through the while loop, such that $x \in F^*, y \in V \setminus F^*$. Then

$$\begin{aligned} c'(\pi^*_{su}) &= c'(\pi^*_{sx}) + m_x c(x, y) + m_y c'(u_j \dots v) \\ &\geqslant d_s(y) + m_y c'(u_j \dots v) \\ &\geqslant d_s(u). \end{aligned}$$

Assuming that $c'(\pi^*_{su})$ is a shortest path provides equality and thus the first invariant is guaranteed after each while loop. The second invariant is also valid, because for every vertex v which is not a neighbor of u, the 'arc cost' $c(u, v)$ is set to infinity. Thus $\min_{x \in F^*} c'(\pi_{sx}) + m_x c(x, v)$ is never decreased by $c'(\pi_{su}) + m_u \cdot c(u, v)$ and for all neighbors of u a possible decrease is checked explicitly. □

The modified Dijkstra algorithm terminates after at most n while-loops (each vertex is added to and removed from F exactly once) maintaining the invariant of Lemma 67. All nodes v with $\pi_{sv} \subseteq A$ are in F^* after termination such that shortest m-paths (and their costs) are determined correctly. Due to non-negativity of arc costs, we employed Dijkstra's algorithm in the *Successive Shortest Path* algorithm and the variant given by Algorithm 2. Replacing Dijkstra for the m-Dijkstra and taking the multipliers into account when computing the maximum flow δ to be augmented, we obtain a generalized variant of the *Successive Shortest Path* algorithm by Algorithm 13.

Algorithm 13 *Flow Multiplier SSP (m-SSP)*

```
Input: $G = (V, E)$, $c : E \to \mathbb{R}^{>0}$, $u : E \to \mathbb{R}^{>0}$, $b : V \to \mathbb{R}$, $\mu : E \to \mathbb{R}^{>0}$
Output: Minimum cost flow $f$ in $G$
 1: $\forall v \in V : p(v) = 0$
 2: $\forall a = (u, v) \in A : f(a) = 0, c_r(a) = c(a), u_r(a) = u(a)$
 3: $E = \{v, b(v) > 0\}, D = \{v, b(v) < 0\}$
 4: while $E \neq \emptyset$ do
 5:     Determine shortest m-path $\pi^*_{ev}$ and $d(e, d)$ for $e \in E, d \in D$ with
        respect to reduced costs $c_r$ in the residual network $G(f)$
 6:     Update $p(v) = p(v) - d(e, v) + d(e, d)$ for each permanently labeled
        $v \in V$
 7:     $\delta := \min\{b(e), -b(d), \min_{a=(uv)\in\pi_{ed}} \frac{u_r(a)}{m_u}\}$
 8:     Augment $\delta$ units of flow along $\pi^*_{ed}$
 9:     $\forall a = (u, v) \in A :$
10:     $f(a), c_r(a) = c(a) - p(u) + p(v), u_r(a) = u(a) - f(a)$
11:     Update $G(f), E, D$
12: end while
13: return $f$
```

To prove the optimality of a flow computed by Algorithm 13, we adapt two lemmas from [2] (Lemma 9.11, 9.12):

Lemma 68 *Let f be a pseudo-flow which satisfies the reduced cost-optimality conditions with respect to some node potentials $p(v)\forall v \in V$, $c_r(a) = c_r(u, v) = m_u c(u, v) - p(u) + p(v)$ and let $d_s(v)$ be the cost of $\pi^*_{sv} \forall v \in V$ in the residual network $G(f)$. Then:*

1. *The pseudo-flow f also satisfies the reduced cost-optimality conditions with respect to the node potentials $p(v)' = p(v) - d_s(v)$.*
2. *The reduced costs $c'_r(a) = c'_r(u, v) = m_u c(u, v) - p'(u) + p'(v)$ are zero for all arcs $a \in \pi^*_{sv}$.*

Proof: We show both properties.

1. According to the assumption, reduced costs are non-negative on all arcs (u, v) and shortest path optimality $(d_s(v) \leqslant d_s(u) + c_r(uv))$ holds for shortest m-paths by Lemma 66 and 67, hence:

$$\begin{aligned} & d_s(v) \leqslant d_s(u) + m_u c(u,v) - p(u) + p(v) \\ \Leftrightarrow\ & 0 \leqslant m_u c(u,v) - (p(u) - d_s(u)) + (p(v) - d_s(v)) \\ \Leftrightarrow\ & 0 \leqslant m_u c(u,v) - p(u)' + p(v)' \\ \Leftrightarrow\ & 0 \leqslant c'_r(u,v) = c'_r(a) \end{aligned}$$

2. For any arc (u, v) on a shortest m-path:

$$\begin{aligned} & d_s(v) = d_s(u) + c_r(u,v) \\ \Leftrightarrow\ & d_s(v) = d_s(u) + m_u c(u,v) - p(u) + p(v) \\ \Leftrightarrow\ & 0 = m_u c(u,v) - (p(u) - d_s(u)) + (p(v) - d_s(v)) \\ \Leftrightarrow\ & 0 = m_u c(u,v) - p(u)' + p(v)' \\ \Leftrightarrow\ & 0 = c'_r(u,v) = c'_r(a). \end{aligned}$$

□

Lemma 69 *Let f be a (pseudo-)flow which satisfies the reduced cost-optimality condition. We obtain f' from f by sending flow along a shortest m-path from node s to some node t. Then f' also satisfies the reduced cost-optimality conditions.*

Proof: With potentials p and p' and reduced costs as defined in Lemma 68, f' also satisfies the reduced cost-optimality conditions with $c'_r(a)$. Further, for any arc $a \in \pi^*_{sv}$, a reverse arc $\bar{a}$ is added to $G(f')$, but as $c'_r(a) = 0$ for such a, $c'_r(\bar{a}) = -c'_r(a) = 0 \geqslant 0$. □

Lemmas 68 and 69 show that Algorithm 13 terminates with a minimum cost flow, after a number of augmentations. In the classical minimum cost flow computation, the number of augmentations is pseudo-polynomially bounded by the maximum of total supply and demand, as the minimum δ is at least 1 in each iteration. For instances whose solutions are known to be of bounded fractionality β, the number of iterations increases by the factor β in the worst case. However, the running time stays pseudo-polynomial.

8 APPROXIMATIONS FOR APPLICATION INSTANCES

As shown in Chapter 4 obtaining a feasible distribution for practical (DP) instances is NP-complete. Therefore, we have to find a compromise between running time and solution quality in practice. In this chapter, we obtain integral assignments by rounding a given half-integral minimum cost flow f in N_I^g, which can be computed in polynomial time (see Chapter 7).

The first approach only rounds strictly half-integral flows and results in a pseudo-flow with the same cost $c(f)$ as the half-integral flow f. The derived distribution is not always formally feasible, hence no approximation in the classical sense. Yet, the constraint violations are small and distinct and may be acceptable in practice as they can be seen as upgrades on single cars of a number of demands. We formally show that the result is an optimal 0.5-upgraded distribution (Definition 32) for instances from the application.

In a second approach, we extend rounding (or change) of flow to arcs a with integral flow $f(a)$. With two additional assumptions on the instances, we obtain a classical 4-approximation to the (DP).

Both ideas are combined in practice to an iterative heuristic. The resulting distribution is feasible for application instances without presumptions, but we cannot guarantee a constant approximation factor with respect to the cost. We study the performance of the heuristic on application data in Appendix A.1.

In the following Section 8.1 we consider the associated network N_I^g of a totalizable 2-heterogeneous instance I and a given half-integral minimum cost flow f in N_I^g. We then investigate properties of strictly half-integral flows $f(a)$ on transit arcs a. In Section 8.2 we partition such transit arcs in paths and/or cycles, which are the basis for the upgraded distribution presented in Section 8.3. In Section 8.4 some paths are extended to cycles before rounding, which results in a 4-approximation to the heterogeneous (DP).

8.1 DEFINITIONS AND IDEA

Let $I = \{S, D, O, B, \mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$ be a totalizable 2-heterogeneous instance of the (DP) from the application (see Section 7.2.3). With respect to the approximation analysis, we assume that all supplies $s_i \in S$ are of type t_x or t_y and all demands $d_j \in D$ are of type t_x, t_y or t_z. Type t_z is an artificial car type to allow demands for cars of type t_x or t_y without specifying the exact type (see Section 2.3). Then assuming $\mathcal{S} = \mathcal{S}|xyz = \{(t_x, t_x), (t_y, t_y), (2t_x, t_z), (t_y, t_z)\}$

poses no additional restriction. We also assume for the moment that no supplies have foreign wagon keepers, all demands share the same strong priority, have domestic destinations (of the loaded run) and $B = \mathcal{B} = \emptyset$ to simplify the analysis. Further, the car types t_x, t_y and t_z are allowed for all operative storage $o \in O$.

Let $N_I^g = (V, A)$ be the associated generalized network to I as in Definition 51. Further, let $V_S \subset V$ ($V_D \subset V$) denote the set of supply (demand) nodes and $V_x, V_y, V_z \subseteq V$ denote the sets of supply or demand nodes with associated type t_x, t_y and t_z respectively. (Note that $V_S \cap V_z = \emptyset$.) Let $T = A_T$ denote the set of transit arcs and T_{all} denote the set of possible transit arcs, if each supply allowed for a demand is also in time for that demand.

We assume f to be a half-integral minimum cost flow in N_I^g. Such a flow can be computed in polynomial time as described in Chapters 5 and 7. For the further analysis we assume a preliminary maximum flow computation and an adjustment of the balance at the sink t (which is unique due to the single level of strong priority occurring in I) has been carried out, such that f is a feasible flow with $|f| = -b(t) = \sum_{a=(i,j)\in T} \nu(t_i, t_j) f(a)$. The basic idea of our algorithms is to round the strictly half-integral flows of f to a feasible integral flow from which we derive a distribution with certain properties.

Definition 70 (Subflow f') *For all $a = (i, j) \in T$ let $f'(a)$ be defined as follows (with $n \in \mathbb{N}$):*

$$f'(a) = \begin{cases} f(a) - n, \ \textit{for}\ f(a) = \frac{2n+1}{2} \\ f(a) - n, \ \textit{for}\ f(a) = n, i \in V_y \\ f(a) - n, \ \textit{for}\ f(a) = n, i \in V_x, j \in V_x \\ f(a) - 2n, \ \textit{for}\ f(a) \in \{2n, 2n+1\}, i \in V_x, j \in V_z \end{cases}$$

For $a = (s, i) \in A \setminus T$ let $f'(a) = \sum_{a'=(i,j)\in T} f'(a')$ and for $a = (j, t) \in A \setminus T$ let $f'(a) = \sum_{a'=(i,j)\in T} \mu(a') f'(a')$.

Observe that $f'(a) = \frac{1}{2}$ only possibly occurs on arcs $a = (i, j)$ with $i \in V_y$ and $j \in V_y \cup V_z$ by the construction we applied to prove bounded fractionality $\beta = 2$ for application instances. In words, flow f' reduces the flow on arcs with strictly half-integral flow f to one half and sets flow on almost all other transit arcs to zero. An exception are the arcs $a = (i, j)$ with $t_i = t_x$ and $t_j = t_z$. If $f(a)$ is odd, then $f'(a)$ is reduced to one and this unit flow represents a car which will be rerouted later in the approximation approach to obtain feasibility.

Our further considerations mainly concern such arcs $a \in T$ (with strictly half-integral flow $f(a)$), which lead to infeasible assignments in a derived distribution $\mathcal{D}^f$. We extend f' to arcs $a \in A \setminus T$ to argue feasibility (or limit infeasibility) of the rounded flow. The following Definition 71 groups arcs and nodes due to their associated flows f' and those of their incident arcs respectively.

Definition 71 (Arc and Node Sets) *Let the arc sets* $T^1, T^2, \tilde{T}^1, \tilde{T}^2$ *and the node sets* V^1, V^2 *be defined as follows:*

- $T^1 = \{a \in A | f'(a) > 0\}$
- $T^2 = \{a \in T^1 | f'(a) < 1\}$
- $\tilde{T}^1 = \{\tilde{a} = (v, u) | a = (u, v) \in T^1\}$
- $\tilde{T}^2 = \{\tilde{a} = (v, u) | a = (u, v) \in T^2\}$
- $V^1 = \{u, v \in V | (u, v) \in T^1\}$
- $V^2 = \{u, v \in V | (u, v) \in T^2\}$

Rounding f' in a naive way can easily cause a violation of capacity bounds on the arcs (s, i) and (j, t) and/or violate flow conservation, especially if we round greedily on behalf of costs. As an example, consider the following rounding heuristic in Algorithm 14 which temporarily allows arc capacities $u(j, t)$ to be violated by 1. The heuristic can always be applied to a half-integral solution until there are only integral flows, because in each iteration, at least two half-integral flows are rounded to an integral pseudo-flow. Hence, the running time is bounded in the worst case by $O(m_t) \in O(m)$ with m_t the number of transit arcs with strictly half-integral flow and $m = |A|$.

The arc capacities on arcs (j, t) may remain violated by at most 1. In terms of distribution, this situation corresponds to sending one more freight car than demanded by d_j to a customer, which means d_j is oversatisfied. Moreover, flow conservation can be violated when excess $b'(i) > 0$ is created at a supply node i. A derived distribution is then infeasible, as not all supply at s_i is assigned. This can happen if half-integral flows on transit arcs (i, j), which are rounded down, have no counterpart (i, j') to be rounded up. Note that in this case the number of outgoing arcs of i with strictly half-integral flow would not be even. On the other hand Algorithm 14 can simply fail to round properly, in case $b'(i)$ is increased. We see that the latter is true. Let $d_2(v) = |\{a = (u, v) | a \in T^2 \cup \tilde{T}^2\}|, v \in V^2$ be the *degree* of v with respect to T^2.

Lemma 72 (Even Supply Degree) *The degree* $d_2(i)$ *is even for all* $i \in V_S \cap V^2$.

Proof: For all $i \in V_S$ is $f(s, i) = \sum_{a=(i,j) \in T} f(a) = n_i$ as f is a feasible flow in N_I^g. Since n_i is integral the number of strictly half-integral flows $f(i, j)$ must be even. Further, with Definitions 70 and 71, each such arc (i, j) is in T^2, hence $d_2(i)$ is even. □

Lemma 72 shows that Algorithm 14 fails to round properly when creating excess $b'(i) > 0$ at supply nodes i. In the following we develop a rounding procedure which rounds locally feasible with respect to flow conservation instead of greedily minimizing cost. At each supply node in V^2, as many strictly half-integral flows on incident arcs are rounded up as are rounded down. At each demand node the same is done, if possible.

Algorithm 14 *Rounding Heuristic*

```
Input: $V_2, T_2, f$
Output: Integral $f$
 1: $F = T^2, \forall i \in V_S : b'(i) = 0$
 2: while $F \neq \emptyset$ do
 3:     Choose $a = (i,j) \in F$ with $c(a) = \min_{a' \in F} c(a')$
 4:     if $\sum_{i'} \mu(i',j) f(i',j) + \mu(i,j)(f(i,j) + \frac{1}{2}) \leqslant u(j,t) + 1$ then
 5:         Round $f(a)$ up, $F = F \setminus \{a\}$
 6:         if $\exists a = (i,j') \in F$ then
 7:             Choose $a = (i,j') \in F$ with $c(a) = \max_{a' \in F} c(a')$
 8:             Round $f(a)$ down, $F = F \setminus \{a\}$
 9:         end if
10:     else
11:         Round $f(a)$ down, $F = F \setminus \{a\}$
12:         if $\exists a = (i,j') \in F$ then
13:             Choose $a = (i,j') \in F$ with $c(a) = \min_{a' \in F} c(a')$
14:             Round $f(a)$ up, $F = F \setminus \{a\}$
15:         else
16:             $b'(i) = b'(i) + 1$
17:         end if
18:     end if
19: end while
20: Adjust $f(j,t)$ according to flow conservation constraints.
```

If we interpret 'round up flow on arc (i,j)' as 'enter node j' and 'round down flow on arc (i',j)' as 'leave node j', then we can round globally feasible, if all arcs $a \in T^2$ compose a cycle (disregarding arc directions) for each component in N_I^g. In other words, we can round globally feasible if the undirected, bipartite graph $G^2 = (V^2, \{(u,v) | (u,v) \in T^2 \cup \tilde{T}^2\})$ is Eulerian [35]. Graph G^2 being Eulerian would imply that and we could show an analogous result as in Lemma 72 for demand nodes $j \in V_D \cap V^2$ as well.

This is not true for every instance of the (DP), as the following example shows. Consider I with unit supplies $S = \{s_1, s_2, s_3\}$ and unit demands $D = \{d_4, d_5\}$ of type t_z, such that $s_1, t_1 = t_x$ matches d_4, $s_2, t_2 = t_y$ matches d_4, d_5 and $s_3, t_3 = t_x$ matches d_5. Then Figure 25 shows the corresponding

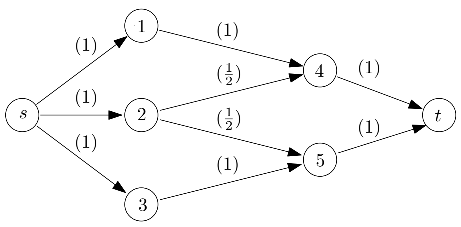


Figure 25: Network N_I^g for a small application instance with single feasible flow (numbers in brackets): For all $j \in V_D \cap V^2$ the degree $d_2(j)$ with respect to T^2 is odd.

Network N_I^g and the only feasible flow (numbers in brackets), which is strictly half-integral on arcs $(2,4)$ and $(2,5)$. Hence $T^2 = \{(2,4),(2,5)\}$, $V^2 = \{2,4,5\}$ and $d_2(j)$ is odd for each $j \in V_D \cap V^2$.

8.2 COVERING ARCS BY CYCLES AND PATHS

Algorithm 15 *Cover Construction*

```
Input: $T^2, \tilde{T}^2, V^2, V_{odd}$
Output: CO covers $T^2$
 1: $CO = \emptyset$
 2: while $V_{odd} \neq \emptyset$ do
 3:    Chose $j \in V_{odd}$.
 4:    $p = \emptyset, v = j$
 5:    repeat
 6:       Chose $a = (u,v), u \in V_S \cap V^2, a \in T^2$
 7:       $p = p \cup \{\tilde{a}\}$,
 8:       $T^2 \setminus \{a\}, \tilde{T}^2 \setminus \{\tilde{a}\}$
 9:       Chose $a = (u,w), w \in V_D \cap V^2, a \in T^2$
10:       $p = p \cup \{a\}$,
11:       $T^2 \setminus \{a\}, \tilde{T}^2 \setminus \{\tilde{a}\}$
12:       $v = w$
13:    until $w \in V_{odd}$
14:    $CO = CO \cup p$
15:    if $j \neq w$ then
16:       $V_{odd} = V_{odd} \setminus \{j,w\}$
17:    end if
18: end while
19: Cover the remaining arcs $a \in T^2$ by cycles $c \in C$.
20: $CO = CO \cup C$.
```

We saw in the previous section, that $d_2(j)$ may be odd for demand nodes $j \in V_D \cap V^2$. Let $V_{odd} = \{j \in V_D | d_2(v) = 2n+1, n \in \mathbb{N}\}$. Further, let B be a set of arcs, $\tilde{B} = \{\tilde{a} = (v,u) | a = (u,v) \in B\}$, $C \subseteq B \cup \tilde{B}$. Then C is said to be *covering* an arc a and arc $a = (u,v) \in C$ is said to be *covered* by C, if *either* $a \in C$ *or* $\tilde{a} \in C$. The set C is a *cover* of B. By definition T^2 is a (special) T-join in N_I^g viewed undirected for terminal set $T = V_{odd}$ and for $V_{odd} = \emptyset$ the T-join coincides with a cover by Eulerian cycles. On the other hand, not each T-join on $T = V_{odd}$ provides a cover for T^2. By Algorithm 15 we obtain such a cover of T^2 consisting of cycles and paths between nodes of V_{odd}, as we show in Lemma 73.

Lemma 73 (Cover of T^2) *The set* CO *as determined by Algorithm 15 covers* T^2 *by* $\frac{1}{2}|V_{odd}|$ *paths and a number of cycles in* N^2.

Proof: A single path $p(j,j')$ (or cycle through $j = j'$) can obviously be constructed by Algorithm 15: The algorithm chooses $v \in V_{odd}$ arbitrarily and subsequently enlarges p by arcs from $T^2 \cup \tilde{T}^2$. Arc $a = (u,v), u \in V_S, a \in T^2$ exists because $v \in V_{odd}$. Then there is also $\tilde{a} \in \tilde{T}^2$, which is included in p (and deleted from $\tilde{T}^2$ as well as a from T^2). By Lemma 72, we know that $d_2(u)$ is even and $d_2(u) > 0$ because of a. Consequently there is $a' = (u,w) \in T^2$. Again a' is included in $p(j,j')$ (and deleted from T^2 as well as $\tilde{a}'$ from $\tilde{T}^2$). For $w \in V_D \cap V^2$ we distinguish two cases. Let $w \in V_{odd}$. Then we found $p = p(v = j, j' = w)$. Let $w \notin V_{odd}$. Then $d_2(w)$ is even and we can proceed enlarging p as done by Algorithm 15 with $v = w$. (Note that during further iterations, we can also encounter the case that $p = c$, a cycle through $j = w$.)

Apart from j and j', after the construction of $p(j,j')$ the values $d_2(v)$ of all vertices in $v \in p$ are reduced by exactly 2. The values $d_2(j)$ and $d_2(j')$ with respect to the remaining arc set is reduced by 1 in case of a path. Thus their degree is now also even and they are removed from V_{odd}. In case of a cycle, the degree of $v = v'$ stays odd and the node is not deleted from V_{odd}. Hence all parity arguments hold for the creation of a subsequent set p'.

Further, either $|V_{odd}|$ is reduced by 2 vertices (path case), or at least $d_2(v)$ of a vertex $v = v' \in V_{odd}$ is reduced by 2 (cycle case). Iterating the latter case leaves all nodes in $v \in V_{odd}$ with $d_2(v) = 1$ and no cycle containing v can occur any more during path construction. The while-loop terminates with $V_{odd} = \emptyset$ after construction of $\frac{1}{2}|V_{odd}|$ paths, as each path has exactly start and end point in V_{odd}. Then $d_2(v)$ is even in the remainder of the arc set T^2 for all vertices $v \in V^2$. Hence, the remaining graph N^2 (or each of its components) is Eulerian and can be covered (each) by a cycle efficiently.

