Novel applications of Column Generation in large-scale linear programming

DISSERTATION

zur Erlangung des akademischen Grades
Dr. rer. nat.

eingereicht an der
Mathematisch–Naturwissenschaftlich–Technischen Fakultät
der Universität Augsburg

Universität Augsburg
Mathematisch-Naturwissenschaftlich-
Technische Fakultät

von

Jonas Schwinn

Tag der mündlichen Prüfung:	2. Juli 2019
Erster Gutachter:	Professor Dr. Ralf Werner
Zweiter Gutachter:	Professor Dr. Jörg Fliege

Zusammenfassung

Die folgende Dissertation befasst sich mit zwei Problemen aus dem Bereich der linearen Optimierung. Der erste Teil der Arbeit behandelt das Transportproblem, ein klassisches Problem der mathematischen Optimierung; eine ausführliche Definition der Problemstellung erfolgt im ersten Kapitel. Die ersten bedeutsamen Anwendungen aus der Logistik entstanden in der ersten Hälfte des 20. Jahrhunderts und motivierten wichtige Forschungsbeiträge zur linearen Optimierung und Netzwerkflusstheorie von Tolstoi, Kantorovich, Hitchcock, Koopmans, Dantzig, Ford und Fulkerson. Dazu kommt heute ein stark ansteigendes Interesse aus Bereichen wie Fertigungsindustrie, Informatik, Bildverarbeitung, Datenanalyse und maschinellem Lernen. Die wahrscheinlich am häufigsten verwendete Lösungsmethode für das Transportproblem ist der ursprünglich von Dantzig (1951) entwickelte Transportsimplex. Das Hauptziel dieser Arbeit wird es sein, Konzepte der Spaltengenerierung auf diesen Algorithmus anzuwenden. Unsere Methode basiert auf einer aktuellen Veröffentlichung von Gottschlich und Schuhmacher (2014) und führt Low-Level-Änderungen in der Implementierung des Transportsimplex ein. Nach unserem besten Wissen existiert dieser Ansatz in der Literatur bisher nicht. Wir bauen auf einer effizienten Implementierung des Transportsimplex auf, um eine numerisch wettbewerbsfähige Version des Spaltengenerierungsansatzes zu entwickeln. Im Rahmen dessen betrachten wir außerdem heuristische Ansätze für das Transportproblem, die verwendet werden, um Startlösungen zu berechnen und einen signifikanten Einfluss auf die Gesamtlaufzeit des Algorithmus haben. Als Nebenprodukt unserer Analyse vergleichen wir die Implementierung des Transportsimplex mit dem allgemeineren Netzwerksimplex. Während wir in theoretischen Untersuchungen sowohl symmetrische als auch asymmetrische Transportprobleme betrachten, konzentriert sich die numerische Analyse auf den symmetrischen Fall.

Im zweiten Teil der Arbeit entwickeln wir eine weitere, speziell zugeschnittene Spaltengenerierung für ein lange bestehendes Problem aus der multivariaten Statistik, das in Kapitel 6 beschrieben wird. Die Prüfung einer Matrix auf Zugehörigkeit zur Familie der Bernoulli-Matrizen ist aufgrund der vielfältigen An-

wendungen der Bernoulli-Vektoren in Bereichen wie Informationstechnologie, Finanzen, Medizin und Operations Research von hoher praktischer Relevanz. Dieses Problem ist außerdem eng verwandt mit der Prüfung auf Zugehörigkeit zum – im Bereich der mathematischen Optimierung besser bekannten – Korrelationspolytop und ist damit NP-vollständig, vgl. Pitowsky (1991). Für unseren Ansatz verwenden wir eine lineare Formulierung; der dabei auftretenden exponentiellen Anzahl von primalen Variablen begegnen wir mit Spaltengenerierung und einer einfachen, aber neuartigen Methode für die Lösung der auftretenden Teilprobleme.

Abschließend wollen wir darauf hinweisen, dass alle Algorithmen in MATLAB realisiert sind und dass die Kapitel 3 und 6 auf den Veröffentlichungen von Schwinn und Werner (2018) und Krause et al. (2018) basieren.

Abstract

In the context of this dissertation we consider two mathematical optimization problems. The first constitutes a classical problem of linear optimization, called the transportation problem; a detailed introduction is given in the first chapter. This problem motivated significant contributions in research on linear optimization and network flow problems and plays an important role in praxis to this day. Besides the logistic applications that motivated the work of Tolstoi, Kantorovich, Hitchcock, Koopmans, Dantzig, Ford and Fulkerson in the first half of the 20th century, the problem is now also attracting renewed interest in areas such as manufacturing industry, computer science, image processing, data analysis and machine learning. The main goal of this thesis is to apply the concept of Column Generation to a popular solution method originally devised by Dantzig (1951) called the Transportation Simplex. To the best of our knowledge, this approach has not yet been taken into account in the literature. Our method is based on a recent publication by Gottschlich and Schuhmacher (2014) and introduces low-level changes in the implementation of the Transportation Simplex. In order to develop a numerically competitive version of the Column Generation approach, we need to build on an efficient implementation of this algorithm, which is described in Chapter 4. In this course, we further consider heuristic approaches to the transportation problem in Chapter 3. These are used to determine initial solutions for the Transportation Simplex and have significant impact on overall performance. As a by-product of our analysis, we compare the Transportation Simplex with the more general Network Simplex. While we consider symmetric as well as asymmetric transportation problems in the theoretical part of this work, the numerical analysis is focused on the symmetric case.

Second, we develop another specifically tailored Column Generation method for a long-standing problem in the field of multivariate statistics in Chapter 6. Testing a given matrix for membership in the family of Bernoulli matrices is of high practical relevance due to the manifold applications of Bernoulli vectors in areas such as information technology, finance, medicine and operations research. The problem is known to be NP-complete, due to its relation

to membership testing on the well-known correlation polytope, cf. Pitowsky (1991). For this purpose, we propose to solve a linear formulation of the problem by Column Generation and deal with the issue of exponentially many primal variables by applying a straightforward, yet novel, solution of the arising subproblems.

As a final note, observe that all algorithms are realized in MATLAB and that Chapters 3 and 6 are based on the publications of Schwinn and Werner (2018) and Krause et al. (2018).

Danksagung

Ohne die Unterstützung zahlreicher Personen hätte die vorliegende Dissertation in dieser Form nicht realisiert werden können, weswegen ich mich für die vielfältige Hilfe an dieser Stelle ganz herzlich bedanken möchte.

Mein besonderer Dank gilt zunächst meinem Doktorvater Prof. Dr. Ralf Werner, bei dem ich eine hervorragende Ausbildung genießen durfte. Danke für die gute Betreuung und Zusammenarbeit sowie die vielen Ideen und Anregungen.

Meinen Kolleginnen und Kollegen – Kathrin, Armin, Christian, Daniel, Jan und Max – danke ich für drei sehr schöne Jahre am Lehrstuhl. Danke für viele anregende und wissenschaftliche, aber auch viele weniger wissenschaftliche und unterhaltsame Gespräche.

Abschließend gebührt der größte Dank meiner Frau Damaris, für all ihre praktische und moralische Unterstützung in den letzten Jahren und meinem Sohn Joel, der seinen Papa an so vielen Abenden und Wochenenden hergegeben hat.

Phil 1:6

Contents

List of Figures

List of Tables

List of Algorithms

1 The transportation problem

The transportation problem is one of the classical problems in mathematical optimization (see e.g. Glover et al. (1974b) or Gass (1990)). It has an early history that can be traced back to the work of Monge (1781) who studied a civil engineering problem of "cutting and filling", posed by the French Academy of Sciences, which constitutes the equivalent of the transportation problem in continuous form. Several years later, it played a key role in the development of linear programming in Kantorovich (1960)[1], Hitchcock (1941) and Koopmans (1949) as well as in the invention of the famous Simplex method, cf. Dantzig (1963), and its specialization for minimum cost flow problems by Dantzig (1951). While these initial contributions were mainly motivated by problems in the field of logistics, its applications today extend to many areas, including computer science, data analysis, finance, engineering, operations research and linguistics.

For a more detailed account of the most important publications towards the problem as well as recent applications we refer the reader to Section 1.6 and 1.7.

In the context of this thesis, we will consider the problem in its classical version which is frequently associated with Hitchcock (1941). More precisely, this implies that we will assume finite sets of origins and destinations and study the linear formulation. Moreover, the respective problem will be uncapacitated and dense without restricted (inadmissible) cells or bounds on the variables. In particular, we will not impose a specific structure on the cost matrix of the problem. This separates our work from the field of optimal transport and in

[1] This refers to the English translation of the original article, published 1939 in Russian.

particular the Wasserstein distance or Earth Mover's distance, see e.g. Rachev and Rüschendorf (1998), Villani (2009) and Rubner et al. (1997). However, these problems will be part of our numerical evaluation and represented by the DOTMARK benchmark of Schrieber et al. (2017), see Section 2.3.5.

We will give the mathematical formulation of the problem discussed in this thesis in Section 1.2 and refer to this form as the *(classical) transportation problem* throughout this work.

1.1 Motivation

Our primary incentive is to apply concepts based on Column Generation to the Transportation Simplex. As our approach modifies the low-level implementation of the Transportation Simplex, a necessary prerequisite for this is an efficient implementation of this algorithm in MATLAB – which is a secondary goal of this work. The Transportation Simplex is arguably the best known algorithm for solving transportation problems. Judging by both older and more recent comparative studies (see e.g. Srinivasan and Thompson (1972), Glover et al. (1974b) and Schrieber et al. (2017)), it yields attractive numerical results and seems to be the algorithm of choice for adept practitioners (Gass (1990)). In order to avoid ambiguities in the following and further support our motivation, let us differentiate the terminology used with regard to the Simplex method at this point: The standard Simplex method is devised for general linear programs, cf. Dantzig (1963); we will refer to this algorithm as LP Simplex or LPS for short. For this algorithm, a specialization for general minimum cost flow problems has been developed which is called the Network Simplex[2] (NS). Since the transportation problem represents a special case of minimum cost flow problems which in turn are linear problems, direct applications of both the LPS and the NS to the transportation problem are possible; moreover,

[2] Observe that there exist variants of the Network Simplex for other network flow problems, as for instance maximum flow or shortest paths. However, since we are interested in a generalization of the transportation problem, we will always refer to the Network Simplex for minimal cost flow problems in the context of this work.

using the NS already yields good results. However, as the numerical studies of Chapters 4 and 5 verify, an additional customization is still worthwhile. The respective algorithm is consequently called Transportation Simplex or TPS[3].

It is noteworthy that the first Network Simplex method of Dantzig (1951) was originally developed for the transportation problem. In fact, this publication was a starting point for further research on this topic, which led to numerous Simplex codes in the middle of the 20th century, which were specifically developed for the transportation problem and optimized for different hardware compositions, cf. Charnes et al. (1975). However, after this initial phase, the focus of research shifted; due to the wider range of applications, theoretical and practical investigations focused on the LPS and the NS. As a result, excellent implementations of these algorithms are easier to obtain these days, which means that they are usually preferred to the TPS in today's applications. Another reason for this approach is that the corresponding algorithms work sufficiently well in many cases, cf. Gass (1990), Kovács (2015) and Schrieber et al. (2017). Accordingly, recent algorithmic studies have been focused on the more general minimum cost flow problem, e.g. Kovács (2015), or were targeting specific subclasses of the transportation problem, for example computing the Wasserstein distance of images as in Gottschlich and Schuhmacher (2014) and Schrieber et al. (2017).

All this contributes to our motivation to develop an efficient implementation of the TPS in MATLAB in Chapter 4. Furthermore, we will evaluate this algorithm on general problem instances in Section 4.5. For this purpose, we present several problem classes, both artificially generated and derived from real-world problems for the transportation problem in Chapter 2. In the course of this, we extend a method proposed by Arthur and Frendewey (1994) for capacitive minimum cost flow problems, by guaranteeing statistical properties for a class

[3]Note that due to its long existence and the many contributions published in course of the last century, several names have been proposed for the TPS. Among many others these include: the MODI (Modified Distribution) Method, the Stepping-Stone Method or the Row-Column Sum Method. In the context of this thesis we choose the term Transportation Simplex to emphasize the structural resemblance with the NS and thus the LPS.

of randomly generated problem instances. Moreover, we will quantify the benefits of a specialized implementation of the TPS versus the general NS, which will be represented by an efficient commercial solver. To this end, it is critical to investigate heuristic approaches to the transportation problem, as they are used to produce initial solutions in the TPS. This will be done in Chapter 3.

Finally, based on our analysis in Chapter 4 and the results of Gottschlich and Schuhmacher (2014), we will try to accelerate the TPS by applying concepts from column generation in Chapter 5. To the best of our knowledge this approach has not yet been considered. The result of this modification will then again be numerically analyzed on the basis of the problem classes already mentioned.

The rest of this chapter is organized as follows. In order to provide an intuitive understanding of the problem under investigation, we will begin by describing the commonly used economic interpretation. Thereupon follows the formal description of the problem and its properties as a linear program. With regard to the subsequent chapters, this approach is then supplemented by an analysis within the framework of network flow problems and the study of some properties of the problem parameters (C, a, b). Finally, a summary of the most important historical developments is given and we conclude with an overview of the current research.

1.2 The model

Let us introduce the problem which will be the subject of a major part of this work. To begin with, we would like to make a non-formal approach to the problem by stating the classic economic interpretation of the primal problem formulation (for an interpretation of the dual problem, see e.g. Gottschlich and Schuhmacher (2014)). The intention is to introduce the common terminology and provide an intuitive understanding of the problem at hand and thereby the concepts treated in the remainder of this work. The mathematical definition will be given in the subsequent section where we additionally state some basic

assumptions made in the context of this thesis.

1.2.1 Economical interpretation

The most commonly used interpretation for the transportation problem is as follows: We consider the distribution of some homogeneous commodity that is in stock at m production sites and in demand at n consumption facilities. To improve the readability, we will refer to these in the following as origins and destinations. In this constellation, each origin i provides a *supply* $a_i > 0$ of the commodity while each destination j has a *demand* $b_j > 0$ for the commodity. Furthermore, we will assume that the problem is *balanced*, i.e. the total supply equals the total demand. The task of distributing this commodity is entrusted to a forwarding company. This company incurs costs for the transport of one unit of the commodity from i to j which are given by arbitrary *cost coefficients* c_{ij} and collectively form the *cost matrix* $C := (c_{ij})_{i,j=1}^{m,n}$. In order to solve the task assigned to it, the company devises a transportation plan (transportation matrix) $X := (x_{ij})_{i,j=1}^{m,n}$ by determining the amount $x_{ij} \geq 0$ of transported units between origin i and destination j. Such a plan is called *feasible*, if all supplies a_i are forwarded and all demands b_j are saturated. The aim of the forwarder is to find a feasible transportation plan with minimum costs which, by virtue of the foregoing characterization, constitutes the transportation problem.

1.2.2 Mathematical definition

More formally, the problem can be stated as the following linear program:

$$\begin{array}{lrcll} \min\limits_{x \in \mathbb{R}^{mn}} & \sum_{i=1}^{m} \sum_{j=1}^{n} & c_{ij} x_{ij} & & \\ \text{s.t.} & \sum_{j=1}^{n} & x_{ij} & = a_i & \forall i \in I \\ & \sum_{i=1}^{m} & x_{ij} & = b_j & \forall j \in J \\ & & x_{ij} & \geq 0 & \forall (i,j) \in I \times J \end{array} \tag{TP}$$

where $I := \{1, \ldots, m\}$ represents the set of origins and $J := \{1, \ldots, n\}$ the set of destinations, respectively. Since the constraints $\sum_{j=1}^{n} x_{ij} = a_i$ for $i \in I$

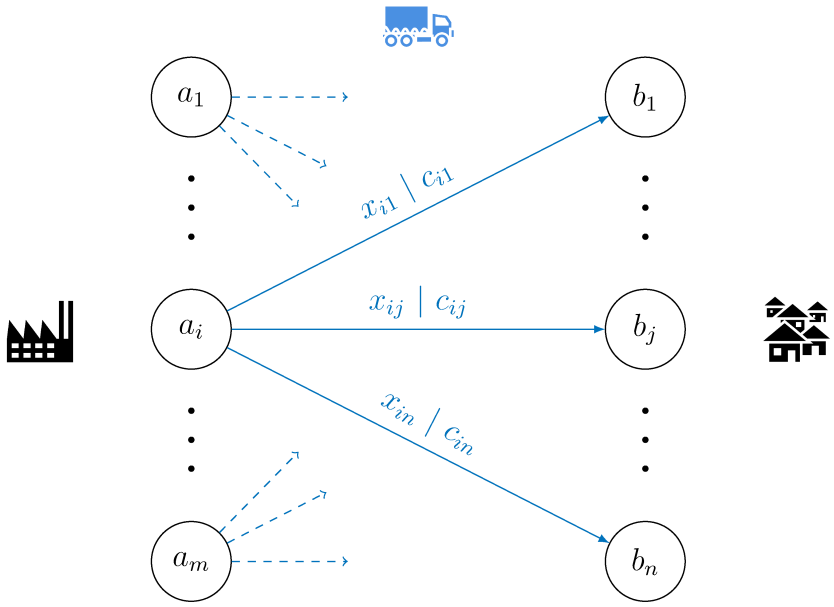


Figure 1.1: A forwarder transports a homogeneous commodity from m origins to n destinations while minimizing his transportation costs.

impose restrictions on the rows of the transportation matrix X, we shall call these the *row constraints* and accordingly $\sum_{i=1}^{m} x_{ij} = b_j$ for $j \in J$ the *column constraints* of the problem. Together with the non-negativity constraints on the x_{ij} they define the *feasible region* S of the transportation problem. Furthermore, observe that according to the above definition a transportation problem is uniquely defined by the triple (C, a, b). Thus, for reasons of simplicity, we will hereinafter speak of the transportation problem (C, a, b) when considering (TP) with parameters (C, a, b). Moreover, we will denote optimal solutions of the problem by x^*, the set of optimal solution by S^* and the optimal value by f^*. If we want to highlight the fact that optimal solutions are achieved for a certain input we will also use the notation

$$S(a, b),\ x^*(C, a, b),\ S^*(C, a, b) \text{ and } f^*(C, a, b).$$

Vectorized formulations

In order to obtain a vectorized formulation, we reshape C and X to vectors

$$c \coloneqq (c_{11}, c_{12}, \ldots, c_{1n}, c_{21}, \ldots c_{mn})^\top$$
$$x \coloneqq (x_{11}, x_{12}, \ldots, x_{1n}, x_{21}, \ldots x_{mn})^\top$$

and set

$$a \coloneqq (a_1, \ldots, a_m)^\top$$
$$b \coloneqq (b_1, \ldots, b_n)^\top$$

which yields the equivalent formulation

$$\begin{array}{lrcl} \min\limits_{x \in \mathbb{R}^{mn}} & \langle c, x \rangle & & \\ \text{s.t.} & Ax & = & a \\ & Bx & = & b \\ & x & \geq & 0 \end{array} \tag{TP_{AB}}$$

where $\langle a, b \rangle \coloneqq a^\top b$ denotes the corresponding scalar product. Here, the row constraints are modeled by means of the *row constraint matrix*

$$A \coloneqq \left(\begin{array}{cccc|cccc|c|cccc} 1 & 1 & \ldots & 1 & & & & & & & & \\ & & & & 1 & 1 & \ldots & 1 & \ldots & & & \\ & & & & & & & & & 1 & 1 & \ldots & 1 \end{array}\right) \in \mathbb{R}^{m \times mn}$$

consisting of m horizontally aligned blocks of size $m \times n$, where the only non-zeros of the i-th block, $i \in I$, are given by its i-th row, which constitutes a vector of ones. In contrast to that, the column constraints are defined by the *column constraint matrix*

$$B \coloneqq \left(\begin{array}{cccc|cccc|c|cccc} 1 & & & & 1 & & & & & 1 & & \\ & 1 & & & & 1 & & & & & 1 & \\ & & \ddots & & & & \ddots & & \ldots & & & \ddots \\ & & & 1 & & & & 1 & & & & 1 \end{array}\right) \in \mathbb{R}^{n \times mn}$$

given in m horizontally aligned identity matrices of size $n \times n$. Furthermore, these matrices will sometimes be concatenated for notational purposes to form the *constraint matrix* M and the right-hand side e

$$M := \begin{pmatrix} A \\ B \end{pmatrix} \text{ and } e := \begin{pmatrix} a \\ b \end{pmatrix}$$

which entails an additional equivalent formulation for (TP):

$$\begin{aligned} \min_{x \in \mathbb{R}^{mn}} \quad & \langle c, x \rangle & & \\ \text{s.t.} \quad & Mx & = & e \\ & x & \geq & 0. \end{aligned} \tag{TP$_M$}$$

Non-redundant formulation

In addition to the three equivalent formulations for the original problem stated above, we introduce the *non-redundant formulation of the transportation problem* with regard to some theoretical statements made later in this thesis. Hereby, we delete, w.l.o.g., the first row of the constraint matrix M in (TP$_M$) to achieve full rank (see Section 1.3) and remove the corresponding entry in e. Accordingly, we delete one column, i.e. set the appropriate dual variable to zero for the *dual non-redundant formulation*.

Remarks on notation

Finally, a few words on notation styles for the transportation problem: The notations (C, X) and (c, x) are mathematically equivalent but have different notation advantages depending on the context. Since most textbooks and publications seem to prefer the perspective of cost matrices and transportation plans, i.e. (C, X), we will try to present the results in this perspective. However, we will switch notations whenever we feel that one allows more intuition or shorter notation than the other.

1.2.3 Preparatory assumptions and propositions

In the following, we will give some fundamental assumptions and definitions concerning the (input) parameters (C, a, b) and prove related propositions:

Assumption 1
We assume $C \in \mathbb{R}^{m \times n}$, $a \in \mathbb{R}^m_{>0}$, $b \in \mathbb{R}^n_{>0}$ *and* $m \geq n$.

This holds without loss of generality since the problem becomes infeasible for $a_i < 0$ or $b_j < 0$ and one can always obtain equivalent problems of lower dimension satisfying $a > 0$ and $b > 0$ by leaving out zero supplies and demands.

Furthermore, we note that the roles of the a and b in the problem are completely interchangeable and therefore assume – in the interest of consistency – that the vector a is always of higher or equal dimension than b. In particular, we will assume $m \geq n$ when stating worst-case complexities of algorithms. With respect to this last assumption we divide the instances of the problem in the following two classes: Instances of the transportation problem where $m \approx n$ are called *symmetric* and accordingly, all instances for which the opposite holds are called *asymmetric*. More rigorously, we define:

Definition 1.2.1
Transportation problems are considered symmetric, *when* $m = \mathcal{O}(n)$ *holds and* asymmetric *in the opposite case.*

Moreover, to ensure a feasible problem, we make a further assumption on the vectors a and b:

Assumption 2
We assume that, the transportation problem is balanced, *i.e.* $\sum_{i=1}^m a_i = \sum_{j=1}^n b_j$.

If the problem is not balanced, there exists well known reformulation tech-

niques, see for example Gass (1969), which can be applied for the majority of applications and ensure that Assumption 2 is satisfied.

Hence, we have collected all the necessary prerequisites for the following three well known proposition, cf. Gass (1969):

Proposition 1.2.2
If Assumptions 1 and 2 hold, the feasible region of the transportation problem is non-empty, i.e. $S \neq \emptyset$.

Proof. Setting

$$x_{ij} := \frac{a_i \cdot b_j}{\sum\limits_{k \in I} a_k}$$

we immediately get $x_{ij} > 0$ since due to Assumption 1 all the a_i and b_j are positive. Furthermore, it also holds

$$\sum_{i \in I} x_{ij} = \sum_{i \in I} \frac{a_i \cdot b_j}{\sum_{k \in I} a_k} = b_j \frac{\sum_{i \in I} a_i}{\sum_{k \in I} a_k} = b_j$$

without any further requirements and

$$\sum_{j \in J} x_{ij} = \sum_{j \in J} \frac{a_i \cdot b_j}{\sum_{k \in I} a_k} = a_i \frac{\sum_{j \in J} b_j}{\sum_{k \in I} a_k} = a_i$$

since by Assumption 2 we have $\sum\limits_{k \in I} a_k = \sum\limits_{j \in J} b_j$. □

Proposition 1.2.3
The feasible region S *of the transportation problem is closed and bounded.*

Proof. Follows by quick inspection of the constraints of (TP). □

Combining these two propositions, we are able to guarantee optimal solutions whenever Assumptions 1 and 2 are satisfied.

Proposition 1.2.4
If Assumptions 1 and 2 hold, the problem allows for an optimal solution, i.e. $S^* \neq \emptyset$.

Proof. By Assumptions 1 and 2 we have that the feasible region is non-empty due to Proposition 1.2.2. Furthermore, the feasible region is closed and bounded by means of Proposition 1.2.3 and, since the linear objective function is continuous, (TP) admits an optimal solution (Weierstrass' Theorem). □

1.3 Linear program

In the following we will review the transportation problem in the context of linear programming and introduce the standard terminology and properties. It is well known that the constraint matrix M is of rank $m+n-1$, cf. Dantzig (1951) or Gass (1969). Hence, we will call a set $\mathcal{B} \subsetneq \{(i,j) \mid i \in I, j \in J\}$ with cardinality $|\mathcal{B}| = m+n-1$ a *basis* of the transportation problem whenever the columns[4] M_{in+j} corresponding to the index tuples in $\mathcal{B}$ are linearly independent. The associated *basis matrix* $M_{\mathcal{B}} := (M_{in+j} \mid (i,j) \in \mathcal{B})$ is obtained by removing all columns from M that are not in $\mathcal{B}$.

Accordingly, we call a solution x with $Mx = e$ a *basic solution* with respect to $\mathcal{B}$, whenever $x_{ij} \neq 0$ implies (i,j) is in $\mathcal{B}$. Equivalently, these solutions are uniquely defined by $M_{\mathcal{B}}x = e$. Moreover, they are called *feasible basic solutions* if x additionally satisfies $x \geq 0$. We speak of *degenerate* basic solutions, whenever $x_{ij} = 0$ for a least one (i,j) in $\mathcal{B}$.

1.3.1 The dual transportation problem

Dualizing (TP), we obtain the following linear program:

$$\begin{aligned} \max_{\substack{u\in\mathbb{R}^m \\ v\in\mathbb{R}^n}} \quad & \langle u, a\rangle + \langle v, b\rangle & \\ \text{s.t.} \quad & u_i + v_j \leq c_{ij} & \forall (i,j) \in I \times J. \end{aligned} \tag{DP}$$

In accordance with the primal notation, we will denote the dual feasible set by $Y(c)$, optimal dual solutions by (u^*, v^*) and the corresponding optimal value by f^* and, lastly, the set of optimal solutions by $Y^*(c,a,b)$.

[4] The numbering of the columns of the matrix is with respect to the definition of the vectors x and c.

In case of the dual problem the reader can verify by quick inspection that the problem is always feasible and the dual region is closed but not bounded in the general case. To show the existence of optimal solutions we rely on the propositions made for the primal problem together with Assumptions 1 and 2.

Proposition 1.3.1
If Assumptions 1 and 2 hold, the dual problem admits an optimal solution.

Proof. By Proposition 1.2.4, we have that the primal problem admits an optimal solution in the case that Assumptions 1 and 2 are satisfied. Then we have by strong linear duality (see below), that (DP) is feasible and admits an optimal solution. □

Corollary 1.3.2
A direct implication of Proposition 1.3.1 is that (DP) *is bounded whenever Assumptions 1 and 2 are satisfied.*

1.3.2 Linear duality and complementary slackness

Naturally, we have weak and strong linear duality, that is, for feasible primal solutions $x \in S$ and feasible dual solutions $(u, v) \in Y$ it holds

$$\langle c, x \rangle \geq \langle u, a \rangle + \langle v, b \rangle \tag{1.1}$$

and optimal primal and dual solutions $x^* \in S^*$ and $(u^*, v^*) \in Y^*$ satisfy

$$\langle c, x^* \rangle = \langle u^*, a \rangle + \langle v^*, b \rangle. \tag{1.2}$$

Furthermore, (TP) and (DP) additionally satisfy the linear complementary slackness conditions. Hereby, we follow the common notation for minimum cost flow problems and introduce the notion of *reduced costs*, cf. Ahuja et al. (1993), to denote the dual feasibility of a given dual solution (u, v):

$$c_{ij}^{uv} := c_{ij} - u_i - v_j. \tag{1.3}$$

Thereby, feasible primal and dual solutions $x \in S$ and $(u, v) \in Y$ are optimal if and only if it holds

$$\begin{aligned} x_{ij} > 0 &\Rightarrow c_{ij}^{uv} = 0 \\ c_{ij}^{uv} > 0 &\Rightarrow x_{ij} = 0 \end{aligned} \tag{1.4}$$

for all $(i, j) \in I \times J$.

1.4 Minimum cost flow problem

Several of the algorithms for the transportation problem permit a better intuition and – in some cases – more efficient solution techniques, when the transportation problem is viewed as a network flow problem on a bipartite graph. Furthermore, a large portion of the literature towards the transportation problem is written from the perspective of network flows. Hence, we will accept a bit of redundancy at this point and introduce the transportation problem as a special case of the (linear)[5] minimum cost flow problem. To this end, we will apply several concepts of graph theory, where we assume the reader to be familiar with basic terms and refer to Korte and Vygen (2007) for a detailed introduction.

1.4.1 The transportation problem as a minimum cost flow problem

In the context of graph theory and network flows, the transportation problem constitutes an *uncapacitated minimum cost flow problem on a complete bipartite undirected graph*. The underlying graph called the *transportation graph* $G_{tp} := (V_{tp}, E_{tp})$ is depicted in Figure 1.2. Its vertex set $V_{tp} := S \dot{\cup} T$ is given by the two disjoint sets $S = \{1, \ldots, m\}$ and $T = \{1', \ldots, n'\}$. The correspond-

[5] The transportation problem is a special case of the minimum cost flow problem with linear cost coefficients. To ensure a compact presentation, we will omit the prefix "linear" in the further course of this work and refer to this general problem as the "minimum cost flow problem".

ing undirected edge[6] set $E_{tp} := \{\overline{ij} \mid i \in S, j \in T\}$ of the complete graph is connecting each node of one of these sets to all nodes of the other set. The

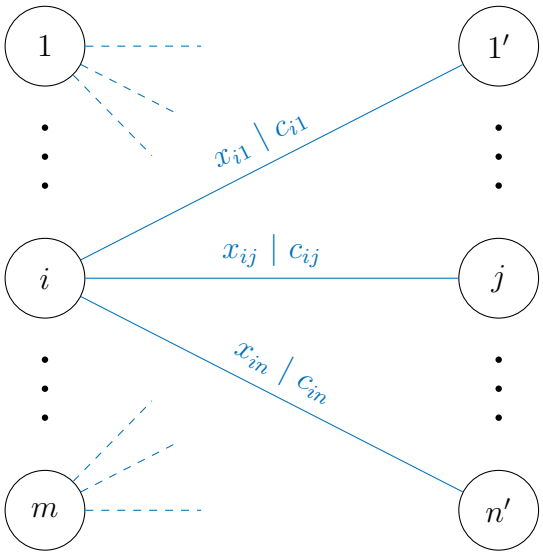


Figure 1.2: The transportation graph G_{tp} with the sources S on the left side and the sinks T on the right side. .

nodes $i \in S$ represent sources with supply a_i and the nodes $j \in T$ sinks with demand b_j respectively. Furthermore, we identify costs c_{ij} and a flow x_{ij} with each edge $\overline{ij}$.

Observe, that we have adapted the previously used notation by labeling the sinks, that is, the nodes in T, with an additional prime. This ensures that the nodes in S and T are distinguishable and thus enables us to provide a mathematically sound definition of the two disjoint sets S and T. Furthermore, this adaptation is particularly important with regard to the XTI representations presented in Section 4.3.1. However, in the interest of readability, we will refrain from using the prime when indices are denoted by variables, that is, we use j instead of j'. Moreover, the prime is always omitted in the notation of edges and flow or cost variables since here ambiguity is eliminated by the order

[6] In the context of this thesis, we will refer to undirected edges as *edges* and indicate them by a bar, i.e. $\overline{ij}$, whereas we will refer to directed edges as *arcs* and indicate those by an arrow, i.e. $\vec{ij}$.

of the two indices, i.e. the first index is always in S and the second one always in T. Thus, we write $\overline{ij}$, x_{ij} and c_{ij} instead of $\overline{ij'}$, $x_{ij'}$ and $c_{ij'}$ as well as e.g. $\overline{11}$, x_{11} and c_{11} instead of $\overline{11'}$, $x_{11'}$ and $c_{11'}$. Analogously we will omit the prime in the notation of dual variables, since e.g. u_1 clearly refers to 1 in S while v_1 is uniquely associated with $1'$ in T.

Now, the minimum cost flow problem is to determine non-negative flows x_{ij} at a minimum costs such that all supply of the sources is transported to the sinks where all demands are satisfied. By virtue of the foregoing definitions, this constitutes exactly the same linear problem as in (TP) with the exception that the sets I and J are now represented by S and T, i.e.

$$\begin{aligned} \min_{x\in\mathbb{R}^{mn}} \quad & \sum_{\overline{ij}\in E_{tp}} c_{ij}x_{ij} & & & \\ \text{s.t.} \quad & \sum_{j\in T} x_{ij} &= a_i & \forall i \in S \\ & \sum_{i\in S} x_{ij} &= b_j & \forall j \in T \\ & x_{ij} &\geq 0 & \forall \overline{ij} \in E_{tp}. \end{aligned} \tag{1.5}$$

In this context the row and column constraints are called *flow preservation constraints* with respect to a_i and b_j. Since the only other restriction on the flow variables is non-negativity and especially no upper bounds are imposed on the x_{ij}, the problem is called *uncapacitated*.

1.4.2 An equivalent representation of reduced costs

Observe that (1.5) can be equivalently modeled as a directed minimum cost flow problem, where the demands are represented by negative values, i.e. $-b_j$ and the direction of the arcs is modeled via representing the sinks by a -1 in the incidence matrix. This leads to the problem formulation

$$\begin{aligned} \min_{x\in\mathbb{R}^{mn}} \quad & \sum_{\overline{ij}\in E_{tp}} c_{ij}x_{ij} & & & \\ \text{s.t.} \quad & \sum_{j\in T} x_{ij} &= a_i & \forall i \in S \\ & \sum_{i\in S} -x_{ij} &= -b_j & \forall j \in T \\ & x_{ij} &\geq 0 & \forall \vec{ij} \in E_{tp}. \end{aligned} \tag{1.6}$$

Due to the bipartite structure of the transportation graph G_{tp}, the (1.5) and (1.6) are clearly equivalent[7] for the transportation problem, however, the transition to directed graphs implies a different representation of the corresponding dual constraints, i.e.

$$\begin{aligned} \max_{\substack{u\in\mathbb{R}^m \\ v\in\mathbb{R}^n}} \quad & \langle u, a\rangle - \langle v, b\rangle \\ \text{s.t.} \quad & u_i - v_j \leq c_{ij} \qquad \forall \vec{ij} \in E_{tp}. \end{aligned} \tag{1.7}$$

Accordingly, the reduced costs for an arc $\vec{ij}$, that is, the feasibility of a given dual solution (u, v) with respect to the constraint of $\vec{ij}$ in (1.7), are given by

$$c_{ij}^{uv} := c_{ij} - u_i + v_j. \tag{1.8}$$

This representation of the reduced costs offers some advantages with regard to the update of dual variables in the Network Simplex, which will be discussed in Section 4.3.3.