As for each arc a ($\tilde{a}$), which is included in a path or cycle, both arcs a and $\tilde{a}$ are removed from the arc set, the union of all path and cycles CO covers T^2 and each vertex $v \in V^2$ lies on one or more paths and/or cycles in CO. □

Algorithm 15 picks each arc from T^2 or its reverse $\tilde{a} \in \tilde{T}^2$ once during path construction or in the cycle cover. Choosing the appropriate arc can be done in constant time with adjacency lists (the first arc in the list always suffices). The administration of vertex and arc sets can also be done in constant time. Hence, Algorithm 15 needs running time $O(m_t) \in O(m)$.

Consider the paths $p = p(v, v') \subseteq CO, v, v' \in V_{odd}$ and cycles $c \subseteq CO$. By construction they alternately contain arcs $a \in T^2$ and $\tilde{a} \in \tilde{T}^2$. Our strategy will be to round flow up on arcs $a \in c, p$ and down on arcs for which $\tilde{a} \in p, c$ or vice versa. Since N^2 is bipartite, paths $p(v, v')$ and cycles constructed by Algorithm 15 are of even length. Thereby we round the same amount of flow up and down on each path or cycle. Especially for each node of a cycle and each inner node of a path, the same amount of flow on incident arcs is rounded up and down. This will be an important argument throughout this chapter:

Corollary 74 (Balanced Cover of T^2) *All paths $p = p(v, v') \subseteq CO, v, v' \in V_{odd}$ and cycles $c \subseteq CO$ are of even length and alternately consist of arcs $a \in T^2$ and $\tilde{a} \in \tilde{T}^2$, such that: $|T^2 \cap p| = |\tilde{T}^2 \cap p|$ and $|T^2 \cap c| = |\tilde{T}^2 \cap c|$ respectively.*

8.3 ROUNDING THE CYCLE- AND PATH-COVER

Algorithm 16 *Rounding the Cycle- and Path-Cover*

```
Input: $f, f', T^2, \tilde{T}^2, V^2, V_{odd}, CO$
Output: Integral $f^* = f - f' + f'' \in \mathbb{N}$
 1: for all $a \in T^2$ do
 2:    if $a \in CO$ then
 3:       $f''(a) = 1$
 4:    end if
 5:    if $\tilde{a} \in CO$ then
 6:       $f''(a) = 0$
 7:    end if
 8: end for
 9: for all $a \in T \setminus T^2$ do
10:    $f''(a) = f'(a)$
11: end for
12: for all $a = (s, u) \in A$ do
13:    $f''(a) = \sum_{(i,j) \in T} f''(a)$
14: end for
15: for all $a = (v, t) \in A$ do
16:    $f''(a) = \sum_{(i,j) \in T} v(t_i, t_j) f''(a)$
17: end for
```

Given CO as determined by Algorithm 15, we round the strictly half-integral flows $f(a)$ on transit arcs $a \in T$ as follows. Let f'' be as determined by Algorithm 16. Then we round f to f^* by setting $f^*(a) = f(a) - f'(a) + f''(a)$ for all $a \in A$. Note that f^* remains equal to f on all transit arcs $a \in T \setminus T^2$. Further, f^* is integral on all transit arcs as intended, which we show in Lemma 75. Possibly f^* is not a feasible flow in N_I^g and consequently a derived distribution $\mathcal{D}^*$ is not guaranteed to be feasible even if it is integral. We therefore investigate the properties of f^* as a pseudo-flow with respect to flow conservation and capacity constraints by Lemmas 76 and 77. In Theorem 81 we conclude that we can derive a 0.5-upgraded optimal distribution with respect to I from f^*.

Lemma 75 (Integrality) *The rounded flow f^* is integral for all $a \in T$.*

Proof: For each arc with strictly half-integral flow $a \in T^2$, $f'(a) = \frac{1}{2}$ and for all other arcs $f'(a) \in \{0, 1\}$, such that $f(a) - f'(a) \in \mathbb{N}$ for all arcs $a \in T$. Further Algorithm 15 chooses $f''(a)$ from $\{0, 1\}$ for each $a \in T^2$, as CO covers T^2. For all other arcs $f''(a) = f'(a) \in \mathbb{N}$ is set. Hence, $f^*(a)$ is integral for all arcs $a \in T$. □

We speak of 'rounding flow up (down) on $a \in T^2$', if $f^*(a) = f(a) - \frac{1}{2} + 1$ ($f^*(a) = f(a) - \frac{1}{2} + 0$). By Lemma 75, all assignments of a derived distribution from f^* are integral, as we intended. Feasibility of $\mathcal{D}^*$ also requires f^* to be feasible. Firstly, we show that f^* preserves flow conservation for all nodes.

Lemma 76 (Flow Conservation) *The rounded flow f^* preserves flow conservation for all $v \in V$.*

Proof: The flow f is feasible in N_I^g and thus preserves flow conservation. Let $v = i \in V_S$ ($v = j \in V_D$). Then (s, i) ((j, t)) is the only incoming (outgoing) arc of i (j). By Definition 70 $f'(s, i)$ ($f'(j, t)$) is the (weighted) sum of $f'(i, j)$ on all outgoing (incoming) arcs of i (j). Algorithm 16 sets $f''(s, i)$ and $f''(j, t)$ analogously. Thus f^* preserves flow conservation for all nodes $v \in V_S \cup V_D$, as $b(v) = 0$.

Let $v = s$. Then v has no incoming arcs and the outgoing arcs are (s, i) with $f^*(s, i) = \sum_{(i,j) \in A} f^*(i, j)$, as seen before. Further, each $i \in V_S$ only occurs on cycles $c \in CO$ or as an inner node on paths $p \in CO$. By Corollary 74 as many flows on incident arcs (i, j) are rounded up as are rounded down by Algorithm 16 and thus $\sum_{(i,j) \in A} f^*(i, j) = \sum_{(i,j) \in A} f(i, j)$. Consequently $f^*(s, i) = f(s, i)$ for each $(s, i) \in A$ and since f is feasible, $\sum_{(i,j) \in A} f(i, j) = b(s)$.

Let $v = t$. Then v has no outgoing arcs and the incoming arcs (j, t) with $f^*(j, t) = \sum_{(i,j) \in A} \mu(i, j) f^*(i, j)$, as seen before. Each $j \in V_D \setminus V_{odd}$

only occurs on cycles $c \in CO$ or as an inner node on paths $p \in CO$. By Corollary 74 as many flows on incident arcs (i,j) are rounded up as are rounded down by Algorithm 16. Moreover, $(i,j),(i,j') \in T^2$ and all arcs in T^2 start in $V_S \cap V_y$ and end in $V_D \cap (V_y \cup V_z)$, such that $\mu(i,j) = \mu(i,j')$ given the relative valencies. Hence, $\sum_{(i,j)\in A} \nu(t_i,t_j)f^*(i,j) = \sum_{(i,j)\in A} \nu(t_i,t_j)f(i,j)$. Each $j \in V_{odd}$ occurs as an endpoint of some path $p(j,j') \in CO$. Thus there is an arc (i,j) with $f^*(i,j) = f(i,j) - \frac{1}{2}$ resulting in $f^*(j,t) = f(j,t) - \frac{1}{2}$ for each arc (i,j') with $f^*(i,j') = f(i,j') + \frac{1}{2}$ resulting in $f^*(j,t) = f(j,t) + \frac{1}{2}$ or vice versa. Again with $\mu(i,j) = \mu(i,j')$ it holds $\sum_{(j,t)\in A} f^*(j,t) = \sum_{(j,t)\in A} f(j,t)$ and since f is feasible, $\sum_{(j,t)\in A} f(j,t) = -b(t)$. □

However, f^* may violate capacity constraints in the following restricted way.

Lemma 77 (Bounded Capacity Violation) *For all* $a \in A : f^*(a) \leqslant u(a) + \frac{1}{2}$.

Proof: The flow is actually rounded on arcs $T^2 \subseteq T$ with $u(a) = \infty$, such that no direct capacity violation occurs. The extension of f^* to arcs $(s,i),(j,t) \in A \setminus T$ is determined by the flow conservation constraints for the incident supply and demand nodes i,j. By Corollary 74, all nodes $v \neq s \neq t \in V \setminus V_{odd}$ are incident to the same number of arcs $a \in CO$ and $\tilde{a} \in CO$, such that the same amount of flow out of v (for $v \in V_S$) or into v (for $v \in V_D$) is rounded up and down respectively. Consequently flows on the arcs (s,v) or (v,t) respectively are not changed and no capacity violation occurs. (Note that strictly half-integral and therefore rounded flows only occur on arcs (i,j), with $t_i = t_y$ and $t_j \in \{t_y,t_z\}$. Therefore $\mu(i,j) = \nu(t_y,t_y) = \nu(t_y,t_z)$ and the latter also holds for arcs (v,t).)

For each node $j \in V_{odd} \subseteq V_D$, exactly one path $p \in CO$ starts or ends in j with edge $e = (i,j)$ or $\tilde{e} = (j,i)$. If $f(e)$ is rounded down, no capacity violation occurs. Otherwise, if already $f(j,t) = u(j,t)$ and $f(e)$ is rounded up to $f^*(e) = f(e) + \frac{1}{2}$, then the capacity $u(j,t)$ is violated. (Note that $f^*(e)$ always stays feasible if $f(j,t) < u(j,t)$ due to integrality of u and half-integrality of f.) As $e \in T^2$, $f(e)$ was strictly half-integral and $i \in V_y$. Hence with $\nu(t_y) = 1$, $f^*(j,t) \leqslant u(j,t) + \frac{1}{2}$. □

By Lemma 77, the flow f^* generally keeps the capacity constraints, apart from some arcs (j,t) on which the capacity is violated by exactly $\frac{1}{2}$. We can further specify such nodes $j \in V_D$ with capacity violation on arc (j,t):

Lemma 78 (Property of oversatisfied Demands) $f^*(j,t) = u(j,t) + \frac{1}{2} \Rightarrow j \in V_{odd} \cap V_z$

Proof: Let j be a demand node with $f^*(j,t) = u(j,t) + \frac{1}{2}$. Since $j \in V_{odd}$, j has an odd number of incident arcs (i,j) with strictly half-integral flow $f(i,j)$.

As f is feasible $f(j,t) = \sum_{(i,j)\in T} \mu(i,j)f(i,j)$ holds and $f(j,t) = u(j,t) \in \mathbb{N}$ by Lemma 77. Consequently there must be an arc (i',j) with $f(i',j) = 2k+1, k \in \mathbb{N}$ and $\mu(i',j) = \frac{1}{2}$. Given $\mu(i,j) = \nu(t_i,t_j)$ and the valencies $\nu(t_y) = \nu(t_y,t_y) = \nu(t_y,t_z) = 1$, $\nu(t_x,t_z) = \frac{1}{2}$ it must hold $i' \in V_x$. Since half-integral flow only occurs on arcs incident to $j \in V_y \cup V_z$ and by network construction $j \in V_z$. □

From Lemma 76 and 77, we see that f^* is a pseudo-flow only with respect to capacity violation on $\frac{1}{2}|V_{odd}|$ arcs $(j,t) \in A$ by $\frac{1}{2}$. Thus $\mathcal{D}^*$ is strictly speaking not yet feasible.

The cost $c(f^*)$ are at most twice the cost of f due to the half-integrality of f. In fact, we can do better in terms of costs: If we chose the arcs which are rounded up or down in Algorithm 16 more carefully, we can show that the cost for the rounded solution do not increase with respect to the given half-integral solution. So far, a flow $f(a)$ is rounded up if $a \in p$ or $a \in c$ for some path $p \subseteq CO$ or cycle $c \subseteq CO$. An arc is rounded down if the same is true for $\tilde{a}$. Note that Corollary 74 also holds if the roles of a and $\tilde{a}$ are exchanged. Hence, we can specify to round $f(a)$ up either if $a \in p$ or if $\tilde{a} \in p$ individually for each path and cycle in CO. We abbreviate this as the 'choice of *rounding direction*' as $d(p) = a$ ($d(c) = a$) or $d(p) = \tilde{a}$ ($d(c) = \tilde{a}$).

Let $c^a(p)$ denote the cost of the rounded flow with respect to arcs covered by p if its rounding direction is $d(p) = a$ and $\tilde{c}^a(p)$ if its rounding direction is $d(p) = \tilde{a}$. Then as a subroutine in Algorithm 16, we compute $c^a(p)$ and $\tilde{c}^a(p)$ for each $p \subseteq CO$. We individually determine the rounding direction such that the resulting cost $c^*(p)$ for each path p is the minimum of both. Cycles are treated analogously. Note that the running time of Algorithm 16 is still linear in m. Let $f^*(a)$ resulting from $f''(a)$ be determined according to the chosen rounding direction of p.

Lemma 79 (Path Cost) *The costs $c^*(p)$ of p with respect to the rounded flow f^* do not exceed the cost $c(p) = \sum_{a\in p} c(a)f(a) + \sum_{\tilde{a}\in p} c(a)f(a)$ of p with respect to the original flow f.*

Proof: The cost $c^*(p)$ can be rewritten as

$$\begin{aligned} c^*(p) &= c(p) \\ &- \sum_{\tilde{a}\in p} c(a)f'(a) + \sum_{a\in p} c(a)f'(a) \\ &+ \sum_{\tilde{a}\in p} c(a)f''(a) + \sum_{a\in p} c(a)f''(a) \end{aligned}$$

Thus is suffices to show that $f'(a)$ causes more cost reduction than $f''(a)$ causes cost increase. As $a \in p$ or $\tilde{a} \in p$ imply $a \in T^2$ by definition $f'(a) = \frac{1}{2}$ holds. Further by (modified) Algorithm 16 either

- $\sum_{\tilde{a}\in p} c(a)f''(a) = 0$ and
- $\sum_{a\in p} c(a)f''(a) = 2\sum_{a\in p} c(a)f(a)' = \sum_{a\in p} c(a)$

or vice versa. W.l.o.g. we consider the former case, then it follows directly from the choice of rounding direction that:

$$\sum_{\tilde{a}\in p} c(a)f'(a) + \sum_{a\in p} c(a)f'(a) = \frac{1}{2}\sum_{\tilde{a}\in p} c(a) + \frac{1}{2}\sum_{a\in p} c(a) \geqslant \sum_{a\in p} c(a) = \sum_{\tilde{a}\in p} c(a)f''(a) + \sum_{a\in p} c(a)f''(a).$$

□

Lemma 79 also holds for cycles $c \in CO$. As f and f^* only differ on arcs T^2, which are partitioned into the paths and cycles of CO, it follows:

Corollary 80 (General Cost) $c(f^*) \leqslant c(f)$.

So far, we investigated properties of f^*. Besides the derived distribution $\mathcal{D}^*$, let $\mathcal{D}$ be the derived distribution of $f(N_i^g)$.

Theorem 81 (Optimal 0.5-upgraded Distribution) *The distribution $\mathcal{D}^*$ is an optimal 0.5-upgraded distribution for I with at most $0.5|V_{odd}|$ demands oversatisfied by 0.5 cars.*

Proof: By construction of N_I^g, for each $\delta^*_{ij} \in \mathcal{D}^*$ holds $s_i \in S^* = S$, $d_j \in D^* = D$ and s_i matches d_j. By Lemma 75 it holds $n^*_{ij} = f^*(i,j) \in \mathbb{N}$. By Lemma 76, also $n^*_{ij} \leqslant n_i$. Further, by Lemma 77, it holds $n^*_{ij} \leqslant n_j$, as $n^*_{ij} \in \mathbb{N}$. Hence δ^*_{ij} is feasible.

Further, by Lemma 76, we obtain $\sum_{\delta^*_{ij}} n_{ij} = n_i$. By Lemma 77, the flow f^* violates the capacity of at most $0.5|V_{odd}|$ arcs (j,t) by 0.5, such that $\mathcal{D}^*$ oversatisfies at most $0.5|V_{odd}|$ demands d_j by 0.5 cars. By Lemma 78 each oversatisfied demand d_j has type t_z and we can downgrade a single car of type t_y to type t_x. Given $\nu(t_y,t_z) = 1$ and $\nu(t_x,t_z) = \frac{1}{2}$, $\mathcal{D}^*$ is feasible for I after the appropriate downgrades.

Moreover $c(\mathcal{D}^*) \leqslant c(\mathcal{D})$ by Lemma 79 and $\mathcal{D}^*$ is an optimal 0.5-upgraded distribution for I. □

8.4 EXTENDING PATHS TO CYCLES

In the previous sections we saw that rounding strictly half-integral flows alternately up and down along cycles is preferable to doing so along paths. Yet, we cannot cover T^2 with cycles only, if $d_2(v)$ is not even for all $v \in V^2$.

We so far only considered changes to the flow on arcs in T^2, which is not obligate. In this section we want to extend paths $p \in CO$ as computed by Algorithm 16 to cycles using arcs from $T^1 \cup \tilde{T}^1$ and eventually $T \setminus (T^1 \cup T^2)$. The flow $f'(a)$ on arcs $a \in T^1$ is either $\frac{1}{2}$ or 1. In the latter case 'rounding up' means set $f''(a) = 2$ and 'rounding down' means set $f''(a) = 0$. The extension of $p \in CO$ to a cycle is not always possible. To obtain a feasible rounded flow f^* we do not need to extend all paths, but only those on which rounding with appropriate rounding direction leads to a capacity violation.

In Section 8.3, the rounding direction was chosen to minimize the cost of the rounded flow. With respect to the feasibility of f^* this is not the best option. Instead, we chose the rounding direction of a path $p(j, j') \in CO$ in Algorithm 16 such that neither $u(j, t)$ nor $u(j', t)$ are violated, if possible. Note that by Lemma 78 we know that $j, j' \in V_z$ if the latter is impossible and a capacity violation occurs on either (j, t) or (j', t). In this case, we want to extend $p(j, j')$ to a cycle. One way to ensure that this is possible is to assume $T = T_{all}$. Let $j \neq j' \in V_{odd}$ and $p = p(j, j') \subseteq CO$, such that rounding the arcs covered by p in either rounding direction leads to capacity violation. Let w.l.o.g. $f^*(j, t) = u(j, t) + \frac{1}{2}$.

Lemma 82 (Path Extension) *There exist two arcs* $a = (i, j)$, $a' = (i, j')$ *such that* $p = p(j, j') \cup \{\tilde{a}, a'\}$ *is a cycle* $c_p \subseteq T$.

Proof: By Lemma 78 we know that $j \in V_z$ and there is an arc (i, j) with $f(i, j) = 2k + 1, k \in \mathbb{N}$ and $i \in V_x$. Hence $a = (i, j) \in T^1$ and $\tilde{a} = (j, i) \in \tilde{T}^1$. Since we assume $T = T_{all}$ and $j' \in V_z$ by Lemma 78 as otherwise the choice of rounding direction for $p(j, j')$ would be different, there is also an arc $a' = (i, j') \in T$. □

The assumption $T = T_{all}$ is rather strong. Lemma 82 for example also holds, if we assume T to contain all arcs (i, j) for which s_i with $t_i = t_x$ is allowed for d_j with t_z. To further weaken the assumption, we need to specify or predefine the pairs of nodes $j, j' \in V_{odd}$, for which extensions of paths $p(j, j')$ are needed. Any predefinition of such pairs in our approach leads to a constrained matching problem, which is therefore addressed separately in Chapter 9.

Algorithm 17 uses the extension of paths in the CO to cycles. The resulting additional change of flow on arcs a, a' can be interpreted as reassignment of single cars of type t_x with respect to $\mathcal{D}$ and $\mathcal{D}^*$. Let f'' be as determined by Algorithm 17 and f', f^* be defined as before. Note that f^* is still integral on arcs $a \in T$ as the additional changes of flow on arcs $a \in T \setminus T^2$ with $f(a) \in \mathbb{N}$ are integral.

Lemma 83 (Feasible rounded Flow f^*) *The rounded flow* f^* *is a feasible flow in* N_I^g.

Proof: The flow is actually rounded on arcs $a \in T$ with $u(a) = \infty$, such that no direct capacity violation occurs. The extension of f^* on arcs $(s,i),(j,t) \in A \setminus T$ is determined by the flow conservation constraints for the incident supply and demand nodes i,j. For all nodes $v \neq s \neq t \notin V_{odd}$, each path $p \in CO$ or cycle $c \in CO$ which passes v enters and leaves the node the same number of times. Hence, the same number of arcs $a \in CO$ and $\tilde{a} \in CO$ are incident to v and the same amount of flow out of or into v is rounded up and down respectively. Consequently flows on the arcs (s,v) or (v,t) respectively are not changed and no capacity violation occurs. For each node $j \in V_{odd} \subseteq V_D$, exactly one path $p \in CO$ starts or ends in j with edge $e = (i,j)$ or $\tilde{e} = (j,i)$. If $f(e)$ is rounded down, no capacity violation occurs. Otherwise, if already $f(j,t) = u(j,t)$ and $f(e)$ is rounded up to $f^*(e) = f(e) + \frac{1}{2}$, $u(j,t)$ is violated.