1.4.3 Basis representation in minimum cost flow problems

Before we continue, a few words on the further procedure. All statements made in the remainder of this section, and in particular Theorem 1.4.2, can be equivalently proven for general minimum cost flows on arbitrary connected graphs. However, since the focus of this work is on the transportation problem, we will present the following results explicitly for this problem. Besides, the more general case is covered thoroughly in the literature, therefore we refer the reader for example to Ahuja et al. (1993). A mathematical definition of minimum cost flow problems is given below in Section 1.4.3.

The main advantage of formulating the (TP) as a network flow problem is the characterization of bases as spanning trees on the bipartite graph G_{tp}. In preparation of the main theorem of this section, we will review the following definition:

[7] This can be easily verified by multiplying all flow preservations constraints with respect to b by -1.

Definition 1.4.1

Let G_{tp} be the transportation graph. A spanning tree of G_{tp} is a maximum acyclic subset $\mathcal{T} \subseteq E_{tp}$, that is, a set of edges connecting all nodes in V_{tp}, while none of its subsets form a circle in G_{tp}.

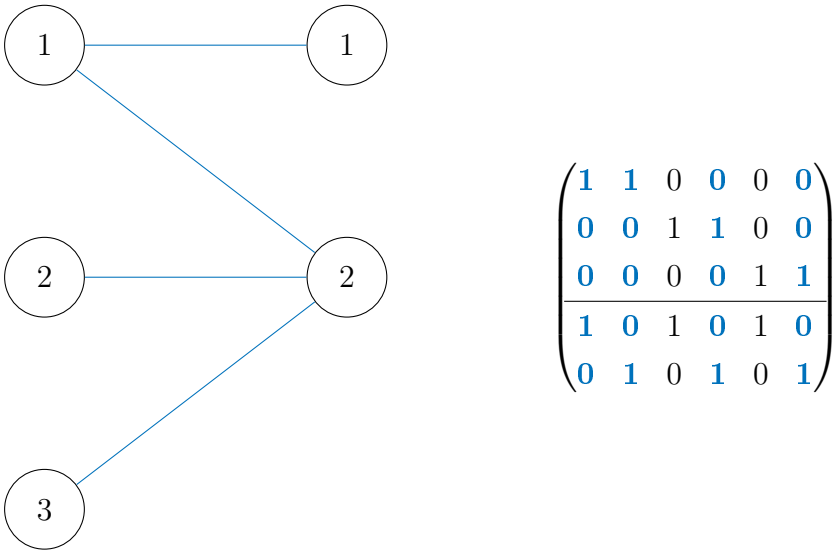


Figure 1.3: A spanning tree for a transportation graph with $m = 3$ sources and $n = 2$ sinks and the corresponding constraint matrix M. Each entry 1 in the first three rows of M represents a source, while each entry 1 in the last 2 rows represents a sink. The columns corresponding to the edges in the spanning tree are highlighted in blue. The edges $\overline{21}$ and $\overline{31}$ which are not contained in the tree correspond to the black columns of M.

In the next step, we will take a closer look at the constraint matrix M and revisit bases of the transportation problem. Observe that each non-zero entry in the first m rows of M corresponds to a source node of the problem and the equivalent statement applies to the last n rows of M and the sink nodes. More precisely, each column M_{in+j} can be identified with an edge $\overline{ij}$ of E_{tp} and vice versa. Thereby, the only non-zero entries of the column are two ones; the first at position i represents the source node and the second at position $m + j$ the respective sink node of the edge $\overline{ij}$. Consequently, in the context of graph

theory, the matrix M is called the *incidence matrix* (or *node-edge incidence matrix*).

Clearly, we can formally relate index tuples, edges and columns, i.e

$$\begin{array}{rccl} \phi: & I \times J & \to & E_{tp} \\ & (i,j) & \mapsto & \overline{ij} \end{array} \qquad (1.9)$$

$$\begin{array}{rccl} \psi: & E_{tp} & \to & \{\text{Columns of } M\} \\ & \overline{ij} & \mapsto & M_{in+j} \end{array}$$

where ϕ and ψ are bijective mappings. By definition, a basis of (TP) is an index set $\mathcal{B} \subsetneq I \times J$ corresponding to a maximum linear independent set of columns $\{M_{in+j} \mid (i,j) \in \mathcal{B}\}$ of M. By means of the following theorem, bases also define maximum acyclic edge sets, i.e. spanning trees, for the transportation graph G_{tp}.

Theorem 1.4.2
By mapping elements via the bisection (1.9) *there exists a one-to-one correspondence between bases of M and spanning trees of the transportation graph G_{tp}.*

Proof. This is a standard result for the incidence matrix of a graph which can be found for example in Ahuja et al. (1993). □

An equivalent property regarding a permutation triangulation of the basis matrix was already stated in Dantzig (1963).

Theorem 1.4.3
Given a basis $\mathcal{B}$ of the non-redundant formulation of the transportation problem, the rows and columns of the basis matrix $M_{\mathcal{B}}$ can be rearranged to be upper (or lower) triangular.

Proof. See for example Bradley et al. (1977). □

By means of these theorems, we will hereinafter call the corresponding spanning tree $\mathcal{T}_\mathcal{B}$ of a given basis $\mathcal{B}$ the *basis tree* of $\mathcal{B}$.

General minimum cost flow problems

For the sake of completeness and with regard to the analysis of the Network Simplex in Chapter 4, let us also state the mathematical definition of general minimum cost flow problems. Therefore we assume some arbitrary connected directed[8] graph $G = (V, \vec{E})$ with vertex set V and arc set $\vec{E}$, where $|V| = n$ and $|\vec{E}| = m$. In contrast to the undirected transportation graph, each arc $\vec{ij} \in \vec{E}$ has an explicit direction from node i to node j. By virtue of this characterization, we define the sets of incoming and outgoing arcs $I_k = \{i \in V \mid \vec{ik} \in \vec{E}\}$ and $O_k = \{j \in V \mid \vec{kj} \in \vec{E}\}$ at node $k \in V$. As in the transportation problem we denote the costs and current flow of each arc $\vec{ij} \in \vec{E}$ by c_{ij} and x_{ij}. Furthermore, the excess e_k describes the flow balance of a node $k \in V$. In case $e_k > 0$, it will be referred to as a supply and whenever $e_k < 0$ it represents a demand. Mathematically, the *uncapacitated minimum cost flow problem* is given as follows:

$$\begin{aligned} \min_{x \in \mathbb{R}} \quad & \sum_{\vec{ij} \in \vec{E}} c_{ij} x_{ij} & & \\ \text{s.t.} \quad & \sum_{j \in O_k} x_{kj} - \sum_{i \in I_k} x_{ik} = e_k & \forall k \in V & \qquad (\text{MCF}_u) \\ & x_{ij} \geq 0 & \forall \vec{ij} \in \vec{E}. & \end{aligned}$$

If in addition, upper bounds $o_{ij} > 0$ on the flow are introduced, we obtain the *capacitated minimum cost flow problem*:

$$\begin{aligned} \min_{x \in \mathbb{R}} \quad & \sum_{\vec{ij} \in \vec{E}} c_{ij} x_{ij} & & \\ \text{s.t.} \quad & \sum_{j \in O_k} x_{kj} - \sum_{i \in I_k} x_{ik} = e_k & \forall k \in V & \qquad (\text{MCF}_c) \\ & 0 \leq x_{ij} \leq o_{ij} & \forall \vec{ij} \in \vec{E}. & \end{aligned}$$

Note that we do not loose any generality by neglecting explicit lower bounds l_{ij} in the formulation, since in this case substituting x_{ij} by x'_{ij} where $x'_{ij} = x_{ij} - l_{ij}$

[8] This does not imply a loss of generality since undirected graphs can be transformed into equivalent directed graphs, cf. Ahuja et al. (1993).

yields an equivalent problem where the flow is bounded from below by zero, i.e. $x'_{ij} \geq 0$, cf. Ahuja et al. (1993).

Clearly, (MCF_u), represents a special case of (MCF_c), whereby we assume the upper bounds o_{ij} are large enough to be omitted in the optimization. Furthermore, (1.5), and thus also (TP), is a special case of (MCF_u).

Reduced costs Lastly, observe that dual variables π_k for $k \in V$ are often called *node potentials* in this context, and, by means of dualizing the problems, the reduced costs for (MCF_u) and (MCF_c) are given by $c^{\pi}_{ij} = c_{ij} - \pi_i + \pi_j$.

1.5 Properties related to the parameters (C, a, b)

After introducing basic assumptions on the input parameters (C, a, b) in Section 1.2.3, we elaborate on different properties of the input and discuss consequences in the following. These are essentially trivial observations collected for the reader's convenience. Therefore, in most cases, we omit the corresponding proofs.

Auxiliary notation

Before we begin, a brief note on the notation used: For simplicity, we denote by $\mathbf{1}^k$ the k-dimensional vector of ones, i.e.

$$\mathbf{1}^k \in \mathbb{R}^k \text{ and } \forall i \in \{1, \dots, k\} : \mathbf{1}^k_i = 1$$

and at times use this notation to write *the sum over a vector by means of a scalar product* instead of using the full sum sign, e.g.

$$\sum_{i=1}^{mn} x_i = \langle x, \mathbf{1}^{mn} \rangle.$$

Furthermore, the notation $w' = w + \beta$ for two vectors w and w' of equal dimension d and a scalar $\beta \in \mathbb{R}$ implies that *each component of w is increased by β*, i.e.

$$\forall i \in \{1, \dots, d\} : w'_i = w_i + \beta.$$

1.5.1 Feasible solutions

To begin with, we state two simple observations on feasible solutions of the problem.

Proposition 1.5.1
The sum over the components of any feasible solution $x \in S$ of a transportation problem (c, a, b) is equal to the sum of the components of the supply a or the demand b, i.e.

$$\langle x, \mathbf{1}^{mn} \rangle = \langle a, \mathbf{1}^{m} \rangle = \langle b, \mathbf{1}^{n} \rangle.$$

Proposition 1.5.2
Let $\lambda > 0$ be a positive scalar. Then vectors x and $\hat{x}$ in $\mathbb{R}^{m \times n}$, a and $\hat{a} \in \mathbb{R}^{m}_{\geq 0}$ and b and $\hat{b} \in \mathbb{R}^{n}_{\geq 0}$ satisfy

$$\begin{aligned} x \in S(a, b) &\iff \lambda x \in S(\lambda a, \lambda b) \\ x \in S(a, b) \text{ and } \hat{x} \in S(\hat{a}, \hat{b}) &\implies x + \hat{x} \in S(a + \hat{a}, b + \hat{b}). \end{aligned}$$

1.5.2 Invariant transformations

Let us advance to some simple propositions regarding invariant transformations of the input parameters of a given problem instance. In this context, "invariant" indicates that the respective transformation does not affect the essential structure of the optimal set, that is, optimal solutions and the optimal value either remain unchanged or can be easily computed, e.g. by means of an affine transformation, after solving the transformed problem:

Definition 1.5.3
A mapping $(c, a, b) \mapsto (\tilde{c}, \tilde{a}, \tilde{b})$ of the input parameters of the transportation problem is called an invariant transformation *if the optimal solutions of the original problem (c, a, b) are given by an affine transformation of the optimal solutions of the transformed problem $(\tilde{c}, \tilde{a}, \tilde{b})$ and the same holds for the corre-*

sponding optimal values, i.e. there exist scalars $\alpha \in \mathbb{R}$ *and* $\beta \in \mathbb{R}$ *such that*

$$S^*(c, a, b) = \alpha \cdot S^*(\tilde{c}, \tilde{a}, \tilde{b}) + \beta$$

and scalars $\gamma \in \mathbb{R}$ *and* $\delta \in \mathbb{R}$ *such that*

$$f^*(c, a, b) = \gamma \cdot f^*(\tilde{c}, \tilde{a}, \tilde{b}) + \delta.$$

In this case the transformed problem $(\tilde{c}, \tilde{a}, \tilde{b})$ *is called* equivalent *to the problem* (c, a, b).

First, we consider transformations of the cost coefficients of a given problem.

Proposition 1.5.4
Let $\alpha > 0$ *and* $\beta \in \mathbb{R}$ *be two scalars. Then a solution* $x \in \mathbb{R}^{m \times n}$ *is optimal for the transportation problem* (c, a, b) *if and only if it is optimal for the problem with transformed cost coefficients* $(\alpha c + \beta, a, b)$*, i.e.*

$$S^*(c, a, b) = S^*(\alpha c + \beta, a, b).$$

As stated in the next proposition, we obtain invariant transformations for the supply and demand vectors as well:

Proposition 1.5.5
Let $\lambda > 0$ *be a positive scalar. Then a solution* $x \in \mathbb{R}^{m \times n}$ *is optimal for the transportation problem* (c, a, b) *if and only if* λx *is optimal for the scaled problem* $(c, \lambda a, \lambda b)$*, i.e.*

$$\lambda S^*(c, a, b) = S^*(c, \lambda a, \lambda b).$$

Proof. Let $\lambda > 0$ and $\lambda x^* \in S^*(c, \lambda a, \lambda b)$ which is equivalent to $\langle \lambda x^*, c \rangle \leq \langle y, c \rangle$ for all $y \in S(\lambda a, \lambda b)$. Moreover, since for each y there exists x such that $x = \frac{y}{\lambda}$

we can substitute y by λx and apply Proposition 1.5.2 to achieve

$$\begin{aligned} & \langle \lambda x^*, c\rangle \leq \langle y, c\rangle && \forall y \in S(\lambda a, \lambda b) \\ \iff & \langle \lambda x^*, c\rangle \leq \langle \lambda x, c\rangle && \forall \lambda x \in S(\lambda a, \lambda b) \\ \iff & \langle \lambda x^*, c\rangle \leq \langle \lambda x, c\rangle && \forall x \in S(a, b) \\ \iff & \langle x^*, c\rangle \leq \langle x, c\rangle && \forall x \in S(a, b) \end{aligned}$$

which is equivalent to $x^* \in S^*(c, a, b)$. □

Observe that Propositions 1.5.4 and 1.5.5 provide the useful option to consider normalized problem instances by transforming arbitrary cost vectors $c \in \mathbb{R}^{m \times n}$ and balanced supplies $a > 0$ and demands $b > 0$ to study equivalent problems with e.g. cost coefficients in $[0, 1]$ and $\langle a, \mathbf{1}^m\rangle = \langle b, \mathbf{1}^n\rangle = 1$. This technique is for example applied in Schwinn and Werner (2018), where the parameters of different problems classes are normalized to ensure a consistent evaluation of the approximation quality of heuristics. The according transformation of the cost coefficients is shown in the following corollary:

Corollary 1.5.6
Let $c \in \mathbb{R}^{m \times n}$ be some cost vector. Then we obtain an invariant transformation $c \mapsto \tilde{c} \in [0, 1]^{m \times n}$ of the cost coefficients by subsequently applying the following operations:

1. *Set $\hat{c} := c - \beta$ where $\beta := \min\{0, \min\limits_{(i,j) \in I \times J} c_{ij}\}$.*
2. *Set $\tilde{c} := \frac{1}{\alpha} \cdot \hat{c}$ where $\alpha := \max\limits_{(i,j) \in I \times J} c_{ij}$.*

Proof. Observe, that after 1. we have $\hat{c} \geq 0$ and thus 2. yields $0 \leq \tilde{c} \leq 1$. □

1.5.3 The transportation paradox and non-negative cost coefficients

To supplement our investigations of the parameters (C, a, b), let us introduce a very interesting and yet underrepresented (in teaching, applications and literature) phenomenon called the transportation paradox. To this end, we formulate

a relaxation of the transportation problem in case of non-negative cost matrices $C \geq 0$. More specifically, we relax the row and column constraints by allowing the forwarder to optionally transport more than a_i or b_j from given origins i or to given destinations j by replacing all equality constraints of (TP) by inequality constraints:

$$\begin{array}{lrcl} \min\limits_{x \in \mathbb{R}^{mn}} & \langle c, x \rangle & & \\ \text{s.t.} & Ax & \geq & a \\ & Bx & \geq & b \\ & x & \geq & 0. \end{array} \tag{TP$_\geq$}$$

Accordingly, we adopt the notation for the original problem and add an additional subscript $\geq$, that is, we will use

$$(C, a, b)_\geq,\ S_\geq,\ S^*_\geq,\ x^*_\geq \text{ and } f^*_\geq$$

to refer to problem instances and feasible as well as optimal solutions of the relaxation. The corresponding dual program is then given by:

$$\begin{array}{lrcll} \max\limits_{\substack{u \in \mathbb{R}^m \\ v \in \mathbb{R}^n}} & \langle u, a \rangle + \langle v, b \rangle & & & \\ \text{s.t.} & u_i + v_j & \leq & c_{ij} & \forall (i,j) \in I \times J \\ & u_i & \geq & 0 & \forall i \in I \\ & v_j & \geq & 0 & \forall j \in J. \end{array} \tag{DP$_\geq$}$$

Here, we have adapted the notation of (DP) accordingly, i.e.

$$Y_\geq,\ Y^*_\geq \text{ and } (u^*, v^*)_\geq.$$

In contrast to (DP)[9], this relaxation has the nice property that the dual feasible region is bounded:

Proposition 1.5.7

For given input $(C \geq 0, a, b)$, the feasible region $Y_\geq(c)$ of (DP$_\geq$) is non-empty