In this case p is extended to a cycle by (j,i) and (i,j'). Further, $f^*(i,j) = f(i,j) - 1$ and $f^*(i,j') = f(i,j') + 1$. Hence, given $i \in V_x$ and the valency $v(t_x,t_z) = \frac{1}{2}$, $f^*(j,t)$ is reduced by $\frac{1}{2}$ to $u(j,t)$ and no longer violates the arc capacity. For i neither flow conservation nor capacity constraints (on (s,i)) are violated. Finally $f^*(i,j')$ is increased by $\frac{1}{2}$. This cannot violate the capacity $u(j',t)$ as f was feasible and $f^*(j',t)$ was primarily set to $f(j',t) - \frac{1}{2}$ due to the rounding direction of p. Hence, f^* obeys the capacity constraints. By the above arguments, also flow conservation holds for nodes $v \in V_S \cup V_D$. Since $f^*(s,i) = f(s,i)$ for all $i \in V_S$ (and $f(s,i) = u(s,i)$ as f is feasible) flow conservation also holds for s. With respect to t, $f^*(j,t) = f(j,t)$ for all $j \in V_D \setminus V_{odd}$. On the other hand, for each $j \in V_{odd}$ $f^*(j,t) \in \{f(j,t), f(j,t) - \frac{1}{2}, f(j,t) + \frac{1}{2}\}$. Let $j \in V_{odd}$ with $f^*(j,t) = f(j,t) - \frac{1}{2}$, then there is $j' \in V_{odd}$ and $p(j,j') \in CO$, such that $f^*(j,t) = f(j,t) + \frac{1}{2}$ and vice versa. □

With respect to the cost of f^*, there are two differences to the flow as rounded by Algorithm 16: On the one hand, we do not chose the rounding direction of paths and cycles in CO appropriate to minimize the cost $c^*(p)$ and $c^*(c)$ any longer. Hence, those costs can be twice as large as the cost with respect to f due to half-integrality.

On the other hand, $c(f)$ and $c(f^*)$ no longer only differ in such cost charged to paths and cycles. Algorithm 17 also increases flow on arcs $a \in T \setminus (T^2 \cup T^1)$ for which possibly $f(a) = 0$. In this case, $c(f)$ and $c(f^*)$ are not comparable. Therefore, we assume the cost $c(i,j)$ $(= c(j,i))$ on arcs (i,j) to obey the triangle inequality $c(u,v) \leqslant c(u,w) + c(w,v)$. This is a reasonable assumption on the (local) transport cost, but can be obscured for example by weak priority terms in the application.

Let c_p be a cycle extending $p \in CO$ by $\tilde{e} \in \tilde{T}^1$ and $e' \in T$, $c^*(c_p) = c^*(p) - c(e) + c(e')$ its cost with respect to the rounded flow f^* and $c(p)$ as before.

Algorithm 17 *Rounding the Cycle Cover*

```
Input: $T^1, \tilde{T}^1, T^2, \tilde{T}^2, V^1, V^2, V_{odd}, f, CO$
Output: $f^* = f - f' + f''$ is an integral, feasible flow in $N_I^g$
 1: for all $p \subseteq CO$ do
 2:   Choose rounding direction $d(p) \in \{1, 0\}$.
 3: end for
 4: for all $a \in T^2$ do
 5:   if $a \in p \subset CO$ then
 6:     $f''(a) = d(p)$
 7:   else
 8:     $f''(a) = 1 - d(p)$
 9:   end if
10: end for
11: for all $a = (s, i) \in A \setminus T$ do
12:   $f''(a) = \sum_{(i,j) \in T} f''(a)$
13: end for
14: for all $a = (j, t) \in A \setminus T$ do
15:   $f''(a) = \sum_{(i,j) \in T} m(i, j) f''(a)$
16:   if $f(j, t) > u(j, t)$ then
17:     Extend $p(j, j')$ to cycle by $(i, j), (j', i)$.
18:     $f''(a) = f''(a) - \frac{1}{2}$
19:   end if
20: end for
21: for all $a \in T \setminus T^2$ do
22:   if $a \in CO$ then
23:     $f''(a) = f'(a) + d(p)$
24:   else
25:     if $\tilde{a} \in CO$ then
26:       $f''(a) = f'(a) - d(p)$
27:     else
28:       $f''(a) = f'(a)$
29:     end if
30:   end if
31: end for
```

Lemma 84 (Extended Path Cost) *The costs* $c^*(c_p)$ *are at most four times the cost* $c(p)$.

Proof: We can rewrite the cost $c^*(c_p)$ as follows:

$$\begin{aligned} c^*(c_p) &= c(p) \\ &- \left(\sum_{\tilde{a} \in p} c(a) f'(a) + \sum_{a \in p} c(a) f'(a) \right) \\ &+ \left(\sum_{\tilde{a} \in p} c(a) f''(a) + \sum_{a \in p} c(a) f''(a) \right) \\ &+ c(e') - c(e) \end{aligned}$$

Ignoring the negative terms, we have $c^*(p) \leqslant 2c(p) + c(e')$ as $f(a)$ is at least $\frac{1}{2}$ on each $a \in p$. Further, due to the triangle inequality $c(e') \leqslant \sum_{a \in p} c(a) \leqslant 2c(p)$, as $c(p)$ possibly only accounts for $f(a) = \frac{1}{2}$ again on each arc a. Hence, we have $c^*(p) \leqslant 4c(p)$. □

The flow f^* only differs from f on arcs covered by CO and the additional arcs e, e' for the path extensions. As before, the different costs can be charged uniquely to the (extended) paths and cycles, such that we obtain the following corollary by Lemma 84:

Corollary 85 (Bounded General Cost) $c(f^*) \leqslant 4c(f)$

Let $\mathcal{D}$ be the derived distribution of the minimum cost flow $f = f(N_I^g)$ and $\mathcal{D}^*$ be the derived distribution of f^*.

Theorem 86 (4-approximate Distribution) *The distribution* $\mathcal{D}^*$ *is a* 4*-approximation to the optimal distribution for* I.

Proof: By Lemma 83 f^* is a feasible flow. Hence by integrality of f^* and Corollary 52 $\mathcal{D}^*$ is a feasible distribution for I. Further $c(\mathcal{D}^*) = c(f^*) \leqslant 4c(f)$ with $c(\mathcal{D}) = c(f)$ being a lower bound on the cost of an optimal distribution. □

8.5 AN ITERATIVE ROUNDING HEURISTIC

As mentioned before, our assumption $T = T_{all}$ is not always given in the application. Further, although the triangle inequality naturally holds for transport costs r_{ij}, this is not guaranteed for the cost function c. Hence, applying the 4-approximation in practice does not always yield a fully feasible dispatching. We therefore combine the 0.5-upgraded distribution with the idea of single car repositioning used in the 4-approximation.

Let $\mathcal{D} = \mathcal{D}(I)$ be an optimal 0.5-upgraded distribution (Section 8.3) for the application instance I. Remember that demands d_j, which are oversatisfied by $\mathcal{D}$, are of type t_z and receive an odd number of cars of type t_x by Lemma 78. Consequently there is a supply s_i of type t_x with $\delta_{ij} = (i, j, n_{ij} \geqslant 1) \in \mathcal{D}$ for each such demand. Given $v(t_x, t_z) = \frac{1}{2}$, the distribution $\mathcal{D}' = (\mathcal{D} \setminus \{\delta_{ij}\}) \cup \{\delta'_{ij} = (i, j, n_{ij} - 1)\}$ no longer oversatisfies the demands d_j. On the other hand $\mathcal{D}'$ does not assign all supply (of type t_x) and is thus no longer feasible. We call the cars which are not assigned by $\mathcal{D}'$ *spare* cars.

Note that the 4-approximation reassigns the spare cars to a special set of demands by rounding the flow on the path extending arcs a, a'. We now allow spare cars to be reassigned to any matching demand. For this, we compute $\mathcal{D}$ and modify it to $\mathcal{D}'(I)$ as above. Then we reduce all supplies and demands of I appropriate to the assignments $\delta_{ij} \in \mathcal{D}'(I)$. Let I' be the reduced instance.

Then I' does not contain any supplies of type t_y any longer, as they are fully distributed by $\mathcal{D}'$ and I' is (empirically) a homogeneous instance. Consequently $N(I')$ is a classical network and we obtain an integral minimum cost flow and thus an optimal solution $\mathcal{D}'' = \mathcal{D}(I')$ in polynomial time. Unfortunately, we cannot bound the additional cost $c(\mathcal{D}'')$, as the following example shows.

Consider an instance I from the application with $S = \{s_1, s_2, s_3\}$ and $D = \{d_4, d_5, d_6, d_7\}$. The unit supplies s_1, s_3 are of type t_x, unit supply s_2 of type t_y. Demands d_4,d_7 order a single car of type t_x and d_5, d_6 a single car of type t_z each. Further we have $\mathcal{T}$ such that s_1 is in time for d_4 and d_5, s_2 for d_5 and d_6 and s_3 for d_6 and d_7. The cost are $c(ij) = r$ for all supply-demand-pairs except for $c(1,4) = c(3,7) = R >> r$. Then Figure 26 shows the structure of the associated network N_I^g together with an optimal half-integral flow f (numbers in brackets).

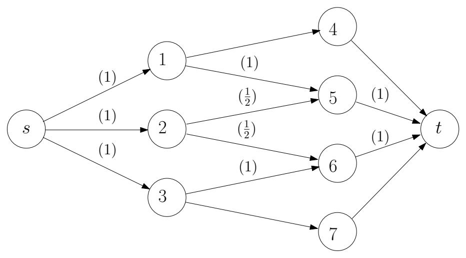


Figure 26: Associated network N_I^g for instance I with optimal half-integral flow (numbers in brackets).

W.l.o.g. we obtain $\mathcal{D} = \{(1, 5, 1), (2, 6, 1)\}$ from Algorithm 16, such that s_3 provides the spare car of type t_x as otherwise d_6 would be over-satisfied.

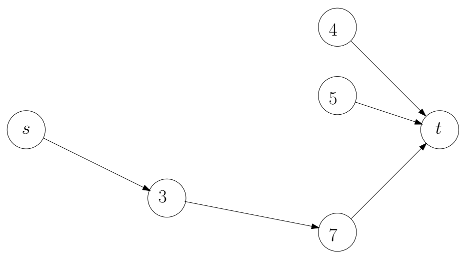


Figure 27: Associated network $N^g_{I'}$ for reduced instance I'.

Figure 27 shows the structure of $N^g_{I'}$ for the reduced instance I'. Obviously any distribution is forced to contain the single possible assignment $(3, 7, 1)$ at cost $c(3, 7) = R$. The cost ratio $\frac{c(\mathcal{D}'(I))+c(\mathcal{D}(I'))}{c(\mathcal{D}(I))} = \frac{r}{R}$ is thus unbounded.

On the other hand, given $c(1, 4) = c(3, 7) = R$, the cost $c(f) = 3r$ of the optimal half-integral flow solution are an arbitrary bad lower bound on the cost of an optimal integral flow. The latter can easily be checked to be $2r + R$ in our example and the heuristic provides an optimal distribution. This is obviously not the case in general, but the integrality gap between $c(\mathcal{D})$ and the cost of an optimal integral distribution has to be taken into account with respect to the performance of the heuristic (see Appendix A.1).

We only considered arc costs in the choice of rounding direction of Algorithm 16. By Lemma 79, this suffices for 0.5-upgraded distributions as they cause no cost increase. In Algorithm 17 costs are completely neglected as rounding directions are chosen such that the number of oversatisfied demands d_j is minimized. This is necessary as otherwise we cannot apply Lemma 78 to ensure that all demands d_j, d'_j with $j, j' \in V_{odd}$ and $p(j, j') \in CO$ have type t_z. But only in this case spare cars can possibly be rerouted feasibly.

Further, the costs for the reassignments of spare cars depend on which pairs of nodes $j \neq j' \in V_{odd}$ are connected by a path $p(j, j')$. The chosen paths $p(j, j')$ also determine if a rerouting from j to j' or vice versa is possible with respect to the given timetable, if we abandon the assumption that supplies are in time for all allowed demands (assumption $T = T_{all}$). To obtain a 4-approximation for more application instances of the (DP), we try to find a (suggestively speaking 'matching') partner $j' \in V_{odd}$ for each $j \in V_{odd}$, such that either $p = p(j, j')$ does not need to be extended or we can find arcs $a, a' \in T$ to extend p to a cycle. Then Algorithm 17 can be applied as a 4-approximation assuming only the triangle inequality for c.

So far, pairing nodes $j \neq j' \in V_{odd}$ with $p(j, j') \in CO$ is done more or less arbitrary due to Algorithm 15. An intuitive way to find pairs (j, j') would be the search for a perfect matching in a graph G_{odd} with node set V_{odd} and the edge set E_{odd} representing possible car reroutings (the existence of arcs $a, a' \in T$) between j and j'. Unfortunately, while it is always possible to find a path from node $j \in V_{odd}$ to some other node $j' \in V_{odd}$ (also on an already reduced set of arcs T^2), this is not true for predefined pairs of j and j'. Hence, it does not suffice to find a perfect (minimum weight) matching on G_{odd}. We need to find such a matching which also allows the paths $p(j, j')$ to form a cover of T^2. In other words, paths $p(j, j')$ in $G^2 = (V^2, T^2 \cup \tilde{T}^2)$ must be edge-disjoint for all matching edges (j, j') in G_{odd}. This constrained matching problem is introduced and investigated in the following Chapter 9.

9 A CONSTRAINED MATCHING PROBLEM

The matching problem is closely related to and (at least) as well-studied as the flow problem in various kinds [31, 32, 86] like (minimum weight) perfect and maximum (weight) matching. In the special case of bipartite graphs the matching problem coincides with a classical integral flow problem on an appropriate instance [39]. The central idea of augmenting paths to solve matching problems on general graphs is also lend from the work on flows.

Up to today the most famous solution algorithm for (weighted) matching problems is the polynomial time *Blossom(-Shrink)* algorithm by Edmonds [31]. It was steadily improved in terms of theoretical worst case running times [86, 46, 43, 42] and implementations [21, 90, 84]. The best currently proved bound on its running time in terms of the number of nodes and edges is $O(n(m + n \log n))$ [42]. For integral weights, also bounds in the largest weight value can be established by using scaling techniques on Edmond's Algorithm [21]. An up-to-date reference implementation is *Blossom V* by Kolmogorov [84], which is designed to find minimum weight perfect matchings.

Additional constraints create further variants of the matching problem. One of the most famous being the *stable marriage* problem introduced by Gale and Shapley [44], where we look for a bipartite matching (between 'men' and 'women'), such that given preferences for matching candidates are met. A marriage (matching) M is stable, if there is no couple ($e \in M$), for which both 'man' and 'woman' prefer another candidate. The stable marriage and also the non-bipartite variant of *stable roommates* are polynomially solvable [45, 36] with some generalizations. For others, the resulting problems are computationally hard [88]. Many other constrained matching variants are also NP-complete [66, 100] or even APX-hard to approximate [37].

In the following Section 9.1 we introduce a matching problem on a graph G with a constraint on a second graph C. For general constraint graphs C the problem is NP-complete. In Section 9.2 we present an algorithm for the case that C is a tree and prove its correctness and a bound on its running time in Section 9.3. The strategy of the algorithm is related to the proof of polynomial-time solvability of the maximum integral multi-commodity flow problem in trees with unit edge capacity [48]. We apply the algorithm to the approximation of the practical instances of the (DP) in Section 9.4.

9.1 DEFINITIONS AND IDEA

Let $G = (V, E)$ and $C = (V', E')$ be undirected graphs, such that $V \subseteq V'$. We call the vertices $v \in V$ *terminals* with respect to C to distinguish them from vertices only present in V'. We define the constrained matching problem on G with edge-disjoint paths in C (MEDPIC) as follows:

Definition 87 (MEDPIC) *Given graphs G, C, find a maximum matching $M(G, C)$ in G, such that for all $(u, v) \in M$ the vertices $u, v \in V'$ can simultaneously be connected by edge-disjoint paths in C.*

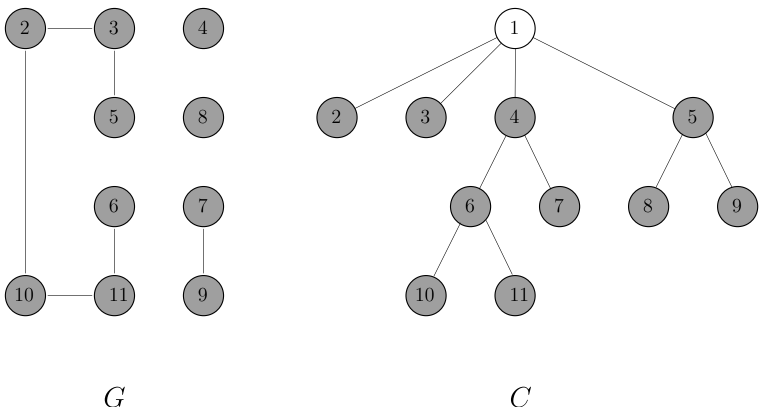


Figure 28: MEDPIC instance: We search for a maximum matching $M = M(G, C)$ in G, such that for all $(u, v) \in M$ u and v can simultaneously be connected by edge-disjoint paths in C. The terminals (grey nodes) V are identified with the corresponding vertices in V'.

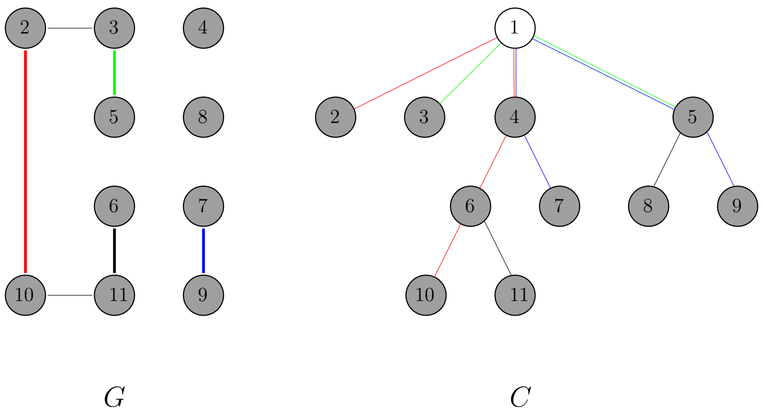


Figure 29: The maximum matching (bold, coloured edges) on G is not edge-disjoint in the tree C (doubly coloured edges).

Consider the following example. Figure 28 shows graph G on which we search for the matching $M(G, C)$ and constraint graph C which must provide the edge-disjoint paths for terminals $u, v \in V$ with $(u, v) \in M(G, C)$. The terminals $V \subseteq V'$ are shaded grey. Here we see that E and E' are not required to have a special subset relation. Note that C is a (rooted) tree.

The maximum matching $M = \{(2, 10), (3, 5), (6, 11), (7, 9)\}$ on the graph G is displayed by bold (and partly colored) edges in Figure 29. The graph cannot contain a perfect matching (with cardinality 5) due to the isolated vertices 4 and 8. In Figure 29, the paths $p(u, v)$ in C with $(u, v) \in M$ are colored corresponding to the respective matching edge in G. We see that the paths $p(2, 10)$ and $p(7, 9)$ as well as the paths $p(3, 5)$ and $p(7, 9)$ are not edge-disjoint. Hence M is no solution to (MEDPIC) on G and C.

We see in the example that out of each subtree $st(r_u)$ at most one terminal $u \in V$ can be connected to a terminal $v \in V$ in a disjoint subtree $st(r_v)$. As both terminals 5 and 9 are in the subtree $st(5)$ they cannot both be connected to terminals in other subtrees. As conversely with respect to G both are only connected to the vertices 3 and 7, which happen to be in different subtrees of C, $M(G, C)$ cannot match both vertices.

An analogous case occurs for the terminals 7 and 10. Hence a solution to (MEDPIC) on G and C has maximum cardinality 3. The two feasible matchings with this cardinality are $M_1 = \{(2, 3), (7, 9), (10, 11)\}$ and $M_2 = \{(2, 10), (3, 5), (6, 11)\}$, which both solve (MEDPIC).

Unfortunately to decide whether a graph contains edge-disjoint paths for $k \geqslant 2$ terminal pairs is NP-complete in general [40] and remains so for planar graphs and $k > 2$ [47, 91]. The latter is also true for partial $l-$trees (graphs of treewidth bounded by a fixed integer l) with $k > 2$ [121] as well as for bidirected trees with arbitrary degree [34]. Thus (MEDPIC) is also NP-complete in the above settings.

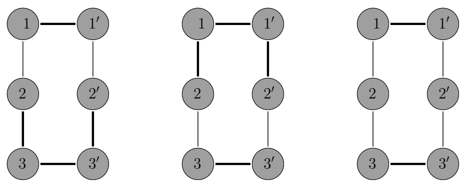


Figure 30: Consider $G = K_{3,3}$ on the nodes $\{1, 2, 3\} \cup \{1', 2', 3'\}$ and C the cycle $1 - 2 - 3 - 3' - 2' - 1' - 1$. Then not all pairs (u, u') can be connected simultaneously by edge-disjoint paths in C, but any two of them can. The latter is illustrated for the terminal pairs $\{(1, 1'), (2, 2')\}$, $\{(2, 2'), (3, 3')\}$ and $\{(1, 1'), (3, 3')\}$ respectively in the pictures above. The corresponding edge-disjoint paths are displayed as bold edges.

Let G be a $K_{3,3}$ with node set $\{1, 2, 3\} \cup \{1', 2', 3'\}$ and edges $\{(i, i')\}, i = 1, 2, 3$. Further, let C be a cycle $1 - 2 - 3 - 3' - 2' - 1' - 1$ on the node set of G. Then of the three terminal pairs $p_i = (i, i'), i = 1, 2, 3$, any two can be connected by edge-disjoint paths in C as shown in Figure 30, but not all three terminal pairs. Hence a maximum matching $M(G, C)$ again cannot be a perfect matching in G.

Also note that the path between terminals 2 and $2'$ changes due to the second chosen terminal pair: if we consider pairs $(1, 1')$ and $(2, 2')$ then $p_1(2, 2') = 2, 3, 3', 2'$ and considering pairs $(2, 2')$ and $(3, 3')$ then 2 and $2'$ are connected by $p_3(2, 2') = 2, 1, 1', 2'$. Analogous situations can come up with sets of terminal pairs of any cardinality in the presence of arbitrary cycles. This observation gives us the intuition that checking the existence of edge-disjoint paths explicitly for arbitrary constellations of matching edges is exponential in the number of terminals.

In constrast, given that $C = T$ is a tree, any chosen terminal pair (u, v) determines the unique path $p(u, v)$ in T. Thus, any two terminal pairs can either be connected by edge-disjoint paths or not (independent of a third terminal pair). Consider a so-called *conflict graph* with a node n_e for any edge e in G. Further, let the conflict graph contain an edge (n_e, n_e') between two nodes, if the corresponding edges of G connect two terminal pairs whose unique connecting paths in C are not edge-disjoint. We call such a pair of edges *conflicting*. (The size of the conflict graph is quadratic in the size of G.)

Observe that a maximum matching in G, which does not contain any conflicting edges, is a MEDPIC solution. This problem formulation can be interpreted as a *matching problem under disjunctive constraints (MMCG)* [27] as given by the conflict graph. The latter problem is already NP-hard for a conflict graph whose connected components are edges. Conversely not all instances of (MMCG) with such a conflict graph can be interpreted as MEDPIC instances where the constraint graph is a tree. Hence, the former result does not prevent a polynomial time solution to MEDPIC in this case.

We present such a polynomial time solution for $C = T$ by employing the well-known *dynamic programming* paradigm (see for example [24]). We therefore determine the maximum set of edge-disjoint paths between matchable terminal pairs in growing subtrees of T, instead of checking the constraint in substeps of the matching solution. This is prepared by a necessary and sufficient condition for the existence of two edge-disjoint paths connecting two terminal pairs in T by Lemma 88.

In the following we assume T to be rooted (arbitrarily). Let $P = \{(u_i, v_i)\}$ be a set of matchable terminal pairs and let p_i denote the unique path between u_i and v_i in T. Further, let the *least common ancestor* for a pair of terminals u_i, v_i be the root r_i of the smallest subtree $\mathrm{st}(r_i)$ of T, such that $\mathrm{st}(r_i)$ contains u_i and v_i.

Lemma 88 (Disjoint Paths Condition) *The paths $p_i \in P$ are pairwise edge-disjoint if and only if no subtree $\mathrm{st}(r)$ of T contains two terminals $u_k, u_l, k \neq l$ and neither v_k nor v_l.*

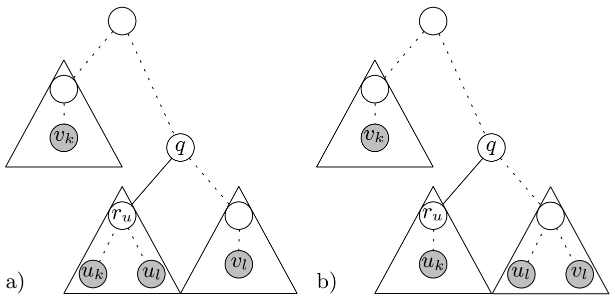


Figure 31: Necessary and sufficient condition for pairwise edge-disjoint paths in a tree: a) subtree $\mathtt{st}(r_u)$ contains terminals u_k and u_l, but neither terminal v_k nor v_l. Consequently the solid edge (q, r_u) must be contained in both paths p_k and p_j which cannot be edge-disjoint. b) subtree $\mathtt{st}(r_u)$ only contains terminal u_k (and not v_k). Thus p_k must use edge (q, r_u), but no other path needs to enter $\mathtt{st}(r_u)$ and hence the solid edge is only used once.

Proof: Consider two pairs of terminals $(u_k, v_k), (u_l, v_l) \in P$ and let no two subtrees of T contain exactly one terminal of each pair. The path p_k (p_l) consist of the paths from both terminals u_k, v_k (u_l, v_l) to their least common ancestor r_k (r_l). Let $e = (q, r)$ be an arbitrary edge on p_k (p_l) and let q be closer or equal to r_k (r_l), such that e leads to the smaller subtree $\mathtt{st}(r)$. As no subtree of T contains exactly one terminal of each pair, this especially holds for the subtree $\mathtt{st}(r)$. Thus no edge e can possibly be used by both paths, because not both paths need to enter the respective subtree. Conversely, consider a pair of paths p_k, p_l in T which is not edge-disjoint and shares the edge $e = (q, r)$. Let q be closer or equal to r_k and r_l. Then r is the root of a subtree containing exactly one terminal of each pair, w.l.o.g. u_k, u_l, otherwise either p_k or p_l would not enter $\mathtt{st}(r)$ and thus e would not belong to both of the respective paths. □

Figure 31 illustrates both cases of Lemma 88. In subfigure a) paths p_l and p_k both need to use edge (q, r_u), because the subtree $\mathtt{st}(r_u)$ contains both terminals u_k and u_l, but neither v_k nor v_l. Hence, the terminal pairs cannot be connected by a pair of edge-disjoint paths and a MEDPIT solution cannot contain both edges (u_k, v_k) and (u_l, v_l). Note that the roles of u_k and v_k (u_l and v_l) can be exchanged. In subfigure b) $\mathtt{st}(r_u)$ only contains u_k, such that only p_k needs edge (q, r_u). Notice that $\mathtt{st}(q)$ again contains u_k *and* u_l, but also v_l, a partner to one of the other terminals.

9.2 THE MEDPIT-ALGORITHM

In the following we present a dynamic programming algorithm to solve instances of MEDPIC with an arbitrary graph G and trees C = T. We call the problem restricted to such instances MEDPIT. Lemma 88 provides us with the central idea for our dynamic programming approach to the MEDPIT on $G = (V, E)$ and $T = (V', E')$. We recursively compute several (sub)matchings in G on subsets of terminals which correspond to terminals occurring in subtrees $st(r)$ of T. For such a subtree, we assume exactly one or no terminal occurring in $st(r)$ to be matched to a terminal outside $st(r)$. Due to edge-disjointness constraint and Lemma 88, one of these submatchings will be part of the MEDPIT solution $M(G, T)$.

We denote by $V(r)$ all nodes in the subtree $st(r)$ and by $T(r)$ all terminal nodes in the subtree. Further, let $S(r)$ denote all direct sons of r and $ST(r)$ all terminals amongst the direct sons of r, including r, if r is a terminal. The recurrence follows a depth first search (post) order in T. In the following, we explain how to obtain the necessary submatchings on leaves (trivial case), subtrees of height one (non-trivial special case), and subtrees of greater height. We start for each case by describing the computation of the submatching for which no terminal is matched outside the subtree. Note that for the root of T this is the only case to be considered.

As the subtree $st(l)$ induced by a leaf l can at most contain one terminal, a corresponding submatching is always empty and receives value 0. We therefore start our explanation with a subtree $st(r)$ of height $h(st(r)) = 1$, which is the first non-trivial case to be considered. For those smallest non-leaf subtrees, we compute maximum cardinality matchings in the subgraphs of G induced by $ST(r)$ (the sons of r), which are terminals, including r if it is a terminal. Further, we subsequently compute such matchings in the subgraphs of G induced by $ST(r) \setminus OUT$ with $OUT = \{u\}$ for all nodes $u \in ST(r)$. The matchings are denoted by $M(r, OUT)$ and annotated with a cost term $c(M) = -|M(r, OUT)|$ for each matching. (The cost term associated with leafs is 0.)

Figure 32 shows the induced subgraphs $G(6, OUT)$ for one of the smallest non-leaf subtrees $st(6)$ of C in the previous example (Figure 28). The graph $G(6, \emptyset)$ contains two possible submatchings of equal size 1 (edge $(6, 11)$ or edge $(10, 11)$). The two graphs $G(6, 6)$ and $G(6, 10)$ each allow for one of these submatchings and Graph $G(6, 11)$ consists of two isolated vertices and thus contains no possible submatching. The corresponding cost terms are therefore $c(M(6, \emptyset)) = -1$ (and we select one of both matchings arbitrarily as a representative), $c(M(6, 6)) = c(M(6, 10)) = -1$ and $c(M(6, 11)) = 0$.

To obtain the maximum cardinality matchings in general, we compute a minimum weight perfect matching on an associated graph as proposed in [22]. Therefore the node set is duplicated and all edges between original nodes are mirrored in the additional node set (*mirror edges*). Further a node is connected to its duplicate via a *self edge*. All original edges receive weight -1 and all added mirror and self edges receive weight zero.

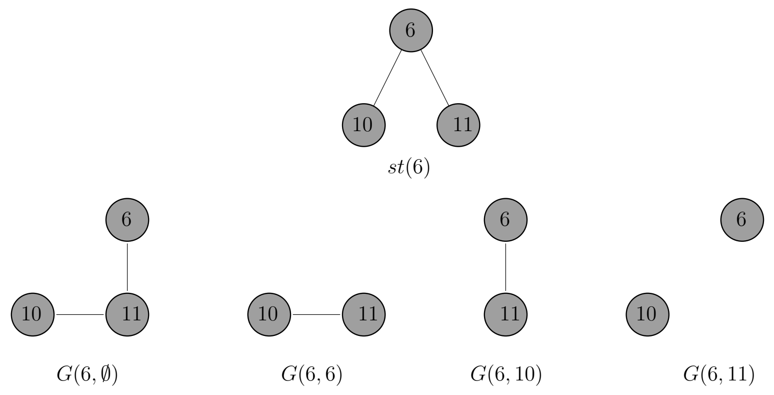


Figure 32: The subtree $st(6)$ of vertex 6 in C has the corresponding induced subgraphs $G(6,\emptyset), G(6,6), G(6,10), G(6,11)$ on which submatchings are computed.

Subtrees $st(r)$ of T with height $h(st(r)) > 1$ are treated in the following way. Again we compute matchings on (a subset of) r's direct sons and possibly r. We take advantage of the submatchings, which are prepared for all sons due to the recurrence order. Yet, for this purpose, it is no longer sufficient to compute maximum matchings on induced subgraphs of G: a son s of r might not be a terminal itself and thus $s \notin V$. On the other hand, if the set of terminals $T(s)$ in the subtree $st(s)$ is not empty, those terminals need to be represented in the submatching in some way. To achieve a proper representation, we exploit the transformation of a maximum cardinality matching into a minimum weight perfect matching as follows.

Firstly, we modify $S(r)$ by removing all sons s of r with $T(s) = \emptyset$. The nodes $s \in S(r)$ are now either terminals or nodes which represent at least one terminal in their corresponding subtree. Let $S(r)$ be redefined appropriately and contain r, if and only if it is a terminal itself. Secondly, we successively compute submatchings $M(r, OUT)$ for the cases that exactly one or no terminal of $st(r)$ is matched to a terminal outside the subtree as before.

We start with the construction of the graph $G(r,\emptyset)$ for the case that no terminal in $st(r)$ is matched to a terminal not in $st(r)$. Let $n = |S(r)|$, then $G(r,\emptyset)$ contains $2n$ vertices $\{v_0, \dots, v_n\} \cup \{v'_0, \dots, v'_n\}$. We call the additional node v'_i the *shadow node* of v_i. For each pair of nodes $v_i \neq v_j \neq r$, we check if the set of terminals $T(v_i)$ in $st(v_i)$ contains a terminal t_i and $T(v_j)$ contains a terminal t_j, such that (t_i, t_j) is an edge in G. If so, we add (v_i, v_j) to the edge set of $G(r,\emptyset)$.

Consider $(v_i, v_j) \in M(r,\emptyset)$. Then the terminals t_i and t_j must be connected by a path in T, which is edge-disjoint to all other such paths between matched terminals in $M(r,\emptyset)$. Thus, for both subtrees $st(v_i)$ and $st(v_j)$, no other terminal than t_i and t_j can be matched outside $st(v_i)$ and $st(v_j)$ respectively. Consequently the maximum submatching of both subtrees are in this case the previously computed $M(v_i, \{t_i\})$ and $M(v_j, \{t_j\})$ respectively. We represent this situation by setting the cost of $e = (v_i, v_j)$ to $c(e) = c(M(v_i, \{t_i\})) + c(M(v_j, \{t_j\})) - 1$, where -1 accounts for the additional matched edge (t_i, t_j). We also add the edge (v_i', v_j') between the corresponding shadow nodes with zero cost. If $T(v_i)$ and $T(v_j)$ contain more than one matchable terminal pair $(t_i, t_j) \in E$ we consequently choose the one minimizing the above given cost term to represent the maximum cardinality submatching in the union of both subtrees.

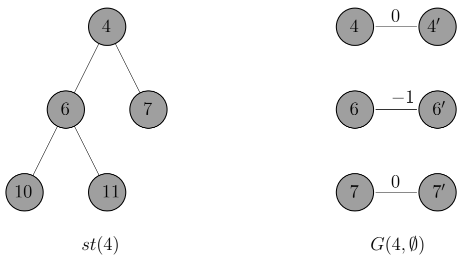


Figure 33: Graph $G(4, \emptyset)$ for the subtree $st(4)$ of vertex 4 in C. The number at the edges are costs corresponding to represented submatchings.

So far, we left out the root r in the case that it is a terminal. Although $T(r) \neq \{r\}$ in general, the node only represents itself. As the terminals $T(r) \setminus \{r\}$ are completely and uniquely covered by the subtrees $st(v_i), v_i \in S(r) \setminus \{r\}$ this is necessary to avoid double counting of the associated submatchings. We thus add an edge $e = (v_i, r)$ only if there is a terminal $t_i \in T(v_i)$ such that $(t_i, r) \in E$ and we set $c(e) = c(M(v_i, \{t_i\})) - 1$. We also add the shadow edge (v_i', r') with zero cost.

Further, the self edges (v_i, v_i') are added to $G(r, \emptyset)$. A self edge (v_i, v_i'), in the matching $M(r, \emptyset)$ either represents that a terminal is not matched if $v_i = r$ or v_i is a leaf or that out of the subtree $st(v_i)$ no terminal is matched to a terminal outside $st(v_i)$. Thus we set the cost of $e = (v_i, v_i')$ to $c(e) = 0$ in the first case and to $c(e) = c(M(v_i, \emptyset))$ in the second case. This completes the construction of $G(r, \emptyset)$ for $h(st(r)) > 1$. Note that its structure resembles the transformed graph to compute a maximum cardinality submatching via minimum weight perfect matching in subtrees of height one.

Figure 33 shows $G(4, \emptyset)$ for the subtree $st(4)$ of C in the example of Figure 28. Here, vertex 4 is the root only representing itself as it is a terminal. The self edge $(4, 4')$ thus receives zero cost. Apart from 4, the set $S(4)$ only contains the nodes 6 and 7. The latter is a terminal and a leaf and thus also only represents itself. Consequently $c(7, 7')$ is also zero. Node 6 represents the subtree $st(6)$ of height 1. The maximum submatching on $T(6)$ with no terminal being matched outside $st(6)$ has cardinality 1, as we saw above. Thus the self edge $(6, 6')$ obtains cost -1. No other edges are added to $G(4, \emptyset)$, as there are no edges in G between 4, 7 and any terminal in $st(6)$.

Algorithm 18 *Construct*

```
Input: $G = (V, E), T = (V', E'), r, t$
Output: $G = (V_g, E_g) = G(r, t)$
 1: $V = E = \emptyset$
 2: if $r \in S(r)$ and $r \neq t$ then
 3:     $V_g = \{r, r'\}, E_g = \{e = (r, r')\}, c(e) = 0$
 4: end if
 5: for all $v \neq r \in S(r), T(v) \neq t$ do
 6:     $V_g = V_g \cup \{v, v'\}, E_g = E_g \cup \{e = (v, v')\}$
 7:     if $t \notin T(v)$ then
 8:         $c(e) = c(M(v, \emptyset))$
 9:     else
10:         $c(e) = c(M(v, \{t\}))$
11:     end if
12: end for
13: for all $v \neq w \in S(r) \setminus \{r\}$ do
14:     if $\exists v_t \neq t \in T(v), w_t \neq t \in T(w) : (t_v, t_w) \in E$ then
15:         $E_g = E_g \cup \{e = (v, w), e'(v', w')\}$
16:         $c(e) = c(M(v, \{t_v\})) + c(M(w, \{t_w\})) - 1, c(e') = 0$
17:     end if
18: end for
19: for all $v \neq r \in S(r)$ do
20:     if $\exists v_t \neq t \in T(v) : (t_v, r) \in E$ then
21:         $E = E \cup \{e = (v, r), e' = (v', r')\}$
22:         $c(e) = c(M(v, \{t_v\})) - 1, c(e') = 0$
23:     end if
24: end for
25: return $(V_g, E_g)$
```

For zero/negative weights on edges (v_i, v_j) and positive/zero weight on shadow and self edges the structure of $G(r, \emptyset)$ always provides a perfect matching M, such that edges $(v_i, v_j) \in M$ coincide with the edges of a maximum cardinality matching on the subgraph of $G(r, \emptyset)$ induced by $\{v_0, \dots, v_{n(r)}\}$ [22]. For our purpose, the edge weights encode cardinalities of submatchings, instead of the weight of a single additional matching edge. Further, we count each matching edge with -1. Thus we obtain a MEDPIT submatching of maximum cardinality on the terminals $T(r)$ by computing a minimum weight perfect matching on $G(r, \emptyset)$. This can be done for example by an implementation of Edmond's blossom shrink algorithm like [84]. The weight of the submatching counts the cardinality of the matching in $G(r, \emptyset)$ as well as the cardinality of all maximum possible submatchings in the respective subtree representative. The overall cardinality is maximized (due to negative weights).

Now we are done for leaves, non-leaf subtrees of height one and for the root r of T ($st(r) = T$). For all other subtrees, there are still the cases to consider, where exactly one terminal is matched to a terminal outside the subtree. For subtrees $st(r)$ of height one, it was sufficient to consider one submatching for each $s \in ST(r)$. We extend this procedure to all terminals $t \in T(r)$.

Each graph $G(r, \{t\})$ is constructed in a similar way as $G(r, \emptyset)$, only possibly on two vertices less (one node and its shadow node) than $G(r, \emptyset)$. This is the case, if $t = r$ or t is the only terminal in a subtree $st(s), s \in S(r)$. Otherwise $s \in S(r)$ with $t \in T(s)$ still represents the terminals $T(s) \setminus \{t\}$ and remains in $G(r, \{t\})$. Generally the edge set of $G(r, \{t\})$ is determined as above, except that for all edges (v_i, v_j) the terminals t_i, t_j must not equal t. Moreover, as t is potentially matched to t' outside $st(r)$, a path $p(t, t')$ must exist, which is edge-disjoint to all other paths between matched terminals. Hence, there can be no edge (s, v_j) in $G(r, \{t\})$, although there may be terminals $t_s \in T(s) \setminus \{t\}$ and $t_j \in T(v_j) \setminus \{t\}$ with $(t_s, t_j) \in E$. Further, the self edge $e = (s, s)$ is associated with the cost $c(e) = c(M(s, \{t\}))$ instead of $c(M(s, \emptyset))$. Note that each perfect matching on $G(r, \{t\})$ is forced to include (s, s). This guarantees the edge-disjointness of $p(t, t')$ in larger submatchings.

Algorithm 19 *MEDPIT*

```
Input: $G = (V, E), T = (V', E'), r$
Output: $M(r, \emptyset)$ is MEDPIT on G with T
 1: while $S(r) \setminus \{r\} \neq \emptyset$ do
 2:     choose $s \in S(r) \setminus \{r\}$
 3:     MEDPIT(G, T, s)
 4:     $S(r) = S(r) \setminus \{s\}$
 5: end while
 6: for all $t \in T(r)$ do
 7:     Construct $G(r, \{t\})$ by Algorithm 18
 8:     $M(r, \{t\})$=minwpm($G(r, \{t\})$)
 9: end for
10: Construct $G(r, \emptyset)$ by Algorithm 18
11: $M(r, \emptyset)$=minwpm($G(r, \emptyset)$)
```

We describe the construction of graphs $G(r, OUT)$ in Algorithm 18. Algorithm 19 provides the recursive depth first search framework to solve the MEDPIT. We assume that the computed submatchings and its costs are stored in an appropriate data structure and the method *minwpm* computes the minimum weight perfect matching on a given graph $G(r, OUT)$. The submatching on a graph $G(r, OUT)$ provides the matched edges of G explicitly and the sets $S(r)$, $ST(r)$, $T(r)$ are available at global scope if necessary.

9.3 CORRECTNESS AND RUNTIME

To prove the correctness of Algorithm 19 we show that the edge-disjoint paths constraint is met by $M(root, \emptyset)$ (Lemma 89). Further, the cardinality $|c(M(root, \emptyset))|$ of $M(root, \emptyset)$, is maximum amongst all matchings respecting the constraint (Theorem 90). Let $M(r, OUT)$ be a minimum weight submatching with respect to the subtree $st(r)$ of T as computed by Algorithm 19.

Lemma 89 (Edge-disjoint Paths Constraint) *The paths* $p(u, v) \in T$ *for all* $(u, v) \in M(r, OUT)$ *are edge-disjoint.*

Proof: We apply induction over the size of subtrees $st(v), v \in V'$ of T. For all leaves $q \in V'$, $M(q, OUT) = \emptyset$ and the claim holds. For inner nodes $r \in V'$ either $T(r) = \emptyset$ and $M = M(r, OUT) = \emptyset$ with $c(M) = 0$ and the claim holds as before or the (sub)matching $M = M(r, OUT)$ composes of the following types of (unresolved) edges (u, v) in $G(r, OUT)$:

1. $u \neq v \in S(r)$ and
   $u = r$ ($v = r$) and v (u) is a leaf or both are leaves:
   Assume $u \neq v \neq r$. Then $p(u, v)$ in T composes of two edges (u, r) and (r, v). Both terminals u, v only represent themselves in the graphs $G(r, OUT \cap \{u, v\} = \emptyset)$, where (u, v) can occur in a submatching. Further, neither u nor v can be matched twice due to the definition of a matching. Thus edges (u, r) and (r, v) cannot possibly be used by another path in T. The same is true for the case $u = r$ ($v = r$) and the single edge (r, v) ((u, r)).

2. $u \neq v \in S(r)$ and
   $u = r$ or u is a leaf and v is the root of the non-leaf subtree $st(v)$:
   Let u be a leaf. Then $(u, t_v) \in E$ for a terminal $t_v \in T(v)$ and $p(u, t_v)$ in T consists of edges (u, r), (r, v) and the path $p(v, t_v)$. As neither u nor v can be matched twice, no other path uses edges (u, r) and (r, v). The edges of $p(v, t_v)$ can also not be used by another path: as $(u, v) \in M$, Algorithm 19 chooses submatching $M(v, t_v)$ on $G(v, \{t_v\})$. The latter contains only edge-disjoint paths including a potential path $p(t_v, t)$, with t outside $st(v)$ by induction. The latter is true as Algorithm 19 recursively ensures submatchings $M(w, \{t_v\})$ are attained on the series of smaller subtrees $st(w) \subset st(v)$ containing t_v. If $u = r$, the argument is repeated, only $p(u = r, t_v)$ in T does not contain an edge (u, r).