[9] The feasible region of (DP) is not bounded in the general case: Consider the counterexample $m = n = 1$, $a = b = 1$ and $c \in \mathbb{R}^1$ where optimal dual solutions are given by the line $u + v = c$.

and compact. In particular, the dual variables are bounded by the following inequalities:

$$0 \leq u_i \leq \min_{1 \leq j \leq n} c_{ij} \quad \forall i \in I$$
$$0 \leq v_j \leq \min_{1 \leq i \leq m} c_{ij} \quad \forall j \in J.$$

Proof. Follows directly from the definition of $(\mathrm{DP}_{\geq})$. □

Let us now discuss, how optimal solutions of (TP) and $(\mathrm{TP}_{\geq})$ are related. Clearly, if there exists any negative cost coefficient $c_{ij} < 0$, the problem $(\mathrm{TP}_{\geq})$ is unbounded, since the inequality constraints allow arbitrary positive values of x_{ij} and thereby arbitrary negative objective values, since we have $c_{ij} \cdot x_{ij} < 0$. Thus, we assume $C \geq 0$ for the remainder of Section 1.5.3. Intuitively, one would assume that for given input data $(C \geq 0, a, b)$, optimal solutions of (TP) will not be dominated by optimal solutions of the relaxation $(\mathrm{TP}_{\geq})$, since the feasible region is extended only to solutions where the total amount of transported units is higher and each additionally transported unit is associated with non-negative transportation costs. However, the example in Figure 1.4 shows that this intuition is not met in the general case. To put it briefly, in the situation of Figure 1.4 one can decrease the overall transportation costs by increasing the total amount of transported units. This phenomenon has been known to the optimization community for quite some time and is called the *transportation paradox* or *more-for-less paradox*. Formally, it is defined as follows:

Definition 1.5.8
The transportation paradox *describes the situation where for fixed input data (C, a, b), the optimal value of $(\mathrm{TP}_{\geq})$ is strictly better than the optimal value of* (TP)*, i.e.*

$$f^*_{\geq}(C, a, b) < f^*(C, a, b).$$

This represents the main motivation to study the paradox from a practical point of view, since it may enable practitioners to improve the optimal solution of a problem instance by exceeding the required shipment quantities,

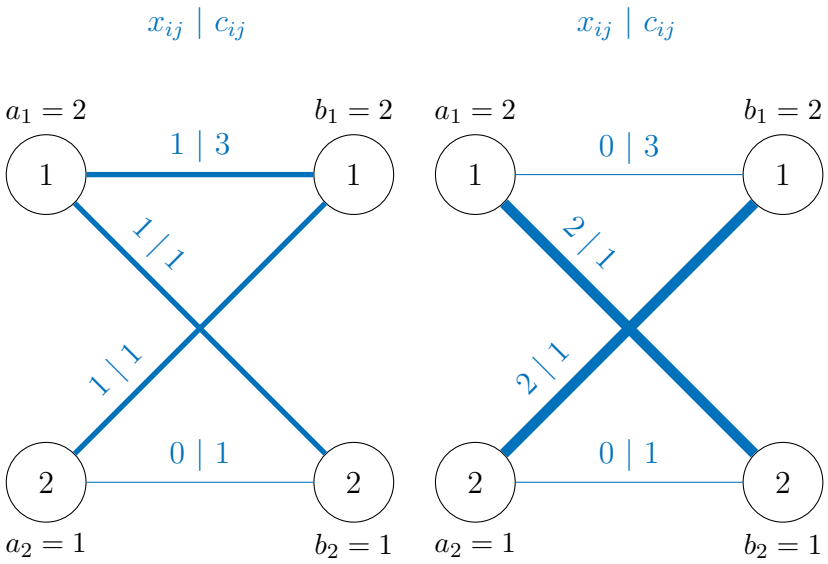


Figure 1.4: On the left side an optimal solution of an instance of (TP) is depicted where 3 units are transported with total costs of 5. On the right side an optimal solution of the relaxation ($\mathrm{TP}_{\geq}$) for the same input data is presented where 4 units are transported with total costs of 4. Thus the transportation paradox occurs.

i.e. relaxing (some) constraints in the problem. While some publications (e.g. Schrenk et al. (2011)) state that the paradox is due to Charnes and Klingman (1971) and Szwarc (1971), it is hinted in Charnes and Klingman (1971) and Deineko et al. (2003) that it had been known for example by A. Charnes and W.W. Cooper for quite a while at this point and was independently used as an exam problem by Alison Doig at the London School of Economics by 1959.

As we will see, the occurrence of the paradox, among other things, is linked to the composition of the cost matrix C. This connection is characterized in Deineko et al. (2003), where the notion of immune cost matrices is introduced:

Definition 1.5.9
A cost matrix C is defined as immune against the transportation paradox*, if for arbitrary supply and demands vectors $a \in \mathbb{R}^m_{>0}$ and $b \in \mathbb{R}^n_{>0}$ the transportation paradox does not occur.*

Their work is based on a structural property of C which we will call immune coefficients.

Definition 1.5.10
A cost matrix $C \in \mathbb{R}^{m \times n}$ is said to have immune coefficients *if and only if for all $1 \leq i, r \leq m$ and for all $1 \leq j, s \leq n$ where $i \neq r$ and $j \neq s$ it holds*

$$c_{ij} \leq c_{is} + c_{rj}.$$

The corresponding theorem yields a necessary and sufficient condition for the immunity of a given problem (C, a, b) to the paradox, which is independent of the row and supply vectors a and b.

Theorem 1.5.11
A cost matrix $C \geq 0$ is immune against the transportation paradox if and only if C has immune coefficients.

Proof. See Theorem 4 in Deineko et al. (2003). □

Interestingly, we can show that there is an invariant transformation for any cost matrix C such that the transformed matrix is immune:

Proposition 1.5.12
Let $C \in [0,1]^{m\times n}$ be a cost matrix with coefficients in $[0,1]$. Then the transformed matrix $\tilde{C}$, obtained by the invariant transformation

$$\tilde{c}_{ij} := \frac{1}{3}(c_{ij}+1)$$

for all $(i,j) \in I \times J$, is immune against the transportation paradox.

Proof. Clearly, the transformation defined above is invariant by means of Proposition 1.5.4 and the transformed coefficients are greater or equal than zero. Furthermore, let i, r and j, s be such that $1 \leq i, r \leq m$ and $1 \leq j, s \leq n$ where $i \neq r$ and $j \neq s$. By definition, we have that

$$\tilde{c}_{ij} \leq \tilde{c}_{is} + \tilde{c}_{rj}$$

is equivalent to

$$\frac{1}{3}(c_{ij}+1) \leq \frac{1}{3}(c_{is}+1+c_{rj}+1).$$

Reducing this expression, we obtain

$$c_{ij}+1 \leq c_{is}+c_{rj}+2,$$

which is obviously satisfied for $C \in [0,1]^{m\times n}$. Hence, $\tilde{C} \geq 0$ has immune coefficients and is therefore immune to the transportation paradox by Theorem 1.5.11. □

Thereby, we can apply Propositions 1.5.4 to achieve $[0,1]$-coefficients for arbitrary cost matrices and subsequently transform these by means of Proposition 1.5.12 to obtain equivalent problems, where the cost matrix is immune against the transportation paradox.

Corollary 1.5.13
Let $C \in \mathbb{R}^{m\times n}$ be a cost matrix. Then Corollary 1.5.6 and Proposition 1.5.12 provide an invariant transformation $C \mapsto \tilde{C}$ such that $\tilde{C}$ is immune against the transportation paradox.

In addition to considering immune coefficients one can characterize instances (C, a, b), where the paradox arises via the dual problem, as was first done by Szwarc (1971). For the following theorem, we consider the non-redundant form (see Section 1.3) of the transportation problem.

Theorem 1.5.14
For a given transportation problem (C, a, b) the transportation paradox arises if and only if in every optimal solution $(u^*, v^*) \in Y^*$ of the dual non-redundant formulation of the problem there exists a variable with negative value, i.e.

$$\exists i \in I : u_i^* < 0 \ \vee \ \exists j \in J : v_j^* < 0.$$

Proof. See Theorem 1 in Szwarc (1971). □

Finally, collecting the results of this section, we want to stress that – apart from its theoretical appeal – the transportation paradox gives a lot of desirable opportunities in the field of applications. Thereby, Theorem 1.5.14 gives an interesting option to practitioners who are willing to relax (some) constraints of their transportation problem. To this end, several numerical approaches have been proposed, see e.g. Szwarc (1971), Charnes and Klingman (1971), Finke (1978) and Schrenk et al. (2011). A numerical study done by Finke (1978) investigated randomly generated instances of the transportation problem and concluded that optimal solutions could be drastically improved by post-optimal exploitation of paradoxical structures. Averaged over all problem instances, optimal values could be improved by 18.6% when the amount of transported units was increased by 20.5%. Furthermore, aside from improving the optimal value, Theorem 1.5.11 gives the opportunity to design problem instances admitting or preventing the paradox, if an application permits some control over the input (C, a, b). Lastly, even if one has no control over the input or the option to relax constraints, knowing whether (TP) and ($\text{TP}_{\geq}$) have the same optimal value gives an interesting insight in the composition of the problem instance one is trying to solve. For example, Proposition 1.5.7 yields that the feasible region of ($\text{DP}_{\geq}$) is bounded.

1.5.4 Integer input data

In the following we state a series of properties holding for integer inputs.

Proposition 1.5.15
In case of integer supplies and demands a and b the transportation problem admits an optimal integer solution.

Proof. This is a standard result for general minimum cost flow problems and can be found for example in Ahuja et al. (1993). Hereby, the result is proved by construction using a cycle-canceling approach. Other proof techniques involve the unimodularity of the incidence matrix in minimum cost flow problems together with Cramer's rule (see e.g. Korte and Vygen (2007)) or triangular rearrangement of the constraint matrix, cf. Gass (1969). □

Proposition 1.5.16
In case of integer coefficients C the dual of the transportation problem admits an optimal integer solution.

Proof. This standard result for general minimum cost flow problems can also be found for example in Ahuja et al. (1993). Again, it is proved by construction using the fact that the Network Simplex will compute integer dual multipliers in every iteration whenever the costs are integral. □

1.5.5 Sparsity

Sparsity generally implies that a large portion of the entries in the *constraint* matrix of a problem assume zero values. For the transportation problem the density, i.e. the ratio of non-zeros to zeros, of the constraint matrix M is given by

$$\frac{2(m+n)}{mn(m+n)} = \frac{2}{mn},$$

which – for large enough values of m and n – will classify the problem as sparse. This fact along with the special structure of the matrix is implicitly exploited in the heuristics and the Transportation Simplex introduced later in this thesis.

In contrast to the constraint matrix M, the density of the cost matrix C is not determined by the problem dimensions but dependent on the specific application, i.e. the input parameters. As stated in the introduction of this chapter, we would like to point out that this thesis is concerned with dense instances of the cost matrix, that is C is not sparse in the general case. The reason for this is twofold:

- Sparsity in the objective function (cost matrix) will – in the general case – not eliminate the necessity to include these entries in the computation because positive transport on routes corresponding to zero costs might still affect the transport on routes with non-zero costs. Thus, exploitation of sparsity goes hand in hand with demanding a specific structure of the cost matrix and thereby reduce the generality of our analysis.
- If on the other hand, sparsity of the objective function relates to the fact that only a small number of arcs is *admissible* (or to use another terminology: a large portion of the arcs is *restricted*), a lot of the special structure motivating the isolated study of the problem is lost. As presented in Section 1.4, the transportation problem constitutes a special instance of the general minimum cost flow problem. In this context the essential feature of the transportation problem is that one operates on a *complete bipartite* graph. This imposes a special dependency structure (see the constraint (or incidence) matrix M) separating the problem from general instances. This structure justifies an isolated treatment and in particular makes it worthwhile to develop specially tailored algorithms. In case of a sparse problem it might be preferable to apply general minimum cost flow solvers optimized for sparse problems since – due to the vast number of applications – algorithms of this area are generally more developed and easier to acquire. However, there exist transportation algorithms where the worst-case complexity depends on the number of

arcs, e.g. Kleinschmidt and Schannath (1995), which will be discussed in Chapter 2 and may present an efficient alternative.

1.5.6 Degeneracy

Finally, let us consider the case of degeneracy. As stated before, we call a given basic solution x with corresponding basis B degenerate, whenever it holds $x_{ij} = 0$ for some $(i, j) \in B$. Occurrence of such solutions is not only of theoretical interest but a major practical concern since degenerate solutions can negatively affect the performance of solution algorithms. For instance, in the Simplex method, the phenomenon of *cycling*, that is an infinite repetitive sequence of degenerate pivots[10] may prevent algorithms from terminating after a finite number of iterations.

This obstacle is already addressed in Dantzig (1951) and usually solved by modifying the respective algorithms. For example, one can use the pivot rule of Bland in the Simplex Algorithm (cf. Luenberger and Ye (2015)) or introduce strongly feasible spanning trees in the Network Simplex (Ahuja et al. (1993)). A necessary and sufficient criterion for the occurrence of degeneracy with respect to the input vectors a and b is already provided by Dantzig (1951).

Theorem 1.5.17
A given transportation problem (C, a, b) admits degenerate solutions, if and only if there exist non-empty subsets $I' \subseteq I$ and $J' \subseteq J$ such that

$$\sum_{i \in I'} a_i = \sum_{j \in J'} b_j.$$

Proof. See Dantzig (1951) for the original proof or Schwinn (2015) for a proof using graph theory concepts. □

In addition to the theorem, Dantzig (1951) states a concept to avoid "degen-

[10] A pivot in the Simplex Method is said to be degenerate whenever it changes the current basis but not the current basic solution, see Section 4.3.5. In particular these steps do not improve the current objective value.

eracy“ problems independent of the algorithm used to solve the problem. The essential idea of this approach is to perturb the entries of a and b by small values in the order of the last valid decimal place to avoid equal partial sums of a_i and b_j.

1.6 History of the transportation problem

In the next section we give an overview of the historically most important contributions to the transportation problem. Many of these publications have had a direct impact on the development of mathematical optimization itself and represent major contributions in sub-areas such as linear optimization, network flows or non-smooth optimization. Furthermore, the importance of the problem is also evident in the quantity of publications, especially in the middle of the last century. Hence, to do justice to all relevant contributions and interesting stories concerning the problem would go beyond the scope of this section. We therefore point out that the following is only a brief account of the most important developments and refer to e.g. Dantzig (1963), Charnes et al. (1975), Gass (1990) and Schrijver (2002) for more historical background.

1.6.1 Exact solvers

The first formal treatment of the continuous formulation of the problem has been published as early as the end of the 18th century by Gaspard Monge, who studied the problem of transporting soil between two locations for the French Academy of Sciences, see Monge (1781). However, according to the current state of research, the first known publication in the context of combinatorial optimization appeared almost 150 years later in 1930. At that time A. N. Tolstoi studied several solution approaches applied to a problem of transporting goods in the railway network of the Soviet Union and was probably the first to consider the now well known cycle criterion for optimality[11], cf. Schrijver

[11] An optimum solution of a general minimum cost flow problem does not have any cycle of negative costs in its residual graph.

(2002). His work was followed by the well known publications of Kantorovich (1960)[12], who stated and analyzed the continuous version, and the statement of the problem formulation which is the subject of this thesis by Hitchcock (1941). Around the same time the problem was independently studied by Koopmans (1949) who, as Kantorovitch (1958), derived a cycle criterion for optimality.

Thereby, Hitchcock (1941) and Koopmans (1949) represent not only some of the first publications on the subject of linear programming but introduce many concepts that presaged the famous Simplex algorithm independently proposed by G.B. Dantzig in the late 1940s (cf. Dantzig (1990)). The importance of the transportation problem for research in the field of mathematical optimization at this time becomes even clearer by the fact that Dantzig (1951) adapted his method shortly thereafter for the transportation problem and thus initiated the development of the Network Simplex. We will introduce a modern version in Chapter 4 and refer to it as the Transportation Simplex or TPS in short. It is probably the most used algorithm for solving the transportation problem and will play a central role in this thesis.

After its original publication the TPS was reviewed and further developed in several papers, among which are Dennis (1958), Glicksman et al. (1960) who investigated coding techniques or Srinivasan and Thompson (1973) who analyzed different combinations of pivot updates and initialization methods. In particular, some major improvements which "contributed dramatically to improving the efficiency of network algorithms"(Glover et al. (1979)) were made with regard to base representation and pivot operations, see e.g. Glover and Klingman (1972), Glover et al. (1972), Srinivasan and Thompson (1972), Glover et al. (1974b), Glover et al. (1979) and Bradley et al. (1977). The practical importance of the algorithm can also be observed by the fact that Charnes and Cooper (1954) made a pedagogical approach to illustrate the operations of the Transportation Simplex to a broader audience, yielding the so called Stepping Stone Method. Furthermore, the TPS was extended towards general minimum cost flow problems, where it is called the Network Simplex. This brought with it a much wider field of possible applications and thereby a lot of

[12]This refers to the English translation of the original article, published 1939 in Russian.

further research. A good overview of the theoretical and practical results can be found in Ahuja et al. (1993) and a recent computational study in Kovács (2015).

Since the problem constitutes a minimum cost flow problem on a bipartite graph (cf. Section 1.4), several attempts to solve the problem using combinatorial techniques were made. Thereby, the importance of the transportation problem in the field of network flows is perhaps emphasized best by the fact that it repeatedly has been the subject of the work of L.R. Ford and D.R. Fulkerson (cf Ford and Fulkerson (1956), Ford and Fulkerson (1957a), Ford and Fulkerson (1957b) and Ford and Fulkerson (1962)). In the course of this, the problem was solved by computing maximal flows in networks (Ford and Fulkerson (1957b)), the Out-of-Kilter Method (Ford and Fulkerson (1957a)) or adaption of the Hungarian Method by Munkres (1957) as well as Ford and Fulkerson (1956).