3. $u \neq v \in S(r)$ and
   u (v) is the root of the non-leaf subtree $st(u)$ ($st(v)$):
   Then $(t_u, t_v) \in E$ for terminals $t_u \in T(u)$, $t_v \in T(v)$. The path $p(t_u, t_v)$ in T consists of the path $p(t_u, u)$, the edges (u, r), (r, v) and the path $p(v, t_v)$. As neither u nor v can be matched twice, no other path uses edges (u, r) and (r, v). The edges of $p(t_u, u)$ and $p(v, t_v)$ also cannot be used by another path: as $(u, v) \in M$, Algorithm 19 chooses submatchings $M(u, t_u)$ on $G(u, \{t_u\})$ and $M(v, t_v)$ on $G(v, \{t_v\})$. The latter contain only edge-disjoint paths including potential paths $p(t_u, t)$ and $p(t_v, t')$ by induction (see case 2).

□

Theorem 90 (Correctness) $M(\mathrm{root}, \emptyset)$ *solves MEDPIT on* G *and* T.

Proof: By Lemma 89 the set of path $p(u, v)$ in T with $(u, v) \in M(\mathrm{root}, \emptyset)$ is edge-disjoint. Further, $M(\mathrm{root}, \emptyset)$ is a minimum weight perfect matching on $G(\mathrm{root}, \emptyset)$ and thus a minimum weight matching on the subgraph of $G(\mathrm{root}, \emptyset)$, induced by the non-shadow nodes. The cost of each edge $e = (u, v)$ in $G(\mathrm{root}, \emptyset)$ are $c(e) = -|M(u, \{t_u\})| - |M(v, \{t_v\})| - 1$. Thus the cost value counts -1 for each possible match edge in $st(u)$ and $st(v)$ under the constraint that terminal t_u in $st(u)$ is matched to t_v in $st(v)$. The cost of a self edge $e = (u, u')$ are $c(e) = -|M(u, \emptyset)|$ for $u \neq r$ (and zero otherwise), which counts -1 for each possible match edge in $st(u)$ under the constraint no terminal of $st(u)$ is matched outside $st(u)$. The costs of shadow edges are always zero.

The minimum weight matching $M(\mathrm{root}, \emptyset)$ on $G(\mathrm{root}, \emptyset)$ therefore selects the maximum sum of cardinalities of submatchings and matched edges (u, v) in $G(\mathrm{root}, \emptyset)$, under the matching and disjoint edge constraints. As the same is true recursively for all submatchings $M(u, OUT)$, $M(v, OUT)$ selected by $(u, v) \in M(\mathrm{root}, \emptyset)$, it solves MEDPIT on G and T. □

In the following, we estimate the running time of Algorithm 19 in the number of nodes and edges of G and T. Let n_T be the number of nodes in the tree T and n_G the number of nodes in the graph G. Then submatchings are computed for each subtree $st(r)$ of T, which are $O(n_T)$. The number of submatchings on each $st(r)$ is determined by the number $|T(r)| \in O(n_G)$ of terminals which are contained in the subtree. Thus the number of computed (sub)matchings is $O(n_T n_G)$

Further, each submatching is computed on a graph $G(r, OUT)$ with less or equal $|S(r)| \in O(n_T)$ nodes and $O(n_T^2)$ edges. As each node in $G(r, OUT)$ represents at least one terminal of G, we can limit the size of $G(r, OUT)$ more strictly with $O(n_G)$ nodes and $O(n_G^2)$ edges. To check existence and cost for an edge (u, v), we have to consider all possible pairs $t_u \in T(u)$ and $t_v \in T(v)$, which are in $O(n_G^2)$. We can also consider each terminal pair and add an edge to $G(r, OUT)$ between the representative of two subtrees, if the terminals are in those different subtrees. (If there already is such an edge, we can try and improve its weight.) Thus the construction time for $G(r, OUT)$ is in $O(n_G^2)$.

The computation of $M(r, OUT)$ on $G(r, OUT)$ is in $O(n_G^3)$ which dominates the construction time. Due to the number $O(n_T n_G)$ of necessary submatchings, we obtain:

Corollary 91 (Run Time) *The running time of Algorithm 19 is in* $O(n_T n_G^4)$*).*

9.4 APPLICATION OF MEDPIT TO THE DP

Let I be an instance of the (DP) with $T \neq T_{all}$ and f a half-integral minimum cost flow in $N_I^g = (V_I, A_I)$ as described in Chapter 7. Further, let T^2, V^2 and V_{odd} be defined as in Chapter 8. Consider the undirected graph $C = C(I) = (V', E')$ with $V' = V^2$, $E' = \{(u, v) | (u, v) T^2 \cup \tilde{T}^2\}$ and assume that C is acyclic. Then Algorithm 19 can be applied to check if Algorithm 17 provides a 4-approximation on I and/or to take the cost of the possible reroutings into account. We describe both approaches in the following.

The demands, which can be oversatisfied after rounding are the vertices of V_{odd}. For each two $j, j' \in V_{odd}$, which are end points of a path $p(j, j')$ in a cover of T^2, one can be oversatisfied. The application of Algorithm 17 therefore requires the assumption that all such paths can be extended to cycles by arcs $a, a' \in A_I \setminus T^2$. These arcs represent the possibility to reroute a car of type t_x from a supply s_i to demand d_j', if it was formerly assigned to the oversatisfied demand d_j.

If arcs a, a' do not exist for all possible pairs $j \neq j' \in V_{odd}$, then paths $p(j, j')$ should not be chosen as arbitrarily as Algorithm 15 suggests. Instead, we are interested in pairing vertices of V_{odd} in such a way that the number of extendable paths is maximized. Then, if all paths are extendable, Algorithm 17 provides a 4-approximation on the respective instance. This goal naturally translates into a maximum cardinality matching on the *rerouting graph* $G_{odd} = G_{odd}(I) = (V_{odd}, E_{odd})$ with $E_{odd} = \{(j, j') | a = (i, j), a' = (i, j') \in A_I\}$.

Let $M(G_{odd})$ be a maximum matching on G_{odd} with respect to cardinality. Then it predefines the endpoints j, j' of extendable (!) paths $p(j, j')$. To successfully apply the approximation algorithm, the paths $p(j, j')$ (and potentially some additional cycles) must provide a cover of T^2. Hence, they must especially be edge-disjoint in C. Therefore the endpoints suggested by $M(G_{odd})$ cannot always be used. Instead, we exactly need to solve the MEDPIT on G_{odd} and C as defined above.

If we obtain a perfect matching as a solution to the MEDPIT, Algorithm 17 provides a 4-approximation without the assumption $T = T_{all}$. The application of Algorithm 19 to G_{odd} and C can be carried out as a preprocessing to the approximation in any case, as the depth first search framework can easily be extended to recognize cycles.

We can further consider the rerouting cost. The cost of rerouting a small car of s_i to d'_j, which was originally assigned to d_j, is the difference between the arc cost of (i, j) and the cost of (i, j'). (The term $c(i, j)$ is saved in the new solution.) Observe that a rerouting also (pre)defines the rounding direction on the path $p(j, j')$ and it can no longer be chosen according to lower cost. Thus, for the cost of a rerouting we also take the potentially higher costs of the rounding into account.

Let $c(j)$ denote costs of the rounded flow on $p(j, j')$, if the edge incident to j is rounded up. (This is the rounding direction, if a small car is rerouted from j to j'.) Let $c(j')$ be defined analogously. Then we define a *rounding penalty* term $\phi(j, j')$ for the rerouting from j to j' as $\phi(j, j') = c(j) - c(j')$. If $\phi(j, j')$ is negative, the rerouting requires the same rounding direction that would be preferred by the rounding procedure anyway. If $\phi(j, j')$ is positive, the extra cost for the forced choice of the more expensive rounding is charged to the rerouting cost. We define edge weights $c^w : E_{odd} \to \mathbb{N}$ as follows:

$$c^w(j, j') = \min\{c(i, j) - c(i, j') + \phi(j, j'), c(i, j') - c(i, j) + \phi(j', j)\}.$$

We now require a minimum weight matching on G_{odd} under the constraint that C still provides edge-disjoint paths for all matched terminals. This can again be restated as a minimum weight perfect matching as described above ([22]). Consequently we change the cost structure of the graphs $G(r, OUT)$ in Algorithm 19, such that an edge between two terminals is not associated with weight -1, but with $c^w(e)$ and the cost of a submatching is replaced by the sum of weights of their matched edges. As the latter are positive terms, we set the cost of shadow edges and self edges (r, r') to the sum of all weights $W = \sum_{e \in E_{odd}} c^w(e)$. This way, edges (u, v) in $G(r, OUT)$ which correspond to (submatchings with more) matching edges in G_{odd} always reduce the cost of the minimum weight perfect matching on $G(r, OUT)$ and assure that the cardinality of the matching is still maximized with respect to matching and edge-disjointness constraints.

With respect to the (DP), the MEDPIC so far is not generally applicable in practice, given the complexity of the edge-disjoint paths problem for unbounded numbers of terminals in arbitrary graphs. Still there are potentially more classes of graphs on which the constraint can be checked in polynomial time. This could also lead to further polynomially solvable special cases of MEDPIC.

10 CONCLUSION AND OUTLOOK

In this thesis we modelled and solved the empty freight car distribution problem under the model assumptions and requirements of DB Schenker Rail Deutschland AG. The respective constraint set partitions (DP) instances into those with homogeneous and heterogeneous substitution rules. The former can be solved efficiently to optimality (see Chapter 5), whereas the latter pose an NP-complete problem (see Chapters 4, 7) and are addressed with approximative and heuristic techniques (see Chapter 8) for the operational purpose.

All solutions where obtained via a combinatorial approach, namely a modification of the *Successive Shortest Path* algorithm [2] and implemented as a prototype. The results (see Appendix A) encourage its application for both the homogeneous and the heterogeneous (DP) instances in terms of running times and solution quality (with respect to the heuristic). Running times and heuristics especially benefit from our reoptimization strategy (see Chapter 6).

The heterogeneous (DP), which was also investigated as an interesting theoretical problem, was so far mainly addressed from a practical point of view in this thesis. We mainly considered special 2-to-1-substitution schemes occurring in the application during the development of approximative and heuristic solutions.

A focus for further research could be the generalization of those approaches to heterogeneous substitution schemes with arbitrary structures and valencies. Rounding, as applied in the cases with guaranteed associated half-integral minimum generalized cost flows, is not likely to be sufficient for good and practical solutions in this case. Even in the restricted setting, the applicability of the approximative solution (see Chapter 8) can possibly be extended to less restrictive (DP) instances.

Moreover, the challenging problem to find matchings with an edge-disjoint paths constraint, which was introduced in this thesis (see Chapter 9), can possibly be solved polynomially for more classes of constraint graphs and may serve as a model for other practical applications.

A IMPLEMENTATION & COMPUTATIONAL RESULTS

In the main part of this thesis, we described a classical and a generalized flow model for the homogeneous and heterogeneous (DP) respectively (Chapter 5 and 7). Further, we developed minimum cost flow algorithms, which are modifications of the *Successive Shortest Path* algorithm to solve those models in a combinatorial way. The major benefits of using combinatorial algorithms here, are to allow the reoptimization approach presented in Chapter 6 and to increase robustness with respect to feasibility of partial solutions. We also developed a heuristic to obtain a feasible (DP) solution from the half-integral minimum cost solution of the generalized flow model for application instances of the (DP) (Chapter 8). Note that the combination of reoptimization and heuristic requires the maintenance of the original residual network before rounding. Otherwise the optimality criterion cannot be reestablished as described in Chapter 6.

We implemented our algorithms in the object oriented programming language C++ using data structures from the Boost Graph Library (BGL). The prototypical implementation FLOh is designed as the optimization core of a new integrated dispatching tool which is in a planning stage at DB Schenker Rail Deutschland AG.

We give a short overview over the main classes of the implementation and their functionality in Section A.1. We present computational results with respect to running time of optimization and reoptimization, as well as performance results for the rounding heuristic in Section A.2.

A.1 IMPLEMENTATION OVERVIEW

The FLOh implementation is command line based and creates a set of output files in comma separated value structure (.csv-files or 'flat' files, as they avoid database management overhead). The latter consists either only of the relevant solution to the given (DP) instance I or additionally of control files. This can be influenced by command line parameters, which also enable a selection between three types of run behaviour.

FLOh offers a 'one-optimization' run in which master and input data of an instance I is read, a solution is computed and the program terminates afterwards. In the 'one-optimization-and-consolidation-run', FLOh waits for a trigger (appearance of a special file on the system) after solving I and generating appropriate output. As soon as the file appears, changes in the input data are read and the solution is reoptimized once, before

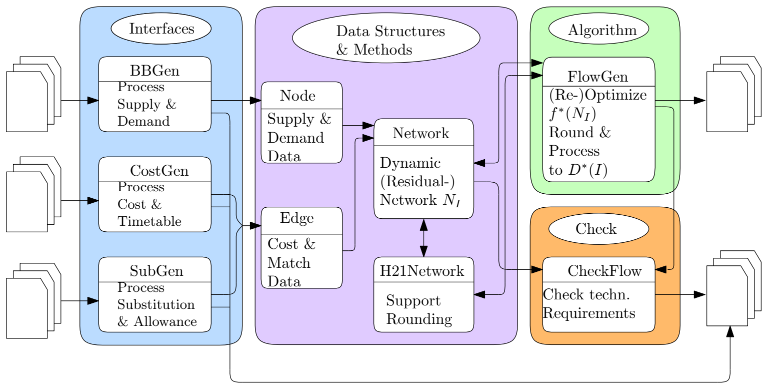


Figure 34: Overview on the main classes of the FLOh implementation.

FLOh terminates. This execution behaviour simulates an optimization run in practice during which manual interference occurs and potentially renders the solution infeasible. In this case the solution needs to be consolidated with the manual changes. The third run option for FLOh is a 'one-optimization-and-reoptimization' state, which is not terminated by FLOh until triggered from the file system. In this state FLOh initially solves a (DP) instance I and constantly reoptimizes the solution, if triggered changes to the input data appear. Note that each reoptimization is again followed by a consolidation.

Other command line parameters switch the usage of weak and strong priorities on or off, influence the treatment of border stations and limit the number of supplies and demands or the accepted range of their attributes for test purposes. The rounding heuristic can be switched off for homogeneous instances (which for example occur when FLOh is applied to separate instances for the main car type groups in a preliminary stage, see Chapter 2). In case of heterogeneous instances, different cost functions can be selected as they sometimes depend on car type valencies in practice.

The created classes can be grouped into *interface*, *data structure*, *algorithm* and *check* classes. See Figure 34 for an overview on the main classes.

A.1.1 *Interfaces*

Interface classes like BBGen, CostGen and SubGen read the input and master data of a DP instance I from 20 flat files with a comma separated value structure. The format was chosen to make the implementation independent of data base structures, which are not yet fully designed. BBGen creates objects of the Node class representing the attribute data of supplies $s_i \in S$, demands $d_j \in D$, operative storage $o_k \in O$ and border stations $b_l \in B$ from the actual input data of different files.

The classes CostGen and SubGen read rule based master data. CostGen internally creates a timetable $\mathcal{T}$ and provides methods to check the in-time relation for all types of Node objects. The class SubGen provides methods to check the different allowance relations for pairs of Node objects based on an internal representation of $\mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{B}$.

An interesting consideration by DB Schenker Rail Deutschland AG is currently to unify all type related attributes of supplies and demands into a bit string representation, such that bit-wise Boolean operations on the strings suffice to check the allowance relation for supply and demand pairs. The same method can be extended to storage and border stations. As this is not yet realized in practice, FLOh prepares the change of input data by internally precomputing the bit string representation and operating the methods for allowance checks on them.

As mentioned in Chapter 2, domestic rules $\mathcal{I}$ are so far based on geographical information. To limit time and space requirements, we clustered the productive area into regions with the help of DB Schenker Rail Deutschland AG. For the check of $\imath$-allowance supply locations and target locations of demands are mapped to regions, which form the basis of explicit rules $\iota \in \mathcal{I}$.

A.1.2 *Data Structures & Algorithm*

The classes Node and Edge are a struct-like aggregations of either input data attributes or attributes needed for the flow computation and configure the template based node and edge constructs of the BGL.

The central data structure for the network model N_I is the class Network, which also provides methods for network construction and changes which allow the representation of the residual network during the flow computation (optimization) as well as during the reoptimization.

The class H21Network provides an auxiliary network used for the rounding heuristic along with the rounding methods themselves. As mentioned above, it is necessary to maintain the residual network corresponding to the half-integral minimum cost flow in the Network instance. Therefore the strictly half-integral flow on transit arcs is represented by an instance of H21Network and rounded there.

The class FlowGen implements the modified variants of the *Successive Shortest Path* algorithm for optimization, reoptimization and rounding in a high-level way, which is supported by the Network and the H21Network respectively.

In the case of heterogeneous instances I the supplies and demands corresponding to assignments of the modified, rounded solution $\mathcal{D}'(I)$ (without spare cars of type t_x) are removed from the Network instance via reoptimization. We directly obtain the solution $\mathcal{D}(I')$ of the reduced and now homogeneous instance I' in this way, as the flow solution in the classical N_I' is integral.

FlowGen then creates the relevant output files by combining $\mathcal{D}'(I)$ and $\mathcal{D}(I')$. With the help of $\mathcal{D}'(I)$ and reoptimization, we can reconstruct the residual network corresponding to a minimum cost flow of N_I in the Network instance. Thus reoptimization due to changes of the input data of I can be done without constantly increasing the heuristic error.

FlowGen also creates important information on supply, which was not distributed and undersatisfied demands. Strictly speaking, a supply $s_i \in S$ which cannot be allocated makes it impossible to find a feasible solution for I with the definitions in Chapter 4. For our theoretical consideration such supplies can be removed from I via a precomputation of a maximum flow on N_I. In practice, such situations may occur and must be recognized, as they indicate serious undercapacities (in operative storage) or inconsistency of the master data.

A.1.3 *Checks*

The class CheckFlow provides about 15 methods, which check the (DP) solution against the given technical requirements. Apart from the inherent optimality criterion with respect to the flow computation, the results of these checks enable control of correct processing and internal interpretation of input and master data for a (DP) instance.

The formal correctness of input and master data is checked via class Parser during the reading process within the interface classes. Moreover, acceptance or rejection of data records can be influenced by certain command line parameters of FLOh as mentioned above. With the appropriate parameter setting, an output file is created for all accepted data records as well as all rejected data records with respect to a single input file for debugging purposes.

A.2 COMPUTATIONAL RESULTS

In this section we present computational results and running times for the FLOh implementation. We start with single optimization runs of homogeneous instances of the (DP) in Section A.2.1. In Section A.2.2 we present running times for the heuristic solution of heterogeneous instances of the (DP). We also analyze the cost of the resulting distribution with respect to the cost of an optimal half-integral solution as a lower bound on the optimal solution.

In Section A.2.3 running times for the solution of two consecutive (DP) instances are compared when both are solved 'from scratch' as opposed to the application of reoptimization to the solution of the first instance to solve the second instance.

All tests were carried out sequentially on an Intel Xeon CPU E5410 at 2.33 GHz with 4 GB RAM running Debian Linux.

A.2.1 *Optimization on homogeneous Instances*

We start by a description of the instances we used as a test setting. DB Schenker Rail Deutschland AG provided supply and demand data of one calendar week comprising of more than $10,000$ supplies and about $6,000$ demands, each for a single up to tens of cars. Usually, around $2,000-3,000$ supplies (the number of actual available cars scales roughly by a factor of 10) are available per day. About the same number of demands are considered, scattered over a time horizon of about two and a half days. The daily dispatching thus assigns about $25,000$ cars. To keep typical supply-demand-structures in the considered instances, we did not draw supplies and demands from the data set randomly.

Instead, from lists of supplies and demands sorted by availability and demand time respectively, we subsequently enlarge the supply and demand sets for our instances in steps of $1,000$, such that we assign cars with a time horizon between one up to five dispatching days. Additionally we take operational storage into account as a kind of low priority demand, which is a new feature in our automated dispatching. We consider storage capacities for up to $10,000$ cars at about 100 different locations, such that (almost) all supply can be assigned in each run. (Otherwise, cars may simply not be allocated in the repositioning process, but in practice each such car has to be stored somewhere nevertheless.)

Table 2 provides an overview over the structure of considered instances and solution times: the second and fourth column contain the number of supplies and demands respectively. Columns 3 and 5 contain the total number of available and ordered cars respectively. (For instances I_6 to I_{10} we use the maximum number of demand data records.) Note that we interpreted all substitution relations as 1-to-1 substitution to obtain homogeneous instances for our optimization tests. Column 6 shows the time (in minutes) consumed by the minimum cost flow computation and the generation of the resulting solution. Running times are measured without the time needed to read the input and master data (less than three seconds in each case, which is neglectable) and network construction time (around two minutes maximum for the largest instance I_{10}).

The righmost column contains the number of augmentations in the flow computation. In Table 2 (as in all our tests) their number was bounded by the total number of nodes in the network. Hence the empirical runtime bound is polynomial in the number of nodes in our setting, although we implemented

Table 2: Test instances on real data (homogeneous interpretation). Columns $|S|$ and $|D|$ contain the number of supply and demand nodes respectively, $\sum n_i$ ($\sum n_j$) is the total number of available (ordered) cars. Running time is measured in minutes and the last column contains the number of augmentations in the flow computation.

I	$\|S\|$ $[10^3]$	$\sum n_i$ $[10^4]$	$\|D\|$ $[10^3]$	$\sum n_j$ $[10^4]$	time [min.]	augm.
I_2	2	1.7789	2	1.9900	0.32	2575
I_3	3	2.6243	3	3.0116	1.02	4503
I_4	4	3.1439	4	4.0095	1.97	6284
I_5	5	4.0913	5	4.9657	3.76	8511
I_6	6	4.9427	5.715	6.2466	5.48	10150
I_7	7	5.4926	5.715	6.2466	6.90	11404
I_8	8	6.3655	5.715	6.2466	9.05	13153
I_9	9	7.2130	5.715	6.2466	11.92	14886
I_{10}	10	7.8308	5.715	6.2466	13.51	14885

the pseudo polynomial *Successive Shortest Path* algorithm without scaling. We decided that the benefit of applying a scaling technique to obtain a polynomial version of the *Successive Shortest Path* algorithm was outweighed by the computational overhead in our case.

A.2.2 *Optimization on heterogeneous Instances*

In the following, we present examples for the heuristic solution of heterogeneous (DP) instances as described in Chapter 8. We again present running times and also empirical approximation ratios between the cost of the heuristic solution and the cost of a half-integral optimal flow as a lower bound on the minimum cost.

Table 3 shows additional information on the instances we interpreted as homogeneous in Table 2. Column 2 contains the total number of available cars involved in heterogeneous substitution and their percentage with respect to the total supply (in brackets). Column 3 contains analogous data for demands. The last column displays the percentage of cars involved in heterogeneous substitution with respect to the sum of total supply and demand. These values show that instances in the application are usually almost homogeneous. Yet, for the practical performance the integrated view is important, for example with respect to mixed type storage capacities.

Table 4 shows running times in minutes for the computation of the half-integral flow in column 'time ($\mathcal{D}$)'. The time for the rounding and

Table 3: Heterogeneity of (DP) instances on real data sets. Column 2 displays the number of available cars involved in heterogeneous substitution and their percentage of the total supply (in brackets). Column 3 shows analogous data for demanded cars and the rightmost column contains the percentage of all such cars with respect to the sum of total supply and demand.

I	$\sum n_i$, het. (%)	$\sum n_j$, het. (%)	het. total %
I_2	327 (1.84)	916 (4.6)	3.3
I_3	670 (2.55)	1,313 (4.36)	3.52
I_4	670 (2.13)	1,815 (4.53)	3.47
I_5	935 (2.29)	2,105 (4.24)	3.36
I_6	1,285 (2.6)	2,777 (4.45)	3.63
I_7	1,285 (2.34)	2,777 (4.45)	3.46
I_8	1,476 (2.32)	2,777 (4.45)	3.37
I_9	1,835 (2.54)	2,777 (4.45)	3.43
I_{10}	1,835 (2.34)	2,777 (4.45)	3.28

reassignment is displayed in column 'time ($\mathcal{D}_h$)' and their sum in column 'time'. Running times are again measured without input time (less than three seconds in each case) and network construction time (around two minutes maximum).

Table 5 presents the total costs of the optimal half-integral flow in N_I^g in column 'cost ($\mathcal{D}$)' and the cost of the heuristic solution in column 'cost ($\mathcal{D}_h$)'. We show the difference 'cost ($\mathcal{D}_h$) - cost ($\mathcal{D}$)' of both costs in column 4 and its percentage with respect to cost ($\mathcal{D}$) (in brackets). As remarked above, the latter cannot be seen as pure cost increase on cost ($\mathcal{D}$), as the cost of an optimal *feasible* solution can exceed this term by far. As a quality measure for the heuristic, the rightmost column displays the value $\frac{\text{cost}(\mathcal{D}_h)}{\text{cost}(\mathcal{D})}$, which we call the 'empirical approximation factor', with 'cost ($\mathcal{D}$)' as a lower bound on the cost of an optimal distribution.

A.2.3 *Reoptimization*

For the reoptimization tests, we generally compare the time to solve two instances of the (DP) separately (or 'from scratch') with the time to solve the first instance in this way and reoptimize its solution for the second instance. We choose the first (or basic) instance identical for all test cases and increase the number of changes to the second instance subsequently.

The basic instance I_B is a middle size instance of about 5000 supply and demand nodes (comparable, but not identical to instance I_5), which corresponds to a real application instance with a time horizon of about two

Table 4: Running times in minutes for obtaining an optimal half-integral solution (column 'time ($\mathcal{D}$)'), rounding and redistribution (column 'time ($\mathcal{D}_h$)') and the total time to obtain a feasible heuristic solution (column 'time total').

I	time ($\mathcal{D}$) [min.]	time ($\mathcal{D}_h$) [min.]	time total [min.]
I_2	0.32	0.08	0.40
I_3	1.02	0.12	1.14
I_4	1.97	0.13	2.1
I_5	3.76	0.17	3.93
I_6	5.48	0.35	5.83
I_7	6.90	0.72	7.62
I_8	9.05	1.45	10.50
I_9	11.92	3.12	15.04
I_{10}	13.51	6.87	20.38

Table 5: Costs for the optimal half-integral 'solution' $\mathcal{D}$ and the feasible heuristic distribution $\mathcal{D}_h$. Their difference 'cost ($\mathcal{D}_h$) - cost ($\mathcal{D}$)' and its percentage with respect to 'cost ($\mathcal{D}$)' is shown in column 4 and 5. The righmost column shows the 'empirical approximation factor'.

I	cost ($\mathcal{D}$) [10^7]	cost ($\mathcal{D}_h$) [10^7]	diff. [10^7]	(%)	$\frac{\text{cost}(\mathcal{D}_h)}{\text{cost}(\mathcal{D})}$
I_2	1.09	1.11	0.02	(1.06)	1.01
I_3	1.32	1.35	0.03	(1.77)	1.02
I_4	1.43	1.46	0.03	(2.05)	1.02
I_5	1.84	1.87	0.03	(1.81)	1.02
I_6	2.05	2.09	0.04	(1.69)	1.02
I_7	2.40	2.43	0.03	(1.45)	1.01
I_8	3.04	3.07	0.03	(1.17)	1.01
I_9	3.47	3.60	0.13	(3.81)	1.04
I_0	3.50	4.01	0.51	(14.42)	1.14

and a half days. The time to solve this instance by a single optimization run including data input and network construction is 3.6 minutes. Here we include input and especially construction time for fairness reasons: the local network (de)construction due to data changes is an integrated part of the reoptimization and is thus also included in the reoptimization run time.

We further generate instances from I_B with an increasing number of changes $\delta \in \{100, 200, 400, 800, 1000, 2000, 2500, 5000\}$.

Remember from Chapter 6, that we distinguish changes in addition, removal and change in the number of cars for either supply or demand. For each number δ of changes, we generated 40 % additions and 40 % removals distributed equally among supplies and demands. Addition of supplies and demands usually occurs on account of the proceeding time line. Removal of supplies and demands usually occurs when assignments are due and drop out of the optimization. In some cases also unforseen supplies come up or cars prove defective and demands can be cancelled short-termed by the customer.

The remaining 20 % of δ are changes in the number of cars, also equally distributed between supplies and demands, which occur due to partly allocated supplies and partly satisfied demands. Changes in the number of cars are also necessary data corrections for the above mentioned reasons (unforeseen/defective/canceled cars).

Let I_δ denote the instance resulting from δ changes in I_B. Table 6 compares the running times for solving I_δ from scratch (column 'optimization') with the time (column 'reoptimization') needed to reoptimize the solution of I_B to fit the δ changes indicated in the first column. We also display the difference between both in column 'diff.' and its percentage with respect to the optimization approach. All times are given in minutes and contain input and network generation time.

RUNNING TIME AND ROBUSTNESS We see that for up to 20% changes ($\delta = 2000$) on the total data volume of the basic instance the reoptimization approach saves running time compared to optimization from scratch. For changes on less than 10% ($\delta \in \{100, 200, 400\}$) of the nodes in the network, the reoptimization time is only about 10% of the time for a complete optimization run on the changed instances I_δ. In these cases it can be assumed that the number of changes between two (DP) instances is also a measure for the similarity of their solution. Such similarities are welcome in practice, as they indicate robustness of a solution.

While for changes on 10% ($\delta = 1000$) of the nodes we still save about 40% running time with the reoptimization compared to a solution from scratch, the benefit drops to 4% when 20% of the network's nodes are subject to changes ($\delta = 2000$). Finally, if about one fourth of the network is changed in one or another way, reoptimization performs worse than an optimization

Table 6: Runtime comparison for δ mixed type data changes via optimization and reoptimization approach. Note that for the instances with 2500 and 3000 changes respectively the reoptimization approach yields worse running times than the optimization approach, which results in a negative difference (column 'diff.').

changes	optimization time [min.]	reoptimization time [min.]	diff. [min.]	(%)
100	3.58	0.12	3.46	96.75
200	3.62	0.23	3.39	93.55
400	3.43	0.38	3.05	88.83
800	3.19	0.9	2.29	71.82
1000	3.22	1.27	1.95	60.64
2000	3.26	3.1	0.16	4.85
2500	3.34	4.07	-0.73	-
3000	3.33	5.13	-1.8	-

from scratch. One reason is that for data changes in the dimension of 25% of the original input data, similarity of the solution for I_B and I_δ is unlikely.

REOPTIMIZATION AND BENEFIT LIMITS It would be nice to ensure that we always benefit from reoptimization in that we need at most the same running time as for the optimization in any case. However, this may not be possible for the following reason. The reoptimization needs to be carried out separately for addition (increase of the number of cars) and removal (decrease of the number of cars) as well as for supplies and demands (cf. Chapter 6). In case of addition for example supply and demand nodes are treated separately to ensure that we find a valid potential.

Hence, if matching supplies and demands are added such that the former is assigned to the latter in an optimal solution, the reoptimization needs at least two augmentations to find the optimal distribution. This happens, because either the supply or the demand is not present in the network until the current solution is repaired with respect to the other. On the other hand, in the optimization, both supply and demand are present from the very beginning, such that a shortest path between them can be found within one augmentation. Analogous situations can occur for other combinations of changes on supplies and demands.

Further, an implementation of the reoptimization is always likely to produce a slight data management overhead with respect to pure optimization.

Table 7: Runtime comparison: Optimization versus reoptimization with additional supply and demand.

changes	optimization time [min.]	reoptimization time [min.]	diff. [min.]	(%)
40	3.62	0.03	3.59	99.08
80	4.24	0.07	4.17	98.43
160	3.56	0.2	3.36	94.38
320	3.63	0.25	3.38	93.1
400	3.45	0.38	3.07	88.89
800	3.88	0.97	2.91	75.09
1000	4.04	1.27	2.77	68.61
1200	4.03	0.98	3.05	75.57

REOPTIMIZATION BENEFIT IN PRACTICE In practice, reoptimization is mainly used as a consolidation to guarantee a quasi non-blocking work-flow: during the main optimization, which takes a few minutes, manual changes can be performed as usual. Those (few) changes have to be integrated into the solution very fast, such that a blocking of manual data entry does not disturb the general work flow.

Moreover, a fast reoptimization can be applied frequently enough to prevent massive data changes in between two runs. Hence, a change of one fourth of the whole (DP) data between two runs is unlikely and indicates that a considerable time interval has passed. In the latter case, an optimization from scratch is the better choice anyway: we limit the possible changes during the reoptimization to node changes, which generally correspond to changes of the input data. With an increasing time interval it becomes more likely that also master data such as the timetable has to be adapted.

REOPTIMIZATION FOR SEPARATE TYPES OF CHANGES After this general comparison of optimization and reoptimization with respect to running time, we also look at the runtime behaviour when only a certain type of change occurs. Tables 7 and 8 are structured as Table 6, but the data changes are limited to addition and removal respectively. We use the same sets of changes δ and ignore changes different from addition or removal respectively, such that we have $\delta \in \{40, 80, 160, 320, 400, 800, 1000, 1200\}$ for each kind of change.

On the one hand, we see from Tables 7 and 8 that addition of supply or demand seems much more well-behaved with respect to the reoptimization approach than removal. On the one hand obviously I_δ is simply smaller in terms of $|S|$ and $|D|$ than I_B, if all changes are removals. Hence, the optimiza-

Table 8: Runtime comparison: Optimization versus reoptimization with removal of supply and demand.

changes	optimization time [min.]	reoptimization time [min.]	diff. [min.]	(%)
40	3.35	0.07	3.28	98.01
80	3.56	0.13	3.42	96.25
160	3.54	0.27	3.27	92.46
320	2.98	0.5	2.48	83.22
400	2.96	0.7	2.26	76.35
800	2.71	1.53	1.18	43.51
1000	2.52	1.88	0.64	25.33
1200	2.44	2.28	0.16	6.3

tion on I_δ can be expected to be faster. Further, all supply is distributed in an optimal solution and demands are usually not completely unsatisfied. Thus a removal always results in an actual change of the solution, whereas for example for additional supply potentially only additional assignments are necessary without changing the assignments in the former distribution. This is likely in our test setting, as additional supply is created by duplicating existent supplies. However, these observations only partly explain our results, as also pure addition and pure removal can cause necessary changes in the optimal solution, especially if supplies and demands are both subject to the changes.

As mentioned above, removals usually occur in practice when assignments are due and the corresponding supplies and demands need to be extracted from the (DP) instance. This states a special case of removal. Consider an assignment $\delta_{ij} = (i, j, n_{ij})$ of n_{ij} cars of supply s_i to demand d_j. Further let no heterogeneous substitution be involved with respect to s_i and d_j and let δ_{ij} be part of the optimal distribution, which means it is no manual assignment differing from the optimization outcome.

Then the minimum cost flow in the network N_I^g contains a flow greater or equal to n_{ij} from s via i and j to one of the sinks t. Thus we can reduce the capacities of the arcs (s, i) and (j, t) by n_{ij} together with the flow on these arcs and $f(i, j)$. (We also delete arcs if their capacity is reduced to zero and remove the nodes i and j if they have no more incoming or outgoing arc respectively.) The resulting flow remains optimal for the resulting network as well as the solution remains optimal, if the single assignment δ_{ij} and the corresponding number of available and ordered cars are removed (see Chapter 6). Hence, in this practically most relevant case of removal, a reoptimization as described in Section 6.2 is not even necessary. Consequently our test results for removals do not limit the practical applicability of the reoptimization approach.

Not surprisingly changes in the number of cars for supplies and demands exhibit a runtime behaviour between those of addition and removal, as we generated increases as well as decreases.

A.2.4 *Conclusion*

Our tests show that the model approach as well as the prototypical implementation meets the application's requirements. Furthermore, the reoptimization is beneficial with respect to speeding up subsequent computations for up to 40% changes in the input data and yields a quasi non-blocking work flow.

B SPECIAL GENERALIZED FLOW NETWORKS

In this appendix we summarize some results on complexity and bounded fractionality of generalized flow problems on special network instances. These results are obtained independently of the (DP).

B.1 EQUIVALENCE OF ARC AND NODE MULTIPLIERS

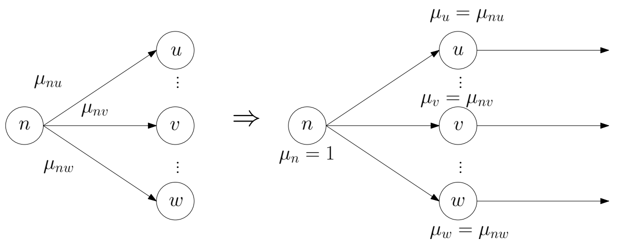


Figure 35: Multipliers on arcs and on nodes are equivalent.

The original NP-completeness proof for generalized flows by Sahni [107] employs the subset sum problem and a variant of generalized flows, where multipliers are attached to nodes instead of arcs. Both variants of flow multipliers are equivalent in the following sense:

A node multiplier μ_i on v_i can be interpreted as an arc multiplier μ_{ij} on all outgoing arcs (v_i, v_j) of v_i. On the other hand, if all outgoing arcs of one node have the same multiplier μ_{ij}, the node multiplier also equals this value. For a number $\#\mu_{ij}$ of different arc multipliers on outgoing arcs of v_i, we introduce $\#\mu_{ij}$ new nodes v_{ik}, $1 \leqslant k \leqslant \#\mu_{ij}$, add the arcs (v_i, v_{ik}) and set $\mu_i = 1$. The originally outgoing arcs of (v_i, x) are now changed to arcs (v_{ik}, x) and the values μ_{ik} are set according to the desired arc multipliers (see Figure 35). In the worst case, $m = |A|$ nodes and edges are added, but the necessary modifications are thus polynomial.

B.2 COMPLEXITY OF GENERALIZED FLOW IN SPECIAL NETWORKS

For the convenience of the reader, we first recapitulate the original proof of [107] for the NP-completeness of the generalized maximum flow problem. It provides a reduction of the *subset sum* problem, which is defined as follows:

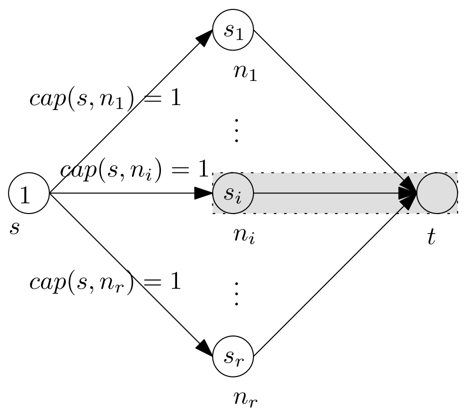


Figure 36: Original reduction of subset sum: Node multiplier values s_i in the nodes n_i, all missing capacities are unlimited and the node balance at t is $b(t) = -M$.

[SubsetSum] Given a multiset $S = \{s_1, \dots, s_r\}$ of positive integers and a positive integer M, ($M \leqslant \sum_{i=1}^{r} s_i$), does there exist a sub multiset of S that sums up to M?

For general positive integer values, the subset sum problem is NP-complete. For a given subset sum instance S, construct a network N_S with a source node s, nodes $n_i, 1 \leqslant i \leqslant r$ with node multiplier s_i for every integer $s_i \in S$ and a sink node t with a demand of $b(t) = -M$. For every i the source is connected to n_i by (s, n_i) with capacity 1. The nodes n_i are connected to t by arcs (n_i, t) with unlimited capacity (see Figure 36). There is an integral s-t-flow solution if and only if there is a sub multiset of S that sums up to M. (Concerning the appropriate node balance $b(s) = r$ and the node balance constraints, we have to add another sink t' with $b(t') = -(\sum_{s_i \in S} s_i - M)$ and the arcs (n_i, t') to Sahni's construction to guarantee a feasible flow solution.)

We adopt the complexity result for generalized networks with multiplicator function $\mu : A \to \{1, 2\}$. We denote such networks by $N_{1,2}$. With all multipliers restricted to 1 or 2 and the above network construction, we can only represent subset sum instances where $s_i \in \{1, 2\}$, which does not suffice for the reduction since those instances are easy to solve. Therefore, in the network N_S we replace each arc (n_i, t) (grey shaded box in Figure 36) by a sub graph $N_S|n_i$ like the one shown in Figure 37. The purpose of the sub graph is to amplify the one unit of flow which is the upper capacity bound on arc (s, n_i) to the appropriate s_i flow units which have to arrive at t for the original unit. Further we want to use only node multipliers 1 and 2 in the sub graph. We will see that producing the appropriate number of flow units resembles the flow equivalent of a bit representation of each s_i.

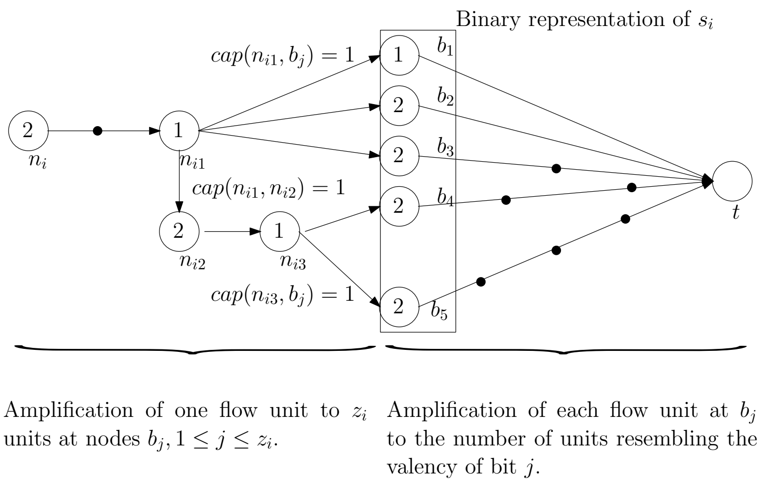

Binary representation of s_i
$cap(n_{i1}, b_j) = 1$
$cap(n_{i1}, n_{i2}) = 1$
$cap(n_{i3}, b_j) = 1$
n_i
n_{i1}
n_{i2}
n_{i3}
b_1
b_2
b_3
b_4
b_5
t
Amplification of one flow unit to z_i units at nodes $b_j, 1 \leq j \leq z_i$.
Amplification of each flow unit at b_j to the number of units resembling the valency of bit j.

Figure 37: Subgraph $N_S|n_i$ for $s_i = 31$: Node multipliers are written in the nodes and all black dots are nodes with multiplier 2. Missing capacities are unlimited.

The sub graph $N_S|n_i$ again consists of two parts: Given s_i, we first see how many bits we need to represent the value. The number of bits z_i we need here is not the length of the binary representation $|(s_i)_2|$, but the number of set bits in it, e.g.:

$$s_i = 31 = (11111)_2 : z_i = 5 \text{ or } s_j = 27 = (11011)_2 : z_j = 4$$

For each set bit in s_i's binary representation, we need a node $b_j, 1 \leqslant j \leqslant z_i$ in $N_S|n_i$ such that one unit of flow enters b_j if and only if one unit of flow enters n_i. Let $p(b_j)$ be the valency, i.e. the position of bit b_j in the bit representation. Then we add a path $\pi_{b_j t}$ of length $p(b_j) - 1$ to $N_S|n_i$ where all node multipliers are set to 2. If $p(b_j) - 1 = 0$, we add the arc (b_j, t) and set the node multiplier of b_j to 1. Now the sum of flow units entering t from all b_j equals s_i.

Further we have to connect the b_j to s: Recall that there is the arc (s, n_i) with capacity 1 and that s has the node multiplier 1. Let l with $2^l < z_i$ be maximal. Then we add an amplification path $\pi_{S n_{i1}}$ of length l and with node multipliers 2 on every node, except for n_{i1}, which has node multiplier 1. For one unit of flow, starting on s there are now 2^l units arriving at n_{i1}. We connect n_{i1} to the nodes $b_j, 1 \leqslant j \leqslant 2^l - 1$ by arcs (n_{i1}, b_j) with capacity 1, such that the first $2^l - 1$ nodes b_j receive exactly one unit of flow if and only if one unit of flow starts from s to n_i.