Another interesting approach, which originated in a completely different area of optimization, was made by N.Z. Shor in the 1960s (cf. Shor et al. (1985)[13]), who – for the first time – applied non-smooth optimization methods to solve highly asymmetric instances of the transportation problem.

In practice, the problem was solved in the first half of the 20th century, mostly by general linear programming algorithms (canonically the problem constitutes an LP), combinatorial methods, as well as algorithms based on the Transportation Simplex. Amplified by the fact that computer systems were not as standardized as today and software could be accelerated significantly by optimizing it for specific large-scale computers, a vast number of codes was available, see Charnes et al. (1975). Based on computational results (e.g. Ford and Fulkerson (1962)) that indicated superiority of combinatorial methods, the general opinion (cf. Glover et al. (1974b)) at that time seemed to be that these methods were generally most efficient with regard to the transportation problem. This motivated the paper of Glover et al. (1974b), who aimed at providing a valid comparative study and refuting several notions that had become "part of the

[13]The original work was published in Russian with the translated title: An application of the method of gradient descent to the solution of the network transportation problem.

folklore around the transportation problem", concerning the efficiency of different solution techniques. Together with Srinivasan and Thompson (1973) their work "represents a landmark" (Gass (1990)) in the development and computational evaluation of computer-based transportation algorithms. It established the Transportation Simplex (meanwhile significantly sped up by the use of modern data structures for base representations) as state-of-the-art solver. An excellent overview of the development of algorithms until the end of the 20th century can be found in Gass (1990).

1.6.2 Heuristics

Apart from exact solvers, several heuristics have been proposed for the transportation problem. Amongst these are such well known names as the North-West Corner Rule (Dantzig (1951)), Vogel's Approximation Method (Reinfeld and Vogel (1958)) or Russel's Method (Russell (1969)). The Interest for studying these (primal) heuristics was twofold: While they were also employed to directly tackle the problem, the focus of research shifted towards their potential to produce starting points for the Transportation Simplex in the second half of the century. This was due to the significant acceleration and growing success of exact solvers and was accompanied by a greater interest in various Minimum Cost Allocation methods since these were reported to guarantee the best combined computation time of the heuristic and a successively run exact solver, cf. Srinivasan and Thompson (1973) and Glover et al. (1974b).

The most recent comparative studies constitute Gottschlich and Schuhmacher (2014) and Schwinn and Werner (2018). The first one considers problem instances derived from image science and compares the newly introduced Shortlist Method to other primal heuristics with respect to combined computation time. In contrast to this, the latter analysis considers general problems, that is various problem classes both derived from real world applications and artificially generated. Moreover, primal as well as dual heuristics are employed to directly solve the problem and the primal-dual gap is used to obtain a certificate for the quality of solutions. Thereby, – and to the best of our knowledge

– this paper provides the first comparative study of dual heuristics for the transportation problem.

1.6.3 Different names of the problem

We would like to conclude this section with a few words towards the various names of the problem. As a logical consequence of its long history, a multitude of designations for the problem have been established over the course of time. Apart from the (classical) transportation or distribution problem, these included almost all combinations of the surnames of the authors who made first contributions to the problem, which are in chronological order: G. Monge, A. N. Tolstoi, L. Kantorovich, F. L. Hitchcock, T. Koopmans and G. Dantzig (see above). In the context of linear programming and especially in the western world, it has been mostly referred to as the (classical) transportation problem or the Hitchcock transportation problem. Only later, as the earlier works of Kantorovich and Monge became known, the problem was also associated with their names.

1.7 Recent research and applications

Today, applications of the transportation problem – and the more general minimum cost flow problem – are widespread and continue to grow in scale. The respective application fields include operations research, financial mathematics, engineering, linguistics and computer science.

Historically, most problem instances originated in the area of operations research. For instance Tolstoi (see Schrijver (2002)) applied his methods to the Soviet railway network and Koopmans (1949) describes the problem of distributing empty cargo ships in overseas trade, see Figure 1.5. While problems of this nature remain relevant today, a large portion of the recent research has been driven by applications in the field of computer science.

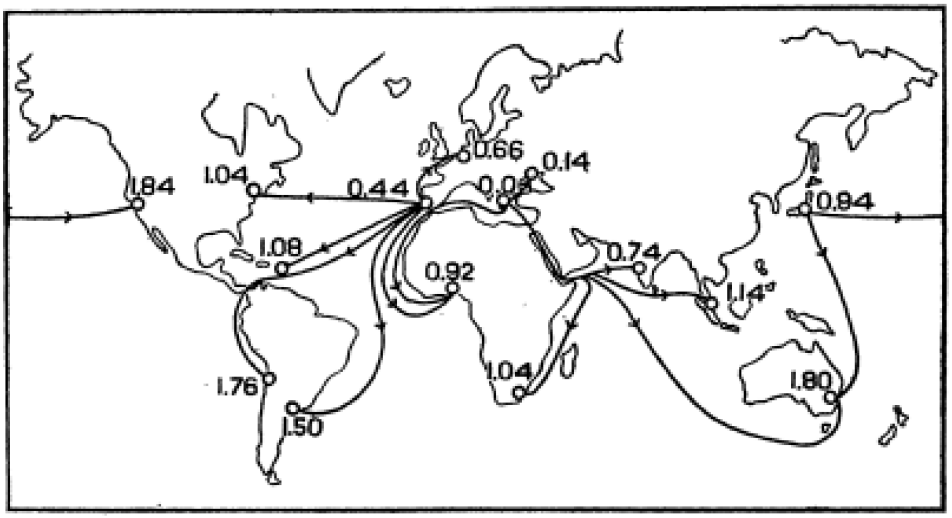


Figure 1. Optimal Routes of Empty Shipping Corresponding to World Dry-Cargo Flows in 1925.

The Figure shown with the representative port of each area represents the net shipping surplus of that area in millions of metric tons of dry-cargo capacity.

Figure 1.5: An illustration of an application of the transportation problem in overseas trade, taken from the paper of Koopmans (1949).

An important part of research originates in the field of imaging science and is related to the Wasserstein distance, see e.g. Rachev and Rüschendorf (1998) and Villani (2009). In the field of computer science it is known as the Earth Mover's distance, due to the seminal work of Rubner et al. (1997). Here, time critical computations in image processing motivated specifically tailored algorithms and respective comparative studies (Schrieber et al. (2017)), where some of these methods can also be applied to the general problem, e.g. the Shortlist Method of Gottschlich and Schuhmacher (2014). Applications of these algorithms include highly relevant tasks: e.g. visual object tracking, shape matching, image retrieval and fingerprint recognition. For an extensive list of applications, see Gottschlich and Schuhmacher (2014).

Initiated by the work of Cuturi (2013), Wasserstein distances and the theory of optimal transport have also taken their way into the machine learning community: Progress has been made in many areas, including among others the work of Schmitz et al. (2017) and Carrière et al. (2017) in (semi-)supervised learning, Arjovsky et al. (2017) for generative adversarial networks (GANs), Staib et al. (2017) and Srivastava et al. (2015) in Bayesian inference, Montavon et al. (2016) on restricted Boltzmann Machines or Frogner et al. (2015) for learning with Wasserstein objective functions. For an overview on recent research and applications, see e.g. Kolouri et al. (2017).

In addition, problem instances which are potentially highly asymmetric – e.g. the placement of VSLI chips (Brenner and Struzyna (2005)) – led to the development of combinatorial algorithms exploiting the unbalanced structure of the underlying bipartite graph by Kleinschmidt and Schannath (1995), Tokuyama and Nakano (1995) as well as Brenner (2008).

Beyond that, research on the transportation problem benefits from the more general field of network flow algorithms, since advancements in this area generally can be applied to the special case. Recent publications in this field, e.g. Kovács (2015), confirm the increasing competitiveness of modern solvers from the combinatorial optimization field. Adaptations to the transportation problem include the work of Kleinschmidt and Schannath (1995) and Tokuyama and Nakano (1995), which is based on the algorithm of Orlin (1988). Applica-

tions in the area of network flows are extensive; exemplary lists can be found for instance in Ahuja et al. (1993) or Kovács (2015).

Furthermore, some special cases of the transportation problem induced their own areas of research. Hereby, prominent cases include the assignment problem (see e.g. Burkard and Çela (1999)) or problem instances admitting Monge Sequences, see Hoffman (1963).

Lastly, another natural incentive for ongoing research is the occurrence of the transportation problem as a subproblem in other optimization problems like the traveling salesperson problem, the warehouse location problem or the fractional knapsack problem with weights, cf. Glover et al. (1974b), Brenner (2008) and Gottschlich and Schuhmacher (2014).

2 Experimental setup

In the subsequent Chapters 3, 4 and 5, we discuss two exact[1] solution methods for the transportation problem. Part of our analysis will be the empirical evaluation of these methods in Sections 4.5 and 5.3. For this purpose, we present our numerical test setup in this chapter, which is organized as follows: First, we specify the IDE and programming language, the external software used and the hardware specifications of our test system. To provide a first reference regarding the difficulty of the problem, this is followed by a discussion of the theoretical worst-case complexities of combinatorial solution methods. Finally, we introduce – in the main part of this chapter – several classes of test problems in Section 2.3.

2.1 Software choice and hardware used

In accordance with the research goals formulated in Chapters 1 and 5, all methods were implemented in MATLAB 2015B. Furthermore, we used the LP interfaces provided for MATLAB by ILOG CPLEX 12.6.2 and GUROBI 7.02 to represent external LP solvers. In particular we used the Network Simplex of CPLEX as a reference point for our methods. This Network Simplex is implemented in C/C++ which gives it an algorithm independent advantage over our MATLAB implementation. All numerical evaluations were performed on a standard personal computer (processor: Intel Core i5-4090, 3.30 GHz, RAM: 16GB).

[1] In Chapter 3, we consider heuristic approaches to the transportation problem which will serve as subroutines in the exact solution methods of the subsequent chapters.

2.2 Theoretical complexity analysis

With regard to the next chapters, we will further briefly review the available theoretical worst-case complexities of exact algorithms for the transportation problem. Our motivation is twofold: First, we provide a first criterion for the difficulty of general problem instances and, secondly, we will rule out certain heuristic approaches in Chapter 3 based on these complexities. To this end, we will – to the best of our knowledge – collect methods with the best strongly polynomial bounds in the following. It should be noted that all these algorithms assume integer input data.

For symmetric instances where the number of supplies equals the number of demands, i.e. $m = n$, the best available complexity is $\mathcal{O}(m^3 \log m)$ which can for example be obtained by directly applying the method of Orlin (1988) developed for general minimum cost flow problems. In the asymmetric case, i.e. $m > n$, this bound can be improved by shifting the complexity to the smaller set of nodes in the transportation graph. This fact has been exploited in several algorithms that provide efficient approaches for different ratios of $\frac{m}{n}$ (see Brenner (2008)). For instance, Kleinschmidt and Schannath (1995) achieve a running time of $\mathcal{O}\big(m \log m(k + n \log n)\big)$ where k is the number of admissible arcs by customizing the algorithm of Orlin (1988) to the transportation problem. As mentioned in Section 1.5.5 this bound is particularly interesting in the presence of sparsity. In the context of this thesis, we assume $k = m \cdot n$ (cf. Section 1.2) and consequently obtain a complexity of $\mathcal{O}(m^2 n \log m)$ for the algorithm of Kleinschmidt and Schannath (1995). For increasing values of $\frac{m}{n}$, additional methods improve this bound even further – e.g. Tokuyama and Nakano (1995) achieve a worst-case running time of $\mathcal{O}(mn^2 \log^2 m)$ and Brenner (2008) obtains a complexity of $\mathcal{O}\big(mn^2(\log m + n \log n)\big)$ by repeatedly solving minimum cost flow problems which only depend on the smaller side, i.e. scale with n. The most extreme case constitute algorithms which are linear in m, but exponential in n, see Brenner (2008).

2.3 Test problems

Before advancing to a detailed introduction of the solution methods for the transportation problem, we would like to introduce the test problems used to practically evaluate the corresponding algorithms. Based on the assumptions made in Chapter 1, we will begin by stating properties that apply to all test instances generated in our numerical studies and elaborate on the way random numbers are sampled. On this basis we then present three different classes of transportation problems: To begin with, we will introduce two classes of artificially generated problem instances called UNIFORM and SOLGEN. The first class represents the common approach for artificially generating problem instances, cf. Srinivasan and Thompson (1973), Glover et al. (1974b), Ahrens and Finke (1980) and Sandi (1986). In order to complement this choice we further incorporate the SOLGEN class, as the optimal solutions of these problems tend to be of a less uniformly distributed structure, cf. Section 4.5, and therefore pose a different challenge to solvers. In addition, a nice feature of the SOLGEN class is that a corresponding optimal solution is implicitly constructed in the generation process of a given problem instance. Lastly, these two artificial classes are supplemented with problem instances that originate from modern applications represented by DOTMARK, a benchmark for transportation problems from Schrieber et al. (2017), which offers 10 different problem classes derived from computational image comparison.

2.3.1 Assumptions on problem instances

A single instance of the transportation problem is represented by an input triple (C, a, b). Naturally, these instances should be feasible, thus, based on Assumptions 1 and 2 we will force

$$\sum_{i \in I} a_i = \sum_{j \in J} b_j, \tag{2.1a}$$

$$a_i \geq 0 \quad \forall i \in I, \tag{2.1b}$$

$$b_j \geq 0 \quad \forall j \in J, \tag{2.1c}$$

$$\exists i \in I : a_i > 0. \tag{2.1d}$$

for all problem instances included in our setup.

One can see that Assumption 1 is relaxed for our practical investigations since the DOTMARK instances include zero supplies and demands. Furthermore, numerically, we will consider all values which are smaller than some threshold $\epsilon = 10^{-12}$ as zeros. While this theoretically implies zero supplies and demands in the artificial problem classes as well, their occurrence in practice was negligible for the problem dimensions considered in this thesis.

The relaxation of Assumption 1 implies that the DOTMARK instances directly qualify as test instances. However, ensuring (2.1a-d) becomes more challenging while generating statistical sound values for a and b, which we will elaborate on in Section 2.3.2.

2.3.2 Sampling random numbers

Historically, the majority of numerical research on the transportation problem has been carried out on randomly generated problem instances, cf. Srinivasan and Thompson (1973), Glover et al. (1974b), Ahrens and Finke (1980) and Sandi (1986). The probability distribution most frequently used to sample the parameters (C, a, b) was the uniform distribution. Interestingly, experiments carried out by Ross et al. (1975), who compared different probability distributions for generating cost coefficients, indicated that the uniform distribution generally produces the hardest instances. Concerning our test instances, this motivated us to generate all cost coefficients by sampling from uniform distributions. In the following, we will denote the uniform distribution on the interval $[a, b]$ by $\mathcal{U}([a, b])$.

Moreover, we would also like to ensure uniformly distributed supply and demand vectors a and b. Here, for simplicity, we set

$$\sum_{i \in I} a_i = \sum_{j \in J} b_j = 1 \tag{2.2}$$

for our artificial instances. As noted by Ahrens and Finke (1980), statistically

sound values of a and b should therefore be uniformly distributed on the standard $(m-1)$-simplex and the standard $(n-1)$-simplex, respectively, to further satisfy equations (2.1b-d). To accomplish this, we sample the supply and demand vectors from a suitable Dirichlet distribution. Accordingly, the expected value and the variance of the components is given by $\frac{1}{m}$ and $\frac{m-1}{m^2(m+1)}$ for a as well as $\frac{1}{n}$ and $\frac{n-1}{n^2(n+1)}$ for b, cf. Balakrishnan and Nevzorov (2005). We proceed with a detailed description of the three problem classes.

2.3.3 UNIFORM

The first problem class represents the common approach for artificially generating problem instances for the transportation problem, cf. Srinivasan and Thompson (1973), Glover et al. (1974b), Ahrens and Finke (1980) and Sandi (1986). As explained in Section 2.3.2, we sample the supplies and demands a and b from m-dimensional and n-dimensional Dirichlet distributions to ensure uniformly distributed points on the respective standard simplices. Furthermore, we draw the cost coefficients c_{ij} from the uniform distribution on $[0, 1]$, i.e. $\mathcal{U}([0, 1])$, since, as stated above, uniform distributed costs have been reported to yield rather challenging problem instances.

2.3.4 SOLGEN

In addition to the UNIFORM class, we include a second problem generator, called SOLGEN, with the advantageous property that a corresponding optimal solution is implicitly created for each problem instance. Moreover, as can be observed in Section 4.5, the optimal solutions of SOLGEN instances tend to include variables with comparably high costs, which results in a problem structure that is less exploitable by certain pivot rules and therefore nicely complements our choice of problem classes. To construct these problems, we use similar concepts as presented in Arthur and Frendewey (1994) for capacitive minimum cost flow problems, who report that their algorithm produces problem instances of "comparable difficulty" to instances generated by NET-

GEN (Klingman et al. (1974)), a commonly used minimum cost flow problem generator.

The generation process follows a reverse methodology for the construction process. Instead of sampling input triples (C, a, b) with guaranteed but unknown optimal solution (or multiple optimal solutions), we start by generating an optimal solution, represented by a basis as well as the corresponding primal and dual basic solutions. Subsequently, we choose (C, a, b) such that optimality conditions for this solution are satisfied. Here, we improve the work of Arthur and Frendewey (1994) by ensuring a uniformly distributed topological structure of the corresponding optimal solution. To achieve this, we will employ random walks on the transportation graph G_{tp} to obtain uniformly distributed spanning trees. While we will illustrate our approach solely for the transportation problem, it *can easily be extended to general (capacitated) minimum cost flow problems*. We will add further statistical value and the possibility to replicate experiments by stating the specific probability distributions for all randomly generated values.

Now, let us present the algorithm for generating SOLGEN instances. To begin with, we will take an isolated glance on the consecutive steps of the construction process and give the complete algorithm at the end of this section. To supplement our explanations, we will illustrate the process for an exemplary problem instance in Figures 2.1, 2.2 and 2.3.

In the first step of the algorithm a random basis B is created:

1. Generate uniform distributed basis $B = \{(i_1, j_1), \ldots, (i_{m+n-1}, j_{m+n-1})\}$.

Hereby, one makes use of the one-to-one correspondence between bases and spanning trees on bipartite graphs (the transportation graph) given in Theorem 1.4.2. Taking advantage of this equivalence, Arthur and Frendewey (1994) propose to compute bases B via randomly generating such spanning trees. To this end, they propose a heuristic approach for which no statistical properties are proven. Here, we add to their work by realizing uniformly distributed spanning trees (and thereby bases) via random walks on the transportation graph. For a detailed description of how to obtain uniform distributed spanning trees

by employing random walks on general graphs, see e.g. Broder (1989).

As a next (or possibly parallel) step, we sample the dual variables u and v

2. Set $u_i := \mathcal{U}([-0.5, 0.5])$ for all $i = 1, \ldots, m$.
3. Set $v_j := \mathcal{U}([-0.5, 0.5])$ for all $j = 1, \ldots, n$.

where each component is sampled from the uniform distribution on $[-0.5, 0.5]$. These first three steps are collectively illustrated in Figure 2.1. Subsequently,

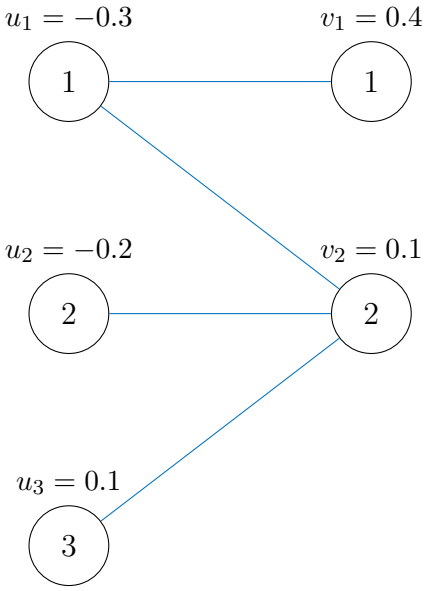


Figure 2.1: Depicted are steps 1. to 3. of an exemplary construction process of SOLGEN for $m = 3$ and $n = 2$. Initially a random spanning tree is created and the dual variables are sampled.

we uniformly sample a point $p = (p_1, \ldots, p_{m+n-1})^\top$ on the $(m+n-2)$-simplex by means of the Dirichlet distribution and iterate through all index tuples (i, j) in $I \times J$ to generate the primal variables and the cost coefficients for the problem:

4. Sample $p = (p_1, \ldots, p_{m+n-1})^\top$.
5. Set $c_{ij} := \begin{cases} u(i) + v(j) & \text{for } (i,j) \in B \\ u(i) + v(j) + \mathcal{U}([0,1]) & \text{otherwise} \end{cases}$

6. Set $x_{ij} \coloneqq \begin{cases} p_k & \text{for the } k\text{-th entry } (i_k, j_k) \text{ in } B \\ 0 & \text{otherwise} \end{cases}$

Hence, we sample cost coefficients such that the dual solution is feasible and the reduced costs corresponding to the index tuples in B are zero, i.e. the respective dual constraints are active. Dual degeneracy can be achieved by setting $c_{ij} \coloneqq u_i + v_j$ for a specified number of non-basic dual constraints[2]. Likewise, we set all non-basic primal variables to zero and sample non-negative values for the basic variables. Moreover, we can also determine the degree of degeneracy of the primal solution by setting specific basic variables to zero[3]. At this point, we have ensured that the complementary slackness conditions (1.4) are satisfied for x and (u, v). We illustrate a possible state of the exemplary problem instance after these operations in Figure 2.2.

Finally, we compute the supplies a and demands b by summing over the basic components of x and thereby guarantee primal feasibility of our solution and in particular (2.1a), i.e. $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$:

7. $a_i \coloneqq \sum_{i:(i,j)\in B} x_{ij}$.
8. $b_j \coloneqq \sum_{j:(i,j)\in B} x_{ij}$.

Moreover, as we have $\sum_{k=1}^{m+n-1} p_k = 1$ for the vector p created in Step 4, it further holds (2.2), i.e. $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j = 1$. Again, we present an example in Figure 2.3.

In summary, we have constructed a problem instance (C, a, b) together with a corresponding primal-dual optimal pair x and (u, v), since this solution is primal and dual feasible and satisfies complementary slackness conditions (1.4). Furthermore, we have ensured Equations (2.1a-d) and in particular (2.2) for our problem instance. Note that the entries of the cost matrix have been generated by summing three uniform distributions (i.e. an Irwin-Hall distribution), consequently their standard deviation will be lower compared to uniformly sampled cost coefficients.

[2] $u(i) + v(j) + \mathcal{U}([0, 1]) = c_{ij}$ for basic index-tuples occurs with probability zero.
[3] As in the dual case, $p_k = 0$ for some k occurs with probability zero.

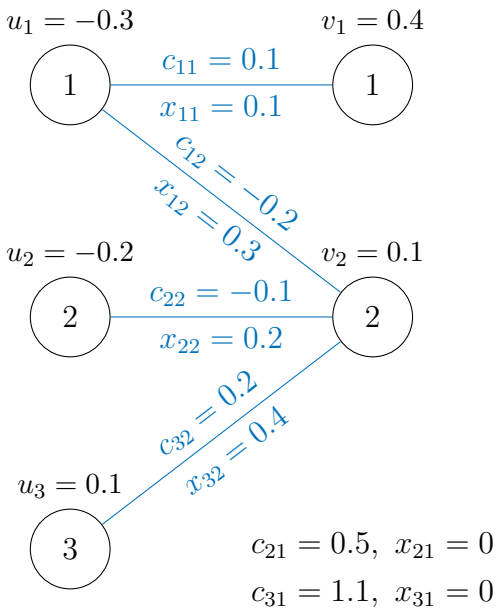


Figure 2.2: Depicted are steps 4. to 6. of an exemplary construction process of SOLGEN for $m = 3$ and $n = 2$. Thereby, the positive primal variables are drawn for the edges of the basis tree while all other primal variables are set to zero. Accordingly, the cost coefficients are set such that complementary conditions hold and the dual solution is feasible.

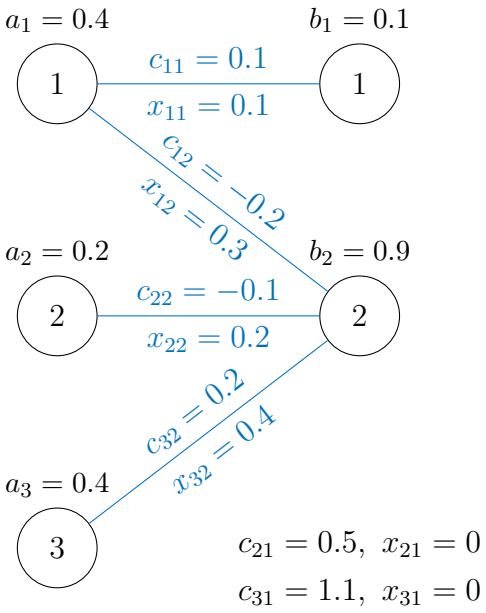


Figure 2.3: Depicted are steps 7. to 8. of an exemplary construction process of SOLGEN for $m = 3$ and $n = 2$. In the final steps, the supplies and demands are determined by summing over the basic primal variables.

We conclude with the complete algorithm for generating SOLGEN instances:

Algorithm 1 Generating SOLGEN instances

1. Generate uniform distributed basis $B = \{(i_1, j_1), \ldots, (i_{m+n-1}, j_{m+n-1})\}$.
2. Set $u_i := \mathcal{U}([-0.5, 0.5])$ for all $i = 1, \ldots, m$.
3. Set $v_j := \mathcal{U}([-0.5, 0.5])$ for all $j = 1, \ldots, n$.
4. Sample $p = (p_1, \ldots, p_{m+n-1})^\top$.
5. Set $c_{ij} := \begin{cases} u(i) + v(j) & \text{for } (i,j) \in B \\ u(i) + v(j) + \mathcal{U}([0,1]) & \text{otherwise} \end{cases}$.
6. Set $x_{ij} := \begin{cases} p_k & \text{for the } k\text{-th entry } (i_k, j_k) \text{ in } B \\ 0 & \text{otherwise} \end{cases}$
7. $a_i := \sum_{i:(i,j)\in B} x_{ij}$.
8. $b_j := \sum_{j:(i,j)\in B} x_{ij}$.

2.3.5 DOTMARK

This benchmark introduced in Schrieber et al. (2017) provides a collection of 10 different classes of black-and-white images. Each class contains 10 images which are given in different resolutions ranging from 32×32 to 512×512. As an example, we illustrate image classes 7 to 10 in Figure 2.4; for a detailed description we refer the reader to the website of the authors[4], where the benchmark is publicly accessible. Note that the first 6 image classes are, in fact, artificially generated. However, as the corresponding transportation problems still have specific cost matrices induced by the Wasserstein distance, they still represent a complementary problem class to the SOLGEN and UNIFORM classes.

[4] http://www.stochastik.math.uni-goettingen.de

On the basis of these image classes, Schrieber et al. (2017) evaluate the computation of the Wasserstein distance between two images P and P' of equal resolution which amounts to the solution of a symmetric, discrete[5] transportation problem. We will include these problems into our evaluation and show how

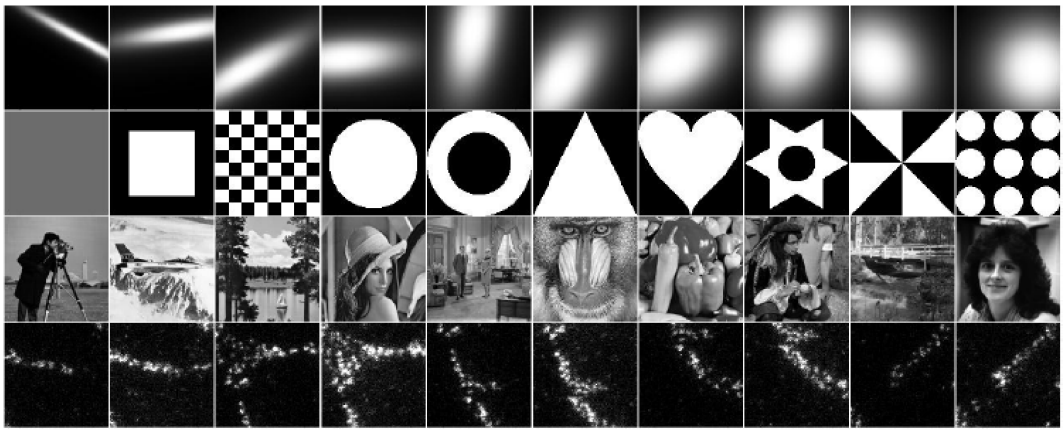


Figure 2.4: A depiction of DOTMARK classes 7 to 10 in resolution 128×128 taken from Schrieber et al. (2017).

the respective transportation problems are constructed: To this end, without loss of generality, each pixel in P represents an origin and each pixel in P' a destination in the transportation model where the corresponding supplies and demands are given by their gray values ranging from 0 to 255. Hence, we have non-negative integer values $a_i \in \mathbb{N}$ and $b_j \in \mathbb{N}$ for all $i \in I$ and $j \in J$. Furthermore, Schrieber et al. (2017) guarantee feasible problems (in our case explicitly Equation (2.1a)) by ensuring that the sum over all gray values are the same for any two images of the same resolution of a class. Since this sum is always positive, all DOTMARK instances satisfy (2.1a-d). To obtain a vectorized form of a and b, we sort the pixels of each image in, without loss of generality, column-wise order. Finally, we obtain the costs $c_{ij} \in \mathbb{N}$ of transporting one unit of the gray value between given origins i and destinations j by computing the squared euclidean distance of the respective pixel positions,

[5] All supplies, demands and costs assume integer values.

i.e.

$$c_{ij} = (p_1^i - p_1^j)^2 + (p_2^i - p_2^j)^2$$

where $p^i \in \mathbb{N}^2$ and $p^j \in \mathbb{N}^2$ denote the pixel position of i in P and j in P'. This results in a very specific structure of the cost matrix C which nicely supplements our artificially constructed problems. For instance, by design, all entries on the diagonal of C will be zero, since the distance between two pixels at the same position is always zero. Furthermore, observe that the DOTMARK instances may come with zero supplies and demands.

Asymmetric problem instances

So far, we have reviewed the construction process applied by Schrieber et al. (2017) to generate symmetric transportation problems from two images of the same resolution. In order to obtain an additional class of asymmetric transportation problem, we adapt their methodology for images of different resolution. Observe that this demands rescaling the of the supplies or demands, which accordingly are no longer integer in the general case, as well as a modification of the underlying cost function.

Assume without loss of generality that the image P has a higher[6] resolution $d \times d$ than the image P' with resolution $d' \times d'$. Then, the supplies and demands are generated as in the symmetric case, with the exception that b is scaled to b', i.e.

$$b' := \frac{\sum_{i=1}^{d^2} a_i}{\sum_{j=1}^{(d')^2} b_j} \cdot b$$

to ensure (2.1a).

Furthermore, in order to keep the calculation of the cost matrix $C \in \mathbb{N}^{d^2 \times (d')^2}$ consistent with the symmetric case, we group the pixels in P. Therefore, let

[6] By demanding that the resolution of P is always higher or equal to the resolution of P', we satisfy $m \geq n$ which is part of Assumption 1

$\delta := \frac{d}{d'} \in \mathbb{N}$ and sets Δ_k be given as

$$\forall k \in \{1, \ldots, d\} : \Delta_k = \{(k-1)\delta + 1, \ldots, k\delta\}.$$

Furthermore, a pixel at position $p = (p_1, p_2)$ in the finer image P is said to have coarse position (k, l), whenever

$$p_1 \in \Delta_k \quad \text{and} \quad p_2 \in \Delta_l$$

for some k and l in $\{1, \ldots, d\}$. Then, we set the pixel position of all pixels i in P to their coarse position, when computing the costs c_{ij}, $j \in J$. This ensures the consistency of the asymmetric costs and implies in particular that the optimal value of the transportation problem is zero, if it is generated from the same image in different resolutions.

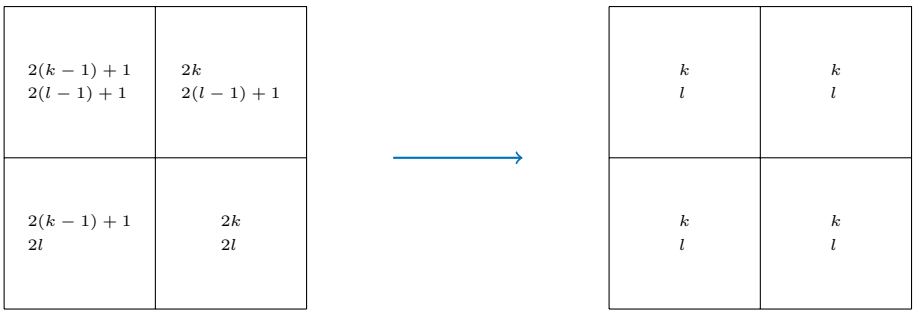


Figure 2.5: An example for determining coarse pixel positions when the ratio between the two resolutions is $\delta = 2$. The pixels are clustered into groups of four and the corresponding pixel positions are mapped to the coarse position.

Problem sizes

We generated *symmetric* transportation problems for all pairs of two different[7] images of a given class, which results in a total of $\binom{10}{2} = 45$ transportation

[7] Clearly, transportation problems could be generated by comparing identical images as well. However, to ensure comparability to Schrieber et al. (2017), we only compared different images.

problems per class when the resolutions are fixed. In terms of symmetric problems, we evaluated images of resolution 32×32, i.e. $m = n = 1024$ and 64×64, i.e. $m = n = 4096$ for all algorithms. Furthermore, we solved instances for resolution 128×128, i.e. $m = n = 16384$ only with our TPS implementation since the reference solver was not able solve these problem on our system instances due to memory limitations. Our test system (cf. Section 2.1) was not able to generate symmetric problem instances corresponding to resolutions 256×256 or higher. Since the main focus of our numerical analysis is on symmetric problems, we restricted our analysis to all pairs of two different images of a given class in resolutions 64×64 against 32×32 to create $\binom{10}{2} = 45$ *asymmetric* instances of the dimensions $m = 4096$ and $n = 1024$. All numerical evaluations are conducted in Section 4.5.

3 Heuristics for the transportation problem

Before we advance to the investigation of the Transportation Simplex let us include a short treatise of heuristic solution methods for the transportation problem. A heuristic constitutes an algorithmic approach that obtains a feasible solution for a given problem which is not guaranteed to be optimal. In case of the transportation problem, the possible application of such methods is twofold: Naturally, they can be applied to obtain approximate solutions to the problem in comparatively short time. The most popular application, however, is within the Transportation Simplex where the heuristics are applied to obtain an initial basic solution in the course of Phase I of the algorithm (see Section 4.3.2).