Then we add two extra nodes n_{i2}, n_{i3} with node multipliers 2 and 1 and the arcs (n_{i1}, n_{i2}) and (n_{i2}, n_{i3}) with capacity 1. For the one unit left at n_{i1}, we now have two units at n_{i3}. We add the arc (n_{i3}, b_{2l}) to cover b_{2l} with one of the two remaining units. Further we add n_{i4} and arc (n_{i3}, n_{i4}) with capacity 1. Now at node n_{i4} we have the same situation as before at node

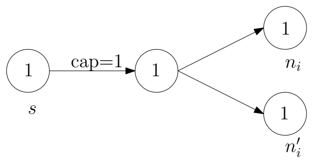


Figure 38: Forking Sub graph : Node multipliers are written in the nodes and missing capacities are unlimited.

n_{i1}, only that we have to generate $z_i - 2^l$ units of flow and distribute them appropriately to the remaining nodes $b_j, 2^l + 1 \leqslant j \leqslant z_i$. This can be done exactly as before, determining the maximum l' with $2^{l'} < z_i - 2^l$ and adding the amplification path, some new nodes and the arcs with capacity 1 to the bit nodes b_j.

The procedure terminates, because each number z_i has a binary representation. Further, this representation is bounded in length by $\lceil \log_2(z_i) \rceil$, thus we have to repeat the above procedure at most $\lceil \log_2(z_i) \rceil$ times and each amplification path itself is at most $\lceil \log_2(z_i) \rceil$ nodes long. The same is true for the number z_i of bit nodes b_j and the paths from b_j to t. Thus the construction of each $N_S|n_i$ is polynomial.

To ensure a feasible flow if and only if there is a subset $S' \subseteq S$ with $\sum_{s_i \in S'} s_i = M$, we duplicate each sub graph $N_S|n_i$ as $N_S|n_i'$ and add another sink t' which substitutes t for the duplicate sub graphs. Then we delete each arc (s, n_i) and add the sub graph depicted in Figure 38, where s, n_i, n_i' are corresponding nodes which already belong to N. Thus with $b(t) = M$ and $b(t') = \sum_{s_i \in S} s_i - M$ we obtain feasible flows in all desired cases.

Now we can set $b(s) = r$. Thus for a valid integral flow solution r units leave s and result in s_i units entering t or t'.

Theorem 92 *It is NP-complete to decide whether a generalized network* $N_{1,2}$ *allows for a feasible integral flow.*

Proof: Given an instance $S = \{s_i, 1 \leqslant i \leqslant r\}, M$ of subset sum, we construct a network as described above. Then for each unit of flow leaving s to a node n_i or n_i', exactly s_i units of flow enter t or t'. Thus, the sinks t, t' with $b(t) = -M, b(t') = -(\sum_{i=1}^{r} s_i - M)$ can be balanced by an integral flow from s if and only if there is a subset $S' \subseteq S$ with $\sum_{s_i \in S'} s_i = M$ (and simultaneously $\sum_{s_i \in S \setminus S'} s_i = \sum_{s_i \in S} s_i - M$). Moreover the problem is in NP: An NTM guesses the flows and verifies the balance and capacity constraints.
$\square$

We further restrict the structure of our generalized networks and investigate the complexity of the decision if feasible integral flows exists. Let $N = N_{1,2}^r$ be a bipartite network $(V = V_s \cup V_t, A)$ with one source $s \in V_t$

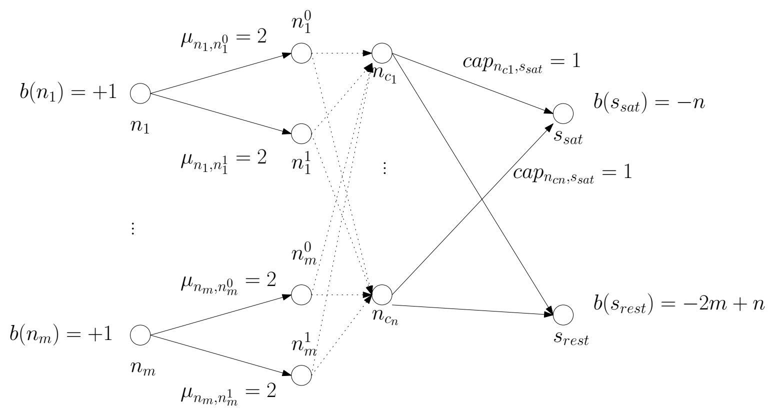


Figure 39: Construction of N_α.

and multiple sinks $t_i \in V_s$. Arcs are only directed from s to V_s, V_s to V_t and V_t to t_i. The multiplicator function $\mu : A \to \{1, 2\}$ is 1 on all arcs (s, v), (w, t_i) and such that all outgoing arcs (v, w) of a node $v \in V_s$ have the same multiplier $\mu(v, w)$. We call such networks 2-regular networks. The latter do not include the network N_S we used in Theorem 92 and we present an alternative reduction of the 3V2L3SAT problem (see Section 4.2).

For a given instance α of 3V2L3SAT, we construct the following network N_α: We have vertices n_k for every variable v_k and two additional vertices n_k^0, n_k^1 for the corresponding literals $\neg l_k, l_k$, which occur in α. For every n_k the node balance b_k is 1 and the node balance of the literal nodes is 0. Nodes n_k are connected to the corresponding literal nodes by arcs (n_k, n_k^0), (n_k, n_k^1) with an arc multiplier of 2.

For each clause C_i in α we need another vertex n_i, which has incoming arcs (n_k^*, n_i) from those literal nodes which correspond to literals occurring in C_i. Finally we add two sink nodes s_{sat}, s_{rest} with node balances $b_{sat} = -n$ and $b_{rest} = -2m + n$ to the graph. We connect the clause nodes to both sinks by arcs (n_i, s_{sat}) with capacity 1 and arcs (n_i, s_{rest}) with unlimited capacity (see Figure 39). Unless otherwise noted, any node n_j has balance $b_j = 0$ and any arc e_{ij} has multiplier $\mu_{ij} = 1$, unlimited capacity and zero costs. Note that the constructed network meets all above restrictions on $N_{1,2}^r$.

Theorem 93 *It is NP-complete to decide whether a generalized network* $N_{1,2}^r$ *allows for a feasible integral flow.*

Proof: Given a 3V2L3SAT formula α, construct $N_{1,2}^r = N_\alpha$ as above. For a valid solution, one unit of flow has to leave each variable node n_k **either** to n_k^0 **or** to n_k^1 as the flow has to be integral. Now we have 2 units of flow at one half of all literal nodes which sum up to $2m$ units of excess exactly as the sum of deficits at the sinks. Thus there is a valid flow if there are paths to the sinks with appropriate capacity.

As the literal nodes only occur in the network if the corresponding literal occurs in α, each such node is connected to at least one clause node n_i, which is again connected to both sinks. As the arcs (n_i, s_{rest}) have no capacity limits, the required flow units can always pass from arbitrary literal nodes to s_{rest}. On the other hand, s_{sat} can be reached by only one unit of flow via each clause node. To balance the deficit $b_{s_{sat}} = -n$ and at the same time satisfy all node and capacity constraints, at least one unit of flow must pass each clause node.

We can thus reinterpret the flow through a literal node n_k^0 resp. n_k^1 as a truth assignment ν with $\nu(v_k) = 0$ resp. $\nu(v_k) = 1$. Each unit of flow through a clause node n_i must then correspond to a true literal in the corresponding clause. A valid flow, especially carrying n flow units to s_{sat}, therefore corresponds to a truth assignment satisfying at least one literal in every clause and thus satisfying the whole formula α.

If there is no valid integral flow, we cannot find a satisfying truth assignment either. Moreover the problem is in NP (Theorem 92). □

B.3 K-REGULAR NETWORKS AND K-FRACTIONALITY

We generalize our notion of 2-regular networks (see Section B.2) to k-regular networks by allowing a multiplicator function $\mu : A \to \{1, k\}$ and keep all other restrictions as before. For k-regular networks $N_{1,k}^r$, we obtain a k-fractional feasible flows in polynomial time, if it exists. We argue this along the lines of an iterative cycle cancelling algorithm, which maintains k-fractionality of the residual capacity on all arcs. This was the first proof that half-integral minimum cost flows $f(N_I^g)$ can be obtained in polynomial time for practical instances of the (DP).

We transform a k-regular network $N = N_{1,k}^r = (V_s \cup V_t, A)$ (see Figure 40 a)) into a generalized minimum cost flow circulation instance N' (see Figure 40 b)) as follows: We split node s into nodes s_1 and s_k and connect all nodes a_i with (s, a_i) and multiplier 1 to s_1, nodes a_i with (s, a_i) and multiplier k (remaining nodes of set V_s) to s_k while the arcs have the same cost and capacity values as before. Further, we add arcs (t, s_1) and (t, s_k) with cost zero, unlimited capacity and multipliers 1 and $\frac{1}{k}$, respectively. The minimum cost flow solution in N is equivalent to the minimum cost flow circulation solution on N' with the following property:

Lemma 94 *Given a feasible circulation f, every cycle in the residual network N_f' is a unit gain cycle, i.e. the product of the multipliers of the nodes in the cycle is 1.*

Proof: Denote by S_1 and S_k the set of nodes containing the source s_1 and s_k respectively and all nodes a_i connected to s_1 and s_k respectively. Let the set T contain all nodes b_j and t_i. By construction flow is generated

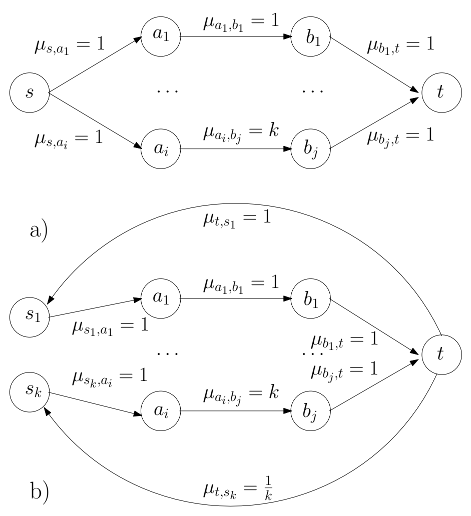


Figure 40: Simple example for a k-regular network N (a). The transformed network N′ is shown in (b).

(multiplied by k) only on arcs from S_k to T and destroyed (divided by k) only on arcs from T to S_k. As there are no arcs between nodes of S_1 and S_k or vice versa, each cycle has to contain as many arcs from S_k to T as from T to S_k and thus the product of all node multipliers along any cycle is 1. (Moreover, it follows, that every path in the residual network has a path multiplier of either $\frac{1}{k}$, 1 or k.) □

It is well known [55, 61] that a feasible generalized circulation f is optimal if and only if N'_f contains no negative cost circuits, where a circuit is a circulation that sends positive flow only along the arcs of a single residual unit-gain cycle (or bicycle, which cannot occur here due to the network structure). Further the optimal generalized circulation can be decomposed into unit-gain cycles.

With this observation, we base our argumentation for the result on the *Cycle Cancelling Algorithm* [83], which successively detects circuits with negative costs in the residual network and cancels them by augmenting the maximum possible amount of flow along them. (Thus at least one of the cycle's arcs is saturated. Due to the multipliers this is not necessarily the one with least residual capacity.) By Lemma 94, we can limit this argumentation to the cancelling of unit-gain cycles.

Theorem 95 *Cycle Cancelling always yields a* k*-fractional solution to the minimum cost circulation problem on* N′*.*

Proof: Let A_k comprise all arcs with tail in S_k, all other arcs are contained in A_1. We show that during the algorithm the residual capacity on all arcs from A_1 and A_k respectively remains integral and k-fractional respectively.

Clearly, this property holds for the zero circulation. If a cycle cancelling step increases the flow along an arc $a \in A_1$ by θ, it is increased along any arc $a' \in A_k$ along the same cycle by $\frac{1}{k}\theta$. Obviously, if the residual capacity u_a of a determines the maximal flow which can be augmented along the current negative cost cycle and u_a is not a multiple of k, the flow θ on arcs in A_1 is still integral, but the flow on arcs in A_k can become k-fractional. Such flows on arcs in A_k lead to k-fractional residual capacities, but those again lead only to k-fractional flows θ on arcs in the same set and to integral flows and residual capacities on arcs of A_1. □

The solution in N' can be transferred to the flow instance N. We can even enlarge the set of allowed multipliers to $\mu : A \to K \subset \mathbb{N}$ (K-regular networks) with the same restrictions as before. By introducing source nodes $s_k, k \in K$ and partitioning the arcs into sets $A_k, k \in K$, we obtain an analogous result as Theorem 95 for K-regular networks:

Corollary 96 *Let k^* be the least common multiple of the set K for a K-regular network N_K^r. Then we can obtain a k^*-fractionate minimum cost flow in N in polynomial time.*

B.4 K-REGULAR NETWORKS AND K-REGULARITY

Here we present an ILP formulation of the generalized minimum cost flow problem in k-regular networks and show that the resulting constraint matrix is k-regular. The latter is a generalization of total unimodularity by G. Appa [7, 8] and guarantees fractionate solutions of integer multiples of $\frac{1}{k}$ (k-fractional flows). First of all, we restate the definition of k-regularity:

Definition 97 (k-Regularity, [8]) *A rational matrix is called k–regular, if for each non-singular square submatrix R, kR^{-1} is integral.*

Consider the LP formulation of the minimum cost network flow problem in $N = N_{1,k}^r = (V, E)$, where A is composed of the node arc incidence matrix $M(N)$ and the identity matrix of dimension m attached below. Further, $\tilde{b}$ is the vector of node balances $b(i)$ with the vector of capacities $u(i,j)$ attached below (in the order of arcs corresponding to the columns of $M(N)$). Thus $Ax \leqslant \tilde{b}$ accounts for node balance and capacity constraints.

$$\begin{aligned} \text{maximize } c^T x \\ \text{s.t.} \\ Ax \leqslant \tilde{b} \\ x \geqslant 0 \end{aligned}$$

Let x^* be an optimal flow. Further, each LP solution attains the bounds of a subset of constraints (those inequalities, which are satisfied with equality). Thus the respective values x^* of the variables x are determined by a system of equalities. The latter must be of full rank and is thus a non-singular square submatrix R of A with $x^* = R^{-1}b$. In case A is k-regular, then also R is k-regular. By Definition 97, this means that kR^{-1} is integral or in other words that each entry of R^{-1} is a multiple of $\frac{1}{k}$. Thus any solution x^* has bounded fractionality of k.

In the following, we show that A is k-regular for a k-regular network with application of some useful lemmas of [8].

Lemma 98 (Invariants [8]) *Let A be k−regular. Then the following matrices are also k−regular:*

- *the transpose of A*
- *any submatrix of A*
- *the matrix obtained by multiplying a row or column of A by -1*
- *the matrix obtained by interchanging two rows or columns of A*
- *the matrix obtained by duplicating a row or column of A*
- *the matrix obtained by dividing a row or column of A by a non-zero integer*

Lemma 99 (Attachment of Identity [8]) *Let A be an integral matrix. Then A is k-regular if and only if $[A, I]$ is k-regular.*

The following Lemma reduces the task of proving the k-regularity of A to proving the k-regularity of $M(N)$.

Lemma 100 (A and $M(N)$) *A is k-regular if and only if $M(N)$ is k-regular.*

Proof: Observe that $A = [M(N), I]^T$. By Lemma 98 A is k-regular if and only if A^T is k-regular and by 99 A^T is k-regular if and only if $M(N)$ is k-regular and vice versa as transposition is symmetric. □

As a preparation, we analyze the structure of $M(N)$. Let $n = |V|$ and $m = |E|$. Further, we denote columns of $M(N)$ corresponding to the arcs (s, i) by $e_1, \dots, e_{|S|}$, the arcs (j, t_p) by $e_{m-|D|+1}, \dots, e_m$ and the arcs (i, j) by $e_{|S|+1}, \dots, e_{m-|D|}$. The rows of $M(N)$ are arranged such that the first row corresponds to s, rows 2 until $|S|$ correspond to nodes a_i, the following $|D|$ rows correspond to nodes b_j and the last rows correspond to the sink nodes t_p. Within these predefinitions, we can order rows corresponding to nodes and columns corresponding to arcs to obtain a general structure as depicted in Figure 41.

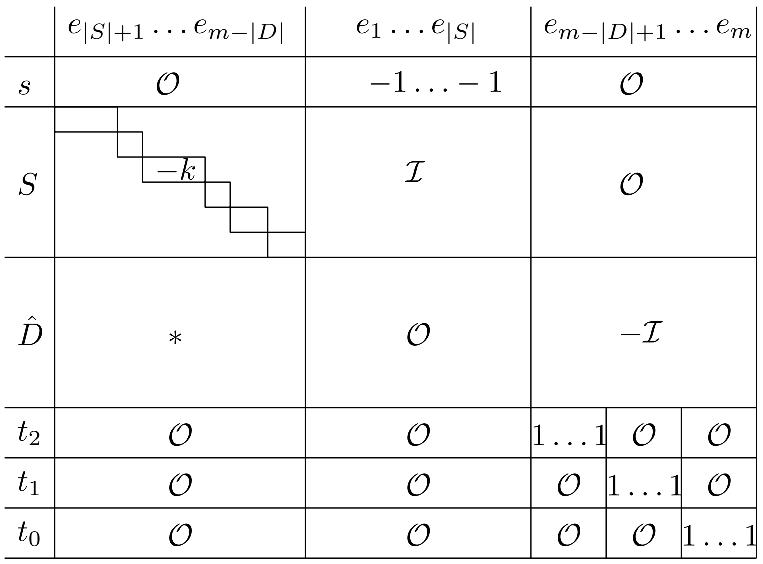


Figure 41: Structure of the arc node incidence matrix $M(N)$ of a k-regular network N.

Here $\mathcal{O}$ stands for the matrix with all entries zero, $\mathcal{I}$ is the identity matrix (of appropriate dimension) and in the area marked by an asterisk $*$, each column contains exactly one entry 1, all others are zero. From this structure obviously most square submatrices are either singular or (totally) unimodular as they resemble ordinary network matrices. It is obvious that any submatrix containing a zero row or column is singular.

A non-singular square submatrix R of $M(N)$ thus can contain a combination of the following non-zero rows or columns respectively.

Property 101 (Submatrices of $M(N)$) *A non-singular square submatrix R of $M(N)$ consists of non-zero rows and columns of one of the following types:*

- *row r:*

Type 1 *contains one entry 1, all others zero*

Type 2 *contains one entry -1, all others zero*

Type 3 *contains up to $|S|$ entries 1, all others zero*

Type 4 *contains up to $|S|$ entries -1, all others zero*

Type 5 *contains up to $|S|$ entries 1 and one entry -1, all others zero*

Type 6 *contains up to $|D|$ entries 1, all others zero*

Type 7 *contains up to $|D|$ (equal) entries $-k$, all others zero*

Type 8 *contains up to $|D|$ (equal) entries $-k$ and one entry 1, all others zero*

- *column c:*

Type 1 *contains one entry* 1*, all others zero*

Type 2 *contains one entry* -1*, all others zero*

Type 3 *contains one entry* $-k$*, all others zero*

Type 4 *contains one entry* -1 *and one entry* 1*, all others zero*

Type 5 *contains one entry* $-k$ *and one entry* 1*, all others zero*

Further properties of a non-singular submatrix R of $M(N)$ are the following:

Property 102 (Type 1,2 or 3 column) R *contains at least one type 1,2 or 3 column.*

Property 103 (Value of $\mathtt{det}(R)$) *The determinant* $\mathtt{det}(R)$ *of* R *is a power of* $\pm k$.

Property 102 is obvious as the row vectors are linear dependant (add all rows with entries $-k$ and k times all rows with entries 1 resulting in the zero row vector) for a submatrix R of $M(N)$ only containing columns of type 4 and 5, such that R is singular in this case.

For property 103 we develop $\mathtt{det}(R)$ based on a column of type 1,2 or 3, such that $\mathtt{det}(R) \in \{\pm\mathtt{det}(R'), -k\mathtt{det}(R')\}$, where R' is the appropriate smaller submatrix of $M(N)$. As R is non-singular, also R' must be non-singular, thus containing a column of type 1,2 or 3. Continuing the development of $\mathtt{det}(R')$ iteratively until the smaller submatrix is a 1×1-matrix and thus the determinant is ± 1 or $-k$ results in the property.

The inverse R^{-1} of R is defined as $R^{-1} = \frac{1}{\mathtt{det}(R)}\mathtt{adj}(R)$, where $\mathtt{adj}(R)$ is the adjoint of R. Let R be a $r \times r$ matrix, then the entries of $\mathtt{adj}(R)$ are ± 1 times the determinants of all $r-1 \times r-1$ submatrices R' of R. To ensure $kR^{-1} \in \mathbb{N}^r \times \mathbb{N}^r$ and given Property 103, we show the following Lemma. (Note that the difference to the argument for Property 103 is that we consider any submatrix R', not only the appropriate and relevant submatrix occurring during the development of $\mathtt{det}(R)$.)

Lemma 104 (Relation of Determinants $\mathtt{det}(R), \mathtt{det}(R')$) *Let* R *be a* $r \times r$ *non-singular square submatrix of* $M(N)$ *and* R' *a* $r-1 \times r-1$ *non-singular square submatrix of* R*, then* $\mathtt{det}(R) \in \{\pm\mathtt{det}(R'), -k\mathtt{det}(R')\}$ *or* R' *is singular.*

Proof: For the relevant $r-1 \times r-1$ submatrix R^* occurring during the development of $\mathtt{det}(R)$ as above, we know $\mathtt{det}(R)$ is either $\pm\mathtt{det}(R^*)$ or $-k\mathtt{det}(R^*)$. Let c be the column by which we developed $\mathtt{det}(R)$. Assume $\mathtt{det}(R) = \pm\mathtt{det}(R^*)$. Then c contains exactly one entry 1 or -1 and all R' containing the same columns as R^* either have the same determinant $\mathtt{det}(R^*)$ or are singular.