The first application is considered in the publication Schwinn and Werner (2018) of the author. This study inspects the simultaneous utilization of primal and dual heuristics to obtain quality certificates for the corresponding solutions based on weak duality. To this end, and to the best of our knowledge for the first time, several simple dual heuristics are applied. The corresponding numerical analysis demonstrated that in this way the optimal value for the SOLGEN instances can be approximated to a few percentage points, but no satisfactory results can be achieved for the UNIFORM and DOTMARK problems. However, since the inherent computing time of the heuristics is at most 10 % of the time of an exact solver, one could decide in certain applications after evaluating a primal and dual heuristics whether an exact solver is used. This approach is particularly suitable for the Transportation Simplex, since in any case a primal heuristic is used to compute the initial solution. Lastly, this

study shows (again) that the performance of the heuristics depends strongly on the selected class of transportation problems.

In the course of this work we will investigate heuristics with regard to the second application, that is, we will analyze their contribution in an efficient implementation of the Transportation Simplex in MATLAB presented in Chapter 4. As indicated in Section 1.6, a broad spectrum of heuristics has been developed for these purposes which offer a wide range of feasible (basis) solutions with the natural compromise between cpu time and quality, i.e. the proximity to an optimal solution. In particular, we will examine different starting heuristics in combination with new types of pivot rules in Chapter 4. Since the performance of pivot rules is generally dependent on the employed starting heuristic, cf. Srinivasan and Thompson (1973) or Glover et al. (1974b), we try to represent all relevant heuristics proposed for the problem. In order to ensure clarity in this process, we have preselected heuristics according to the following criteria:

- Considering the development of exact solvers and Network Simplex methods in particular, several proposed heuristics are no longer of practical relevance. Hence, we excluded any candidate where the execution of the heuristics takes a disproportionately long time compared to the time it takes to produce exact solutions. With respect to Section 2.2 this applies in particular to any method whose theoretical cpu time was worse than $\mathcal{O}(m^2 \log(m))$. Heuristics excluded by this criterion included methods such as Russell's method, Russell (1969), Houthakker's method of mutually preferred streams, Houthakker (1955), and Vogel's approximation method, Reinfeld and Vogel (1958), as well as some more recent adaptations of it.
- In the event that several heuristics employed similar concepts we decided to use one representative of these heuristics that produces the best results in terms of cpu time and solution quality. Among thereby excluded methods were the Two Smallest in a Row Method, and the (Alternating) Row Column Rule, cf. Srinivasan and Thompson (1973), which are represented by the Row Minimum Rule and the Modified Row Minimum Rule

and their column counterparts, cf. Srinivasan and Thompson (1973), as well as the Tree Minimum Rule.

- In accordance with Gottschlich and Schuhmacher (2014) and Schwinn and Werner (2018) we include the Matrix Minimum Rule, the Modified Russel's Method, Gottschlich and Schuhmacher (2014), and the Large Amount Least Cost Rule proposed by Lee[1] which are accelerated by iterating through the sorted lists of the cost coefficients instead of inspecting a gradually decreasing matrix.
- Finally, we restrict ourselves to *primal basic heuristics*, that is, heuristics which produce primal feasible basic solutions for (TP) and therefore qualify as starting procedures for the Transportation Simplex.

All methods presented in the following implement iterative approaches. They determine in each step a new element (i, j) of the basis along with the corresponding values of the variables x_{ij} until a primal feasible basic solutions for the transportation problem is obtained. Their respective approaches can be subdivided into three basic classes: The first class, which we will denote as *elimination heuristics*, consists of heuristics inspecting in each step a cell, row or column of the cost matrix to identify a (several) new basis element(s) (i, j). Subsequently, the determination of the value(s) of x_{ij} eliminates a row or column from further consideration and thereby reduces the effort to compute the remaining basis variables. The second class is represented by *greedy heuristics* which inspect tuples (i, j) according to the (modified) cost coefficients c_{ij}, sorted in ascending order. For each element (i, j) considered the maximum possible flow is assigned to the corresponding variable x_{ij}. In the degenerate case, this method is supplemented by an additional subroutine which is run after the greedy algorithm and adds basis variables with zero flow until a basis is obtained. The third and last class is represented by a single heuristic called *dual-to-primal rule* which constitutes a combination of a dual heuristic and a primal heuristic which minimize the complementary slackness with feasible

[1] This method was developed as part of a Master's thesis submitted at the Graduate School of Business at the University of California in 1968 see, e.g. Srinivasan and Thompson (1973). Unfortunately, the author was not able to gain access to the original work.

dual solution in order to obtain a primal basic solution.

3.1 Elimination heuristics

The following well known elimination heuristics (cf. Srinivasan and Thompson (1972), Glover et al. (1974b), Gottschlich and Schuhmacher (2014) and Schwinn and Werner (2018)) will be incorporated in our study.

- The North-West Corner Rule (NWCR), see Dantzig (1951).
- The Row Minimum Rule (RMR) and the Column Minimum Rule (CMR), see Srinivasan and Thompson (1972) and Glover et al. (1974b).
- The Modified Row Minimum Rule (MRMR) and the Modified Column Minimum Rule (MCMR), see Srinivasan and Thompson (1972) and Glover et al. (1974b).

While the NWCR is known to produce inefficient starting solutions in comparison with the other heuristics, it constitutes a suitable choice to measure the impact of the starting heuristics in general, cf. Section 4.5. Furthermore, our choice of classical heuristics is supplemented by an additional elimination rule stated in Schwinn and Werner (2018) which is motivated by the basis representation in the Transportation Simplex and will be presented in Section 3.1.

Tree Minimum Rule

As mentioned before our incentive in studying heuristics is to obtain "good" initial solutions for the Transportation Simplex. The key to the efficiency of this algorithm is a very intricate basis representation, which is called the XTI method and described in Section 4.3.1. Since this representation requires the identification of a depth-first traversal of the basis tree corresponding to the current basis (Section 1.4), it is separately obtained for the initial solution after it was computed by one of the heuristics presented in this chapter. This motivated a further heuristic called the Tree Minimum Rule (TMR) proposed

in Schwinn and Werner (2018), which considers the basis elements in a depth-search sequence with respect to the basis tree and thereby allows to integrate the computation of the basis representation into the execution of the heuristic. Moreover, minimum cost allocation rules, that is, the (MRMR), (MCMR) and the greedy methods were found to produce the best solutions with respect to the total running time of the Transportation Simplex on general problem instances (cf. Srinivasan and Thompson (1972), Glover et al. (1979) and Gass (1990)). Thus, the TMR was designed to mimic their selection procedure, i.e. it identifies basis elements by choosing minimum entries in rows and columns of the cost matrix.

Tree Minimum Rule (TMR) In the initial step of the TMR the tuple (i, j), corresponding to the minimum entry c_{ij} of the cost matrix, is selected. Afterwards the maximal possible flow $\min\{a_i, b_j\}$ is assigned to x_{ij} and a and b are updated accordingly, i.e.

1. $x_{ij} := \min\{a_i, b_j\}$,
2. $a_i := a_i - x_{ij}$,
3. $b_j := b_j - x_{ij}$.

Now, assume w.l.o.g. that $a_i < b_j$; this implies that the row i is excluded from further consideration since the available supply is already used. In the next step, the algorithm selects the minimum element c_{kj}, $k \neq i$ in the column j and likewise assigns maximum possible flow and updates a and b. In the contrary case, i.e. $a_i \geq b_j$[2], the algorithm subsequently considers the elements c_{ik}, $k \neq j$ of the row i. In this fashion, the algorithm alternately selects minimum entries of a row or column of the cost matrix until the corresponding supply or demand is exhausted. After $m + n - 1$ assignments have been made, a feasible basic solution is established. In the worst case the rule selects m elements of a single row in the solution process which implies inspecting $\sum_{k=1}^{m} k$ elements

[2]In the event that $a_i = b_j$, the row i is still considered in the next step where a zero flow is assigned. This procedure ensures that a basic solution is computed in the degenerate case.

and results in a theoretical complexity of $\mathcal{O}(m^2)$. Additionally, we illustrate in Figure 3.1 that, as intended, the basis entries are considered in a depth-search sequence with respect to the basis tree.

Lastly, let us give some implementation details: It is common practice to accelerate the algorithm by maintaining lists of not yet exhausted supplies and demands, cf. Srinivasan and Thompson (1973), Glover et al. (1974b) and Schwinn and Werner (2018).

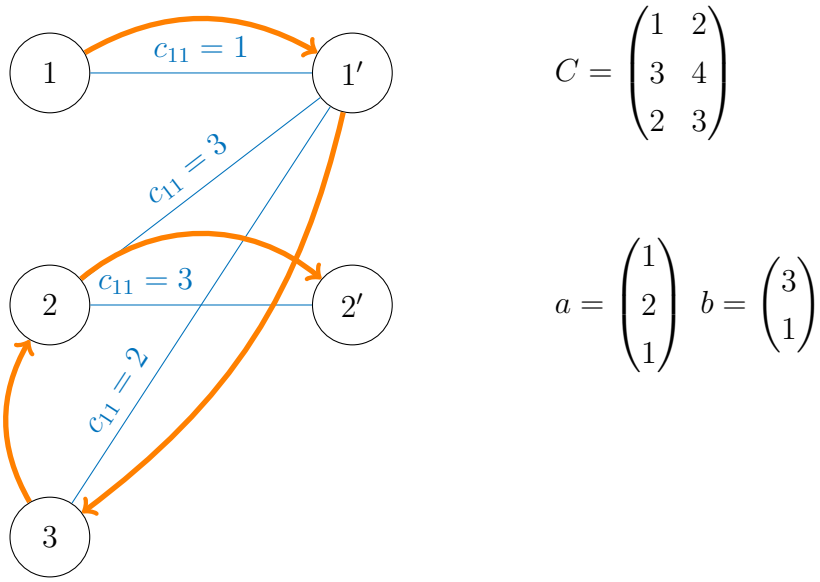


Figure 3.1: Consider the transportation problem depicted above with given input parameters C, a and b for dimensions $m = 3$ and $n = 2$. Indicated by the orange arrows is the order in which the TMR selects its basis elements, starting with the tuple $(1, 1)$. Observe that the orange arrows define a depth-first traversal of the corresponding basis tree starting in the root node 1.

3.2 Greedy heuristics

From the class of greedy heuristics we choose the following three methods, cf. Schwinn and Werner (2018).

- The Matrix Minimum Rule (MMR), see e.g. Srinivasan and Thompson (1972) and Glover et al. (1974b).
- The Modified Russel's Rule (MRUR) by Gottschlich and Schuhmacher (2014).
- The Large Amount Least Cost (LALC) by Lee[3].

3.3 The Dual-To-Primal Rule

As described in Schwinn and Werner (2018), we will also employ a heuristic that includes information from a feasible dual solution in the computation of a primal basic solution by means of the Matrix Minimum Rule. The idea is to initially employ a dual heuristic to obtain a feasible dual solution (u, v) and subsequently apply the MMR to the reduced cost matrix $C^{uv} := (c_{ij}^{uv})_{i,j=1}^{m,n}$ to approximate the complementary slackness conditions (1.4) by minimizing $\sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} c_{ij}^{uv}$. On the base of the numerical results of Schwinn and Werner (2018), we will choose the Pull Push Method (PPM) proposed in the same publication for generating the dual solution. The corresponding method, i.e. the application of the MMR to a dual solution obtained by the PPM, will be called *Dual-To-Primal Rule* or D2PR for short.

[3]See footnote 1.

3.4 Theoretical complexities

Finally, table 3.1 gives an overview of the included heuristics with the corresponding theoretical worst-case complexities.

Heuristics	Abbreviation	**Complexity**
North-West Corner Rule	NWCR	$\mathcal{O}(m)$
Row Minimum Rules	RMR, MRMR	$\mathcal{O}(mn)$
Column Minimum Rules	CMR, MCMR	$\mathcal{O}(m^2)$
Tree Minimum Rule	TMR	$\mathcal{O}(m^2)$
Matrix Minimum Rule	MMR	$\mathcal{O}(mn \log m)$
Modified Russel's Rule	MRUR	$\mathcal{O}(mn \log m)$
Large Amount Least Cost Rule	LALC	$\mathcal{O}(m \log m)$
Dual-To-Primal Rule	D2PR	$\mathcal{O}(mn \log m)$

Table 3.1: Overview of the worst-case complexities of all presented heuristics, cf. Schwinn and Werner (2018). Note that in the D2PR the complexity of the PPM is dominated by the MMR.

3.5 Implementation in Matlab

To conclude the chapter, a few notes on the implementation of the heuristics in MATLAB: To this end, note that MATLAB uses *column-major memory*, that is, in a multidimensional array, the entries of the left-most index are stored directly one after the other and can therefore be processed faster than the entries of the remaining indices. In particular, we observed a faster iteration over columns of a symmetrical matrix than over rows. Hence, as evidenced by our numerical studies, the Row Minimum Rules RMR and MRMR are most effectively implemented as their column equivalents on the transposed problem, i.e. on $(C^\top, b, a)$ instead of (C, a, b). However, we would like to point out that this procedure is implicitly handled in the implementation and therefore the assumption $m \geq n$ still applies.

Moreover, although all methods were implemented to the best of our knowledge, for the Large Amount Least Cost Method we only achieve a complexity of $\mathcal{O}(m^2 \log m)$ instead of $\mathcal{O}(m \log m)$. The reason for this efficiency loss is due to the lack of pointers in MATLAB, which impedes a proper efficient implementation of priority lists as for instance Fibonacci Heaps.

4 The Transportation Simplex

In this chapter we introduce the TPS, which is the basis for our algorithmic investigations of the transportation problem. It is structured as follows: We precede our explanations by reviewing the essential operations of the Simplex algorithm for general linear programs (LPS). Thereafter, we present a comprehensive description of our implementation of the TPS in Section 4.3. Since the differences between TPS and NS are very subtle, we finally point out how the individual components of the TPS would be adapted for the development of a general NS in Section 4.4.

4.1 Standard Simplex for linear programming (LPS)

Let us briefly recall the basic concept of the LPS. This is done for the sole purpose of introducing the structure of the algorithm which is inherited by the NS as well as the TPS. Accordingly, we refrain from a detailed technical introduction and only provide a short treatise of the main algorithmic components. For a comprehensive presentation we refer the reader to Luenberger and Ye (2015).

Let us further assume that a general linear program (LP) is given. Then, the execution of the Simplex algorithm is divided into two main phases: First, an initialization procedure called *Phase I* is executed to obtain a feasible basic solution for the LP or determine the infeasibility of the problem. Subsequently, in the event that a basic solution could be found, *Phase II* of the algorithm is invoked to gradually improve this solution until optimality is reached or

unboundedness of (LP) is observed.

Generally, the observation of infeasibility after Phase I or unboundedness of the problem after Phase II, are both possible outcomes. Note, however, that these cases are excluded when considering (TP), since due to the assumptions made in Chapter 1, we assume feasible and bounded problem instances. Thus, as the sole incentive in studying Simplex algorithms is in solving (TP), we will henceforth restrict our explanations to the feasible and bounded case.

4.1.1 Phase I

The initialization process of LPS consists in finding a basis $\mathcal{B}$ as well as a primal feasible basic solution x and a not necessarily feasible dual solution (u, v) such that the reduced costs c_{ij}^{uv} corresponding to basic index tuples $(i, j) \in \mathcal{B}$ are zero. Since by definition, all non-basic primal variables of a basic solution are zero, this implies that the complementary slackness conditions (1.4) for x and (u, v) are satisfied. Thereby (u, v) along with c^{uv} allow to determine a beneficial modification of the current primal solution x, as described in Phase II. In this context (u, v) are often called *dual multipliers* (cf. Luenberger and Ye (2015)); hence, in the remainder of this work, we will also refer to (u, v) as the *corresponding dual multipliers* of the primal solution x to emphasize the fact that (u, v) was obtained as described above. Apart from that we will naturally refer to (u, v) as the dual (basic) solution and to (x, u, v) as the primal-dual (basic) solution. Observe that throughout the algorithm, i.e. Phase I as well as Phase II, x is guaranteed to be a primal feasible basic solution whereas (u, v) is a not necessarily feasible dual basic solution.

In general, the initial solution is computed by solving an auxiliary problem with Phase II of the algorithm. This problem is constructed such that an initial solution is known and every optimal solution yields an initial feasible solution for the original problem.

4.1.2 Phase II

Afterwards, the algorithm starts an iterative execution of so-called *pivot steps* in Phase II. In each individual pivot step, initially reduced costs of the primal-dual pair (x, u, v) are evaluated by means of a *pivot rule*. Whenever all reduced costs are non-negative, the dual multipliers (u, v) are a feasible dual solution and since the complementary slackness conditions are satisfied by construction, this implies that an optimal solution was found. In the contrary case, a non-basic element, called the *entering element*[1] (i_e, j_e) with negative reduced costs $c^{uv}_{i_e,j_e} < 0$ is selected to enter the basis $\mathcal{B}$. In case there exist multiple eligible candidates, the selection of the element depends on the implementation of the specific pivot rule. This specific implementation is a key component of the algorithm and will therefore be separately considered in Section 4.3.4. Subsequently, the *leaving element* (i_l, j_l) is identified and replaced by (i_e, i_e) in the basis $\mathcal{B}$ and the corresponding primal-dual pair (x, u, v) is updated accordingly. More precisely, the new primal feasible basic solution is computed such that

$$M_{\mathcal{B}} x = e \quad \text{and} \quad x_{ij} > 0 \Rightarrow (i, j) \in \mathcal{B}$$

which in particular implies $x_{i_e,j_e} > 0$ and $x_{i_l,j_l} = 0$ and the dual multipliers (u, v) are adapted such that the complementary slackness conditions (1.4) are again satisfied. Written in a more compact form, the consecutive operations of a pivot step are given in Algorithm 2.

The existence of an optimal basic solution for feasible and bounded problems is guaranteed by the fundamental theorem of linear programming. Assuming a careful implementation[2], the algorithm will compute such a solution in finite time and in particular avoid the phenomenon of cycling, see Section 1.5.6 and

[1] In the literature the terms entering *variable* and leaving *variable* are more widely used, but in the course of the TPS (i, j) will be understood as the edge $\overline{(i, j)}$ and in particular we consider entering and leaving edges. To make this transition clearer, we use the terms entering *element* and leaving *element* in the LPS.

[2] For the LPS, finiteness can be guaranteed for example by employing the pivot rule of Bland. In case of the TPS, we introduce the concept of strongly feasible spanning trees in Section 4.3.5.

Algorithm 2 Pivot step in the Simplex

1. Compute reduced costs c_{ij}^{uv} of the current primal-dual basic solution (x, u, v) by means of a pivot rule resulting in one of the two following alternatives:
   i) Selection of an entering element (i_e, j_e) with $c_{i_e,j_e}^{uv} < 0$.
   ii) Observation of optimality, i.e. $c_{ij}^{uv} \geq 0$ for all $i \in I$ and $j \in J$ and thereby termination of the algorithm.
2. Determine the leaving element (i_l, j_l).
3. Update the basis $\mathcal{B}$, the primal feasible basic solution x and the corresponding dual multipliers (u, v).

e.g. Luenberger and Ye (2015).

4.2 Transition to minimum cost flow problems

Since general minimum cost flow problems define a specific class of linear programs, we could tackle these problems directly using the LPS. However, this approach completely ignores the underlying network structure of the problems and will not be competitive to specialized combinatorial solvers (cf. Ahuja et al. (1993)).

The crucial part of the Simplex is the execution of the pivot steps in Phase II. In the LPS, all components of a pivot step are carried out as linear algebra operations involving the basis matrix $M_{\mathcal{B}}$. In particular, the computation of the primal and dual variables generally includes the solution of linear equations. Although decomposition techniques and intricate update rules for the basis matrix $M_{\mathcal{B}}$ (or its inverse) have accelerated the process, it still remains the most expensive part of the update. For more details, we again refer to Luenberger and Ye (2015).

The major structural advantage of general minimum cost flow problems con-

stitutes the one-to-one correspondence between bases and spanning trees. By means of Theorem 1.4.2[3] one can replace the representation of the basis matrix by the corresponding basis tree and perform the bulk of the pivot step operations directly on the underlying graph. If the relevant information is stored in adequate data structures, this allows a substantial acceleration of the iterations.

Furthermore, observe that this transition from matrix representation to spanning trees has no effect on the basic sequence of Phase II given in Algorithm 2 but only affects the realization of its components, i.e. the pivot operations described in Algorithm 2, which are carried out as graph-theoretical operations.

4.3 Implementation of the Transportation Simplex (TPS)

In the following we state the implementation of the TPS. Before we begin, let us give a brief outline of this section. The main algorithmic components of the TPS and the NS are the same; apart from the realization of Phase I and the choice of pivot rules, the differences are very subtle. Moreover, the literature already provides excellent descriptions of the NS and its implementation, see e.g. Ahuja et al. (1993) or Kovács (2015). Thereby, starting with Section 4.3.1, we directly introduce and illustrate the TPS for the transportation problem and cover the differences to the NS later in Section 4.4. Additionally, to maintain a clear presentation in the process, we omit classical proofs of correctness for the executed operations. This is in particular with regard to the fact that the according statements constitute special cases of their equivalents for the NS and thus are proven implicitly for example in Ahuja et al. (1993).

Naturally, we will hereinafter equivalently use LP terminology and the network

[3]Since the focus is on transportation problems, this theorem and the according definitions were explicitly given and illustrated for (TP). However, as indicated in Section 1.4, the equivalent statements hold for (MCF_u) and (MCF_c) and are stated for example in Ahuja et al. (1993).

terminology introduced in Section 1.4. To this end, the reader may in particular review the definition of the transportation graph G_{tp}. Lastly, we would like to explicitly emphasize that all further explanations are based on the equivalent formulation of the dual problem (1.7) introduced in Section 1.4.2. In particular, we adopt the representation of reduced costs given in (1.8), that is we assume

$$c_{ij}^{uv} := c_{ij} - u_i + v_j.$$

for the remainder of Chapter 4. The motivation for this choice is given in Section 4.3.3.

we begin our description of the TPS by introducing the basic representation in Section 4.3.1, since it constitutes the crucial component of the method. Subsequently, we cover Phase I and Phase II of the algorithm and proceed with a discussion of promising pivot rules. Finally, we discuss a measure to prevent cycling and reduce the number of degenerated pivot steps and summarize the TPS in Algorithm 3.

4.3.1 Basis representation

As stated above, Theorem 1.4.2 allows to represent the basis matrix $M_{\mathcal{B}}$ by the basis tree $\mathcal{T}_{\mathcal{B}}$ in the TPS. Thereby, the storage scheme along with the update procedures of $\mathcal{T}_{\mathcal{B}}$ and the corresponding primal and dual solutions x and (u, v) represent the major advantage in comparison to the LPS. To this end, researchers have proposed several concepts, cf. Glover and Klingman (1972), Glover et al. (1972), Srinivasan and Thompson (1972), Glover et al. (1974b), Glover et al. (1979) and Bradley et al. (1977) or Ahuja et al. (1993) for an overview. Recent computational investigations done by Kovács (2015) for the NS reported a substantially superior performance of the XTI (eXtended Threaded Index) labeling method suggested by Glover et al. (1979) over the more commonly used ATI (Augmented Threaded Index) method (Glover et al. (1972) and Glover et al. (1974c)) for basic representation. These observations can be directly applied to the TPS, as the representation of the basis tree and all operations performed on it, i.e. pivot operations 2 and 3, are equivalent in

the TPS and the NS, see Section 4.4. we therefore follow their recommendation and implement the XTI method, which will be explained in more detail below.

XTI representation

Before we present the components of the XTI technique, let us give some preparatory definitions and notation. In addition, let us point out that all following definitions will be with respect to the transportation graph $G_{tp} := (V_{tp}, E_{tp})$.

The commonly used methods for spanning tree representation are based on *rooted trees* (cf. Glover et al. (1979) and Ahuja et al. (1993)). Thus, a designated *root node* $q \in V_{tp}$ for the spanning tree is chosen. Furthermore, in our explanations, we assume that q is the uppermost node in a graphical representation of the tree with the remaining $m + n - 1$ nodes and $m + n - 1$ edges hanging below. On the basis of the above characterization, we introduce the auxiliary definition of *subtrees*; thus, we denote by $\mathbf{T}(w) \subseteq V_{tp}$ the *subtree rooted in* $w \in V_{tp}$. Naturally, a node $w' \in V_{tp}$ is included in $\mathbf{T}(w)$ if and only if w lies in the direct path between w' and the root node q which in particular implies $w \in \mathbf{T}(w)$. Consequently, this definition is dependent on the choice of the root node. An example is illustrated in Figure 4.1.

Now let us present the components of the XTI basis representation:

- The topology of the basis tree $\mathcal{T}_\mathcal{B}$ is stored by means of a *predecessor function* $\mathbf{p} : V_{tp} \to V_{tp} \cup \{0\}$ where $\mathbf{p}(w) = w'$ indicates that w' is the second node (after w itself) on the direct path from w to the root node q. Accordingly, w' is also called the *parent node* of w. Since, by the above definition, the root node q has no predecessor, one sets $\mathbf{p}(q) := 0$.
- The *primal solution* x is represented by a function $\mathbf{x} : V_{tp} \to \mathbb{R}_0^+$ where $\mathbf{x}(w)$ denotes the flow from w to $\mathbf{p}(w)$. In case of the root node, one sets $\mathbf{x}(q) = 0$.
- The *dual solution* (u, v) is given by a function $\mathbf{y} : V_{tp} \to \mathbb{R}$ where $\mathbf{y}(w)$ denotes the value of the dual variable corresponding to the node w, i.e.

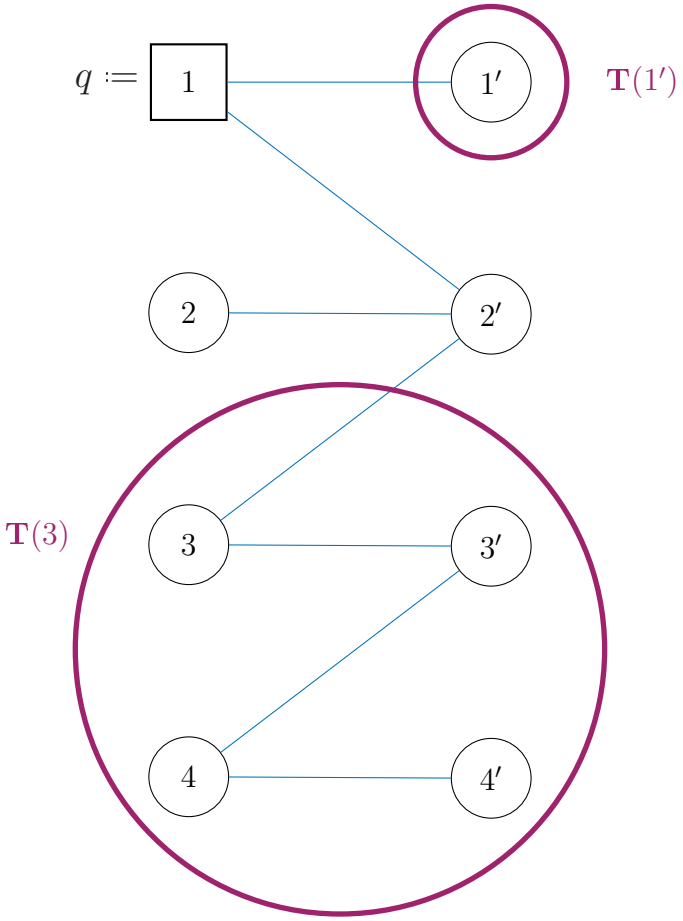


Figure 4.1: Consider a spanning tree for a transportation graph G_{tp} with 4 origins and destinations each, i.e. $m = n = 4$. For simplicity, assume that the node 1 was defined as the root node. Then the subtree $\mathbf{T}(3)$, rooted in 3, consists of the nodes $3, 3', 4$ and $4'$. The subtree $\mathbf{T}(1')$ only contains the node $1'$ itself.

u_w if $w \in S$ and v_w if $w \in T$ respectively.

- The *thread function* $\mathbf{s} : V_{tp} \to V_{tp}$ defines a depth-first traversal[4]

$$\Big(q,\ \mathbf{s}(q),\ \mathbf{s}^2(q),\ \ldots,\ \mathbf{s}^{m+n-1}(q)\Big)$$

  starting in the root node q and cycling back to the root after all nodes of the graph have been visited by setting $\mathbf{s}^{m+n} := q$. This function represents a key component of the basis representation as it allows to rapidly determine other XTI components, see e.g. Section 4.3.3. More intuitively, it defines a sequence where – given that the spanning tree is drawn accordingly – each node of the transportation graph is traversed exactly once in a left to right, top to bottom order starting from the root node and, after each node has been visited, returns to the root node.

- With regard to the tread function, we follow the recommendation of Glover et al. (1979) and Kovács (2015) and additionally store the *reverse thread* function $\mathbf{r} : V_{tp} \to V$ to further accelerate the updating process of the other XTI functions. This represents the inverse of $\mathbf{s}$ and accordingly is characterized by $\mathbf{s}(\mathbf{r}(w)) = w$. Its explicit storage eliminates additional calculations in the update process, see. Glover et al. (1979).

- The function $\mathbf{t} : V_{tp} \to \{1, \ldots, m+n\}$ denotes for each $w \in V_{tp}$ the number of nodes in the subtree $\mathbf{T}(v)$ of $\mathcal{T}_\mathcal{B}$. As explained later, it constitutes the main innovation of the XTI method.

- Furthermore, the function $\mathbf{f} : V_{tp} \to V_{tp}$ maps each $w \in V$ on the last node in $\mathbf{T}(w)$ with respect to the thread function $\mathbf{s}$, that is, $\mathbf{f}(w)$ represents the node in $\mathbf{T}(w)$ with the rightmost position in the tuple

$$\Big(q,\ \mathbf{s}(q),\ \mathbf{s}^2(q),\ \ldots,\ \mathbf{s}^{m+n-1}(q)\Big).$$

- In addition to the functions above, the root node q of $\mathcal{T}_\mathcal{B}$ is stored since the basis tree may be rerooted during the update operations

Note that the common domain of all XTI functions, except the root node, is the set of nodes $V_{tp} := S \dot\cup T$. Thereby, as indicated by Glover et al. (1979)

[4] We set $\mathbf{s}^2(w) := \mathbf{s}(\mathbf{s}(w))$, $\mathbf{s}^3(w) := \mathbf{s}(\mathbf{s}(\mathbf{s}(w)))$, etc.

and Kovács (2015), these functions can be efficiently implemented by means of 7 arrays of length $(m + n)$, where the first m elements store the values of the nodes in S and the last n elements the values of the nodes in T, respectively. This array representation is illustrated for example in Table 4.1. Additionally, an illustration of the XTI functions containing topology information, i.e. **p**, **s**, **r**, **t** and **f**, is given in Figure 4.2 and a full realization of the XTI functions for the basic solution presented in Figure 4.4 is given in Table 4.1.

XTI update

Let us elaborate on the realization of the XTI update operations. The purpose of the XTI method is to enable a quick transition from one basic solution to the next. A key factor for its good performance is the observation, already made for the ATI method by Glover et al. (1972), that the majority of the update operations performed in a pivot step can be restricted to the smaller of two disjoint subtrees of the basis tree. To quickly determine this subtree, Srinivasan and Thompson (1972) proposed to expand the ATI data structure by the function **t**, which lists the number of nodes in a subtree. However, the performance gain achieved by the fast identification of the smaller subtree were largely canceled out by the updating concept which was provided for the new function **t** itself.

The essential innovation of the XTI method of Glover et al. (1979) is a relabelling scheme that allows **t** to be updated efficiently. In order to guarantee a seamless integration of the update of the other ATI functions, that is **y**, **p**, **s** and **r**, the additional index **f** is introduced. Moreover, Glover et al. (1979) showed that **f** allows to further streamline the update of the ATI information. Their computational results indicated that these improvements make the XTI method approximately twice as fast as the ATI method. However, while its efficiency in practice is confirmed by Glover and Klingman (1982) as well as Kovács (2015), none of the mentioned publications provide theoretical complexities of the method.

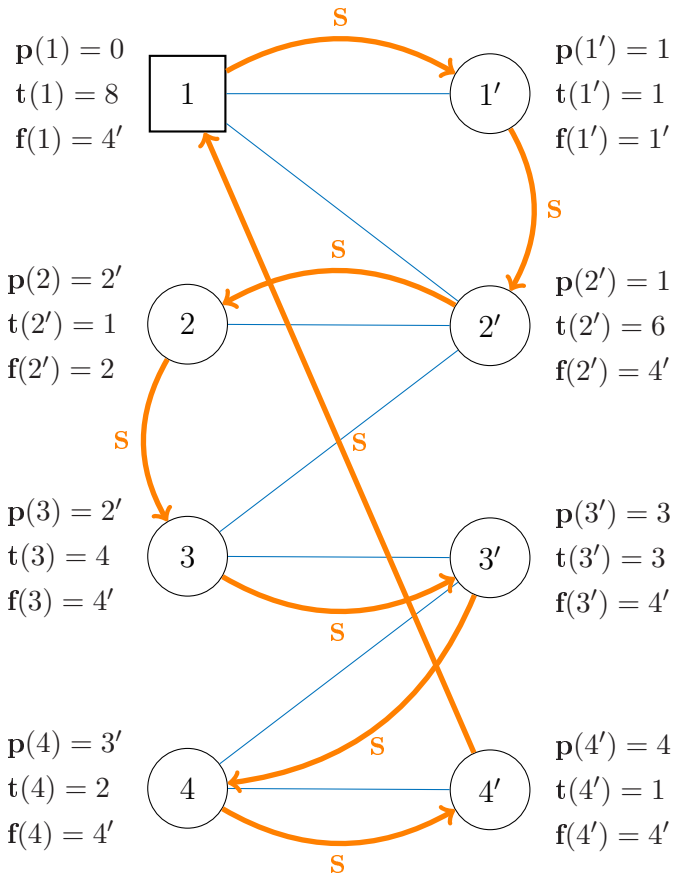


Figure 4.2: Consider the spanning tree introduced in Figure 4.1. A possible depth-first traversal $\mathbf{s}$, starting in the root node 1, is indicated by the orange arrows. Accordingly a backward traversal of these arrows defines the reverse thread $\mathbf{r}$. In addition, for each node $w \in \{1, 2, 3, 4, 1', 2', 3', 4'\}$ the predecessor $\mathbf{p}(w)$, the number of nodes $\mathbf{t}(w)$ in its subtree $\mathbf{T}(w)$ and the last node $\mathbf{f}$ in its subtree $\mathbf{T}(w)$ with respect to $\mathbf{s}$ are given.

Procedure In the following, we describe the essential operations of the XTI update procedure. The functions representing topological information, that is $\mathbf{p}$, $\mathbf{s}$, $\mathbf{r}$, $\mathbf{t}$, $\mathbf{f}$ and the root node $\mathbf{q}$, are updated in the following consecutive steps:

(1) Computation of the entering and leaving edges (variables).

(2) Identification of two disjoint subtrees by eliminating the leaving edge from the basis tree.

(3) Rerooting the smaller subtree in the node incident to the entering edge.

(4) Joining the two subtrees by the entering edge.

Each step is accompanied by the corresponding adaptation of the XTI representation. Observe that Step (4) changes the root of the basis tree, when the former root was contained