Further, each R', which contains c does not contain another column c', which is contained in R^*. If c' does not contain an entry $-k$, then again (depending on which rows R' contains) $\det(R') = \det(R^*)$ or R' is singular. If c' contains an entry $-k$, then (depending on which rows R' contains) either $\det(R') = \frac{-1}{k}\det(R^*)$, as the development of $\det(R')$ lacks a column of type 3 compared to $\det(R^*)$ or R' is singular. Thus, if R' is not singular $\det(R') \in \{\det(R^*), \frac{-1}{k}\det(R^*)\}$ and as we assumed $\det(R) = \pm\det(R^*)$ it follows that $\det(R) \in \{\pm\det(R'), -k\det(R')\}$.

Now assume $\det(R) = -k\det(R^*)$. Then c contains exactly one entry $-k$ and all R' containing the same columns as R^* either have the same determinant $\det(R^*)$ or are singular. Further, each R', which contains c does not contain another column c', which is contained in R^*. If c' contains an entry $-k$, then again (depending on which rows R' contains) $\det(R') = \det(R^*)$ or R' is singular. If c' does not contain an entry $-k$, then (depending on which rows R' contains) either $\det(R') = -k\det(R^*)$, as this time the development of $\det(R^*)$ lacks a column of type 3 compared to $\det(R')$ or R' is singular. Thus, if R' is not singular $\det(R') \in \{\pm\det(R^*), -k\det(R^*)\}$ and as we assumed $\det(R) = -k\det(R^*)$ it follows that $\det(R) \in \{k\det(R'), \pm\det(R')\}$.
□

We can now show:

Theorem 105 ($M(N)$ is k-regular) *The node arc incidence matrix of a k-regular network is k-regular.*

Proof: For each non-singular square submatrix R of $M(N)$, the inverse $R^{-1} = \frac{1}{\det(R)}\mathrm{adj}(R)$ has entries $a = \frac{\det(R')}{\det(R)}$. By Lemma 104 either $a = 0$ (if R' is singular) or $a \in \{\pm 1, \frac{-1}{k}\}$, such that $ka \in \mathbb{N}$ in each case. □

BIBLIOGRAPHY

[1] I. Adler and S. Corsares. A strongly polynomial algorithm for a special class of linear programs. *Operations Research*, 39(6):955–960, 1991. (Cited on page 97.)

[2] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993. (Cited on pages 4, 9, 10, 31, 34, 36, 51, 57, 58, 66, 102, and 137.)

[3] R.K. Ahuja, C.B. Cunha, and G. Sahin. Network models in railroad planning and scheduling. *Tutorials in Operations Research*, 1:54–101, 2005. (Cited on pages 6 and 8.)

[4] R.K. Ahuja, A.V. Goldberg, J.B. Orlin, and R.E. Tarjan. Finding minimum-cost flows by double scaling. *Mathematical Programming*, 53:243–266, 1992. (Cited on page 57.)

[5] R.K. Ahuja, K.C. Jha, and J. Liu. Solving real-life railroad blocking problems. *Interfaces*, 37(5):404–419, 2007. (Cited on page 7.)

[6] R.K. Ahuja, J. Liu, J.B. Orlin, D. Sharma, and L.A. Shughart. Solving real-life locomotive-scheduling problems. *Transportation Science*, 39:503–517, 2005. (Cited on page 7.)

[7] G. Appa. K-integrality, an extension of total unimodularity. *Operations Research Letters*, 13:159–163, 1993. (Cited on pages 36 and 160.)

[8] G. Appa and B. Kotnyek. Rational and integral k-regular matrices. *Discrete Mathematics*, 275(1-3):1–15, 2004. (Cited on pages 36, 87, 160, and 161.)

[9] A.A. Assad. Models for rail transportation. *Transportation Research Part A: General*, 14:205 – 220, 1980. (Cited on page 8.)

[10] F. Barahona and E. Tardos. Note on weintraub's minimum-cost circulation algorithm. *SIAM Journal on Computing*, 18, 1989. (Cited on page 97.)

[11] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958. (Cited on page 98.)

[12] K. Beygang. Modelle und Algorithmen für die Leerwagendisposition im Schienengüterverkehr, 2008. Diplomarbeit, AG Optimierung, Fachbereich Mathematik, Technische Universität Kaiserslautern. (Cited on pages 14 and 26.)

[13] K. Beygang, S.O. Krumke, and C. Zeck. Generalized max flow in series-parallel graphs, 2010. WIMA Report, Band 125. (Cited on page 99.)

[14] R.G. Bland and D.L. Jensen. On the computational behavior of a polynomial-time network flow algorithm. *Mathematical Programming*, 54:1–39. (Cited on page 57.)

[15] R.G. Busaker and P.J. Gowen. A procedure for determining a family of minimal-cost network flow patterns. Technical Report 15, Operational Research Office, John Hopkins University, Baltimore, MD, 1961. (Cited on pages 10, 57, and 97.)

[16] A. Caprara, M. Fischetti, and P. Toth. Modeling and solving the train timetabling problem. *Operations Research*, 50:851–861, 2002. (Cited on page 7.)

[17] A. Caprara, M. Monaci, P. Toth, and P.L. Guida. A lagrangian heuristic algorithm for a real-world train timetabling problem. *Discrete Applied Mathematics*, 154:738–753, 2006. (Cited on page 7.)

[18] V. Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, 1983. (Cited on pages 35 and 36.)

[19] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 23(6):1313–1347, 1994. (Cited on page 97.)

[20] E. Cohen and N. Megiddo. New algorithms for generalized network flows. *Mathematical Programming*, 64:325–336, 1994. (Cited on page 97.)

[21] W. Cook and A. Rohe. Computing minimum-weight perfect matchings. *INFORMS Journal on Computing*, 11:138–148, 1999. (Cited on page 123.)

[22] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1998. (Cited on pages 128, 131, and 136.)

[23] J.-F. Cordeau, P. Toth, and D. Vigo. A survey of optimization models for train routing and scheduling. *Transportation Science*, 32(4):380–404, 1998. (Cited on pages 6 and 7.)

[24] T.H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2001. Second Edition. (Cited on pages 34 and 126.)

[25] T. G. Crainic, M. Gendreau, and P. Dejax. Dynamic and stochastic models for the allocation of empty containers. *Operations Research*, 41(1):102–126, 1993. (Cited on pages 6 and 9.)

[26] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, USA, 1963. (Cited on pages 35, 57, and 96.)

[27] A. Darmann, U. Pferschy, J. Schauer, and G.J. Woeginger. Paths, trees and matchings under disjunctive constraints. Optimization Online. (Cited on page 126.)

[28] P. Dejax and T. G. Crainic. A review of empty flows and fleet management in freight transportation. *Transportation Science*, 21(4):227–247, 1987. (Cited on page 6.)

[29] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. (Cited on page 58.)

[30] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics Doklady*, 11:1277–1280, 1970. (Cited on pages 57 and 97.)

[31] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. (Cited on page 123.)

[32] J. Edmonds and E. L. Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 5:88–124, 1973. (Cited on page 123.)

[33] J. Edmonds and R.M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19, 1972. (Cited on pages 57 and 97.)

[34] T. Erlebach and K. Jansen. The maximum edge-disjoint paths problem in bidirected trees. *SIAM J. Discrete Math*, 14:326–355, 2001. (Cited on page 125.)

[35] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1736. (Cited on pages 32 and 108.)

[36] T. Fleiner, R.W. Irving, and D. Manlove. Efficient algorithms for generalized stable marriage and roommates problems. *Theoretical Computer Science*, 381:162–176, 2007. (Cited on page 123.)

[37] L. Fleischer and Z. Svitkina. Preference-constrained, oriented matching. In *2010 Proceedings of the Seventh Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 66–73. SIAM, 2010. (Cited on page 123.)

[38] L. Fleischer and K. D. Wayne. Fast and simple approximation schemes for generalized flow. *Mathematical Programming*, 91(2):215–238, 2001. (Cited on page 99.)

[39] Jr. L. R. Ford and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962. (Cited on pages 10, 51, 57, 96, and 123.)

[40] A. Frank. chapter Packing paths, curcuits and cuts – a survey, pages 47–100. Springer, Berlin, 1990. (Cited on page 125.)

[41] D. R. Fulkerson. An out of kilter method for minimal cost flow problems. *SIAM Journal on Applied Mathematics*, 9:18–20, 1961. (Cited on page 4.)

[42] H.N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In D. Johnson, editor, *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, pages 434–443, San Francisco, CA, USA, 1990. SIAM. (Cited on page 123.)

[43] H.N. Gabow, Z. Galil, and T.H. Spencer. Efficient implementation of graph algorithms using contraction. *Journal of the ACM*, 36:540–572, 1989. (Cited on page 123.)

[44] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–14, 1962. (Cited on page 123.)

[45] D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11:223–232, 1985. (Cited on page 123.)

[46] Z. Galil, S. Micali, and H. Gabow. An $O(EV \log V)$-algorithm for finding a maximal weighted matching in general graphs. *SIAM Journal on Computing*, 15:120–130, 1986. (Cited on page 123.)

[47] M. Garey and D. Johnson. *Computers and Intractability: A guide to the theory of NP-Completeness*. W.H. Freeman, New York, USA, 1979. (Cited on pages 35, 46, and 125.)

[48] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997. (Cited on page 123.)

[49] J.M. Gigon and F. Schlaepfer. Kybernetik und Elektronik bei den Eisenbahnen – Die Leerwagenverteilung auf einer elektronischen Rechenanlage. Technical report, 1967. (Cited on page 4.)

[50] F. Glover, J. Hultz, D. Klingman, and J. Stutz. Generalized networks: A fundamental computer-based planning tool. *Management Science*, 24:1209–1220, 1978. (Cited on page 97.)

[51] F. Glover and D. Klingman. On the equivalence of some generalized network problems to pure network problems. *Mathematical Programming*, 4:269–278, 1973. (Cited on pages 56, 85, 94, and 97.)

[52] A. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988. (Cited on page 97.)

[53] A. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM*, 36, 1989. (Cited on page 97.)

[54] A. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, 15, 1990. (Cited on pages 57, 58, 97, and 98.)

[55] A.V. Goldberg, S.A. Plotkin, and E. Tardos. Combinatorial algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 16:351–379, 1991. (Cited on pages 97, 98, and 159.)

[56] D. Goldfarb and J. Hao. A primal simplex algorithm that solves the maximum flow problem in at most nm pivots and $O(n^2m)$ time. *Mathematical Programming*, 47:353–365, 1990. (Cited on page 57.)

[57] D. Goldfarb and J. Hao. Polynomial-time primal simplex algorithms for the minimum cost network flow problem. *Algorithmica*, 8:145–160, 1992. (Cited on page 57.)

[58] D. Goldfarb and Z. Jin. A faster combinatorial algorithm for the generalized circulation problem. *Mathematics of Operations Research*, 21, 1996. (Cited on page 97.)

[59] D. Goldfarb, Z. Jin, and Y. Lin. A polynomial dual simplex algorithm for the generalized circulation problem. *Mathematical Programming*, 91(2):271–288, 2002. (Cited on pages 96 and 97.)

[60] D. Goldfarb, Z. Jin, and J.B. Orlin. Polynomial-time highest-gain augmenting path algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 22, 1997. (Cited on pages 97 and 98.)

[61] M. Gondran and M. Minoux. *Graphs and Algorithms*. John Wiley & Sons, New York, 1984. (Cited on pages 98 and 159.)

[62] M.F. Gorman, D. Acharya, and D. Sellers. CSX railway uses OR to cash in on optimized equipment distribution. *Interfaces*, 40(1):5–16, 2010. (Cited on pages 8 and 9.)

[63] M.F. Gorman, K. Crook, and D. Sellers. North american freight rail industry real-time optimized equipment distribution systems: State of the practice. *Transportation Research Part C: Emerging Technologies*, 19(1):103–114, 2011. (Cited on pages 8 and 9.)

[64] A.E. Haghani. Rail freight transportation: A review of recent optimization models for train routing and empty car distribution. *Journal of Advanced Transportation*, 21:147–172, 1987. (Cited on page 8.)

[65] A.E. Haghani. Formulation and solution of a combined train routing and makeup, and empty car distribution model. *Transportation Research*, 23:433–452, 1989. (Cited on page 8.)

[66] A. Hefner and P. Kleinschmidt. A constrained matching problem. *Annals of Operations Research*, 57:135–145, 1995. (Cited on page 123.)

[67] H. Herren. Computer controlled empty wagon distribution on the SSB. *Rail International*, 8(1). (Cited on page 8.)

[68] D.S. Hochbaum. Instant recognition of half integrality and 2-approximations. In *APPROX: International Workshop on Approximation Algorithms for Combinatorial Optimization*, 1998. (Cited on page 97.)

[69] D.S. Hochbaum, N. Megiddo, J. Naor, and A. Tamir. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Mathematical Programming*, 62:69–83, 1993. (Cited on page 97.)

[70] D.S. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23, 1994. (Cited on page 97.)

[71] K. Holmberg, M. Joborn, and J.T. Lundgren. Improved empty freight car distribution. *Transportation Science*, 32(2):163–173, 1998. (Cited on page 9.)

[72] P Ireland, R. Case, J. Fallis, C. Van Dyke, J. Kuehn, and M. Meketon. The canadian pacific railway transforms operations by using models to develop its operating plans. *Interfaces*, 34(1):5–14, 2004. (Cited on page 9.)

[73] M. Iri. A new method of solving transportation-network problems. *Journal of the Operations Research Society of Japan*, 3:27–87, 1960. (Cited on pages 10 and 57.)

[74] J.J. Jarvis and A.M. Jezior. Maximal flow with gains through a special network. *Operations Research*, 20:678–688, 1972. (Cited on page 97.)

[75] W. S. Jewell. Optimal flow through networks with gains. *Operations Research*, 10:476–499, 1962. (Cited on pages 10, 57, and 96.)

[76] W.S. Jewell. Optimal flow through networks. Technical Report 8, MIT, Cambridge, MA, 1958. (Cited on pages 10 and 96.)

[77] M. Joborn, T.G. Crainic, M. Gendreau, K. Holmberg, and J.T. Lundgren. Economies of scale in empty freight car distribution in scheduled railways. *Transportation Science*, 38:121–134, 2004. (Cited on pages 7 and 9.)

[78] R. Kannan. A polynomial algorithm for the two-variable integer programming problem. *Journal of the ACM*, 27(1):118–122, 1980. (Cited on page 97.)

[79] S. Kapoor and P. Vaidya. Speeding up Karmarkar's algorithm for multicommodity flows. *Mathematical Programming*, 73:111–127, 1996. (Cited on page 98.)

[80] N. Karmarkar. A new polynomial–time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984. (Cited on pages 57 and 98.)

[81] M.H. Keaton. Designing optimal railroad operating plans: Lagrangian relaxations and heuristic approaches. *Transportation Science*, 23:415–431, 1989. (Cited on page 8.)

[82] L. G. Khachian. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20(1):191–194, 1979. (Cited on pages 56 and 98.)

[83] M. Klein. A primal method for minimal cost flows. *Management Science*, 14:205–220, 1967. (Cited on pages 57 and 159.)

[84] V. Kolmogorov. Blossom v: A new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation (MPC)*, 1:43–67, 2009. (Cited on pages 123 and 131.)

[85] E. Landau. *Handbuch der Lehre von der Verteilung der Primzahlen.* B.G. Teubner, Leipzig, Berlin, 1909. (Cited on page 34.)

[86] E. L. Lawler, editor. *Combinatorial Optimization: Networks and Matroids.* Rinehart and Winston, New York, 1976. (Cited on page 123.)

[87] J. Liu. Solving real-life transportation scheduling problems. Dissertation, University of Florida. (Cited on page 7.)

[88] D. Manlove, R.W. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276:261–279, 2002. (Cited on page 123.)

[89] C. D. Martland and J. M. Sussman. A perspective on rail systems modelling: What works and what doesn't. Working Paper 95-1, Center for Transportation Studies, Massachusetts Institute of Technology, Cambridge, MA. (Cited on page 10.)

[90] K. Mehlhorn and G. Schäfer. Implementation of $O(nm \log n)$ weighted matchings in general graphs: The power of data structures. *ACM Journal of Experimental Algorithmics*, 7, 2002. (Cited on page 123.)

[91] M. Middendorf and F. Pfeiffer. On the complexity of the disjoint paths problems. *Combinatorica*, 13:97–107, 1993. (Cited on page 125.)

[92] G.J. Minty. Monotone networks. *Proceedings of the Royal Society of London*, 257A:194–212, 1960. (Cited on pages 4 and 57.)

[93] A.K. Narisetty, Richard J.-P. P., D. Ramcharan, D. Murphy, G. Minks, and J. Fuller. An optimization model for empty freight car assignment at union pacific railroad. *Interfaces*, 38(2):89–102, 2008. (Cited on page 9.)

[94] A.M. Newman, L. Nozick, and C.A. Yano. *Optimization in the Rail Industry*, pages 704–718. Oxford University Press, New York (NY), 2002. (Cited on pages 6, 9, and 10.)

[95] H.N. Newton, C. Barnhart, and P.H. Vance. Constructing railroad blocking plans to minimize handling costs. *Transportation Science*, 32:330–345, 1998. (Cited on page 7.)

[96] J. D. Oldham. Combinatorial approximation algorithms for generalized flow problems. *Journal on Algorithms*, 38(1):135–169, 2001. (Cited on page 98.)

[97] K. Onaga. Dynamic programming of optimal flows in lossy communication nets. *IEEE Transactions on Circuit Theory*, 13:308–327, 1966. (Cited on page 97.)

[98] K. Onaga. Optimal flows in general communication networks. *Journal of the Franklin Institute*, 283:308–327, 1967. (Cited on page 97.)

[99] J.B. Orlin. Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem. Technical Report 1615-84, Mass. Inst. of Technology, Sloan School of Management, Cambridge, 1984. (Cited on page 57.)

[100] M. Padberg and A. Sassano. The complexity of matching with bonds. *Information Processing Letters*, 32:297–300, 1989. (Cited on page 123.)

[101] W.B. Powell and T. Carvalho. Dynamic control of logistics queueing networks for large-scale fleet management. *Transportation Science*, 32(2):90–109, 1998. (Cited on page 9.)

[102] W.B. Powell and T. Carvalho. Real-time optimization of containers and flatcars for intermodal operations. *Transportation Science*, 32(2):110–126, 1998. (Cited on page 9.)

[103] T. Radzik. Faster algorithms for the generalized network flow problem. In *Proceedings of the 34th Annual Symposium on Foundations of Comptuer Science*, pages 438–448, Palo Alto, CA, 1993. IEEE. (Cited on page 98.)

[104] T. Radzik. Improving time bounds on maximum generalised flow computations by contracting the network. *TCS: Theoretical Computer Science*, 312, 2004. (Cited on page 98.)

[105] M. Restrepo and D.P. Williamson. A simple GAP-canceling algorithm for the generalized maximum flow problem. In *SODA*, pages 534–543. ACM Press, 2006. (Cited on page 98.)

[106] H. Röck. *Scaling Techniques for Minimal Cost Network Flows*, pages 182–191. Hanser, München, 1980. (Cited on page 57.)

[107] S. Sahni. Computationally related problems. *SIAM Journal on Computing*, 3:262–279, 1974. (Cited on pages 86 and 153.)

[108] F. Schlaepfer. *Praktische Studien zur Unternehmensforschung*, chapter Kostenoptimale Verteilung leerer Güterwagen. Springer Verlag, 1970. (Cited on page 4.)

[109] F. Schlaepfer. Rechnergesteuerte Leerwagenverteilung (LWV) – Das mathematische Modell, 1991. (Cited on page 4.)

[110] M. Shigeno. A survey of combinatorial maximum flow algorithms on a network with gains. (Cited on page 97.)

[111] E. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–256, 1985. (Cited on page 57.)

[112] E. Tardos and K.D. Wayne. Simple generalized maximum flow algorithms. In *IPCO: 6th Integer Programming and Combinatorial Optimization Conference*, 1998. (Cited on page 98.)

[113] H. Topaloglu and W.B. Powell. Sensitivity analysis of a dynamic fleet management model using approximate dynamic programming. *Operations Research*, 55(2):319–331, 2007. (Cited on page 9.)

[114] C.A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8:85–89, 1984. (Cited on page 46.)

[115] K. Truemper. On max flows with gains and pure min-cost flows. *SIAM Journal on Applied Mathematics*, 32(2):450–456, 1977. (Cited on pages 56, 97, and 98.)

[116] M.A. Turnquist. Economies of scale and network optimization for empty car distribution. Technical report, Cornell University, Ithaca, NY, 1994. (Cited on page 7.)

[117] M.A. Turnquist and B.P. Markowicz. An interactive microcomputer-based model for railroad car distribution. Working Paper, Cornell University, Ithaca, NY. (Cited on page 8.)

[118] P. Vaidya. Speeding-up linear programming using fast matrix multiplication. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1989. (Cited on page 98.)

[119] K.D. Wayne. A polynomial combinatorial algorithm for generalized minimum cost flow. *Mathematics of Opererations Research*, 27:445–459, 2002. (Cited on page 99.)

[120] W. W. White and A. M. Bomberault. A network algorithm for empty freight car allocation. *IBM Syst. J.*, 8:147–169, 1969. (Cited on page 8.)

[121] X. Zhou, S. Tamura, and T. Nishizeki. Finding edge-disjoint paths in partial k-trees. *Algorithmica*, 26:3–30, 2000. (Cited on page 125.)

[122] E. Zhu. Scheduled service network design for integrated planning of rail freight transportation. Dissertation, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), University of Montreal, Canada. (Cited on page 7.)

Erklärung

Ich versichere, dass ich die von mir vorgelegte Dissertation selbstständig angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe, dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat, dass sie - abgesehen von den unten angegebenen Teilpublikationen - noch nicht veröffentlicht wurde sowie, dass ich eine solche Veröffentlichung vor Abschluss des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen der Promotionsordnung der mathematisch-naturwissenschaftlichen Fakultät der Universität zu Köln sind mir bekannt. Die von mir vorgelegte Dissertation wurde von Prof. Dr. Rainer Schrader betreut.

Teilpublikationen

[1] B. Engels, R. Schrader
"A Generalized Flow Network for Freight Car Dispatching."
Manuskript eingereicht zu "Networks - Special Issue on Algorithmic Approaches for Transportation Modelling, Optimization and Systems", Dezember 2010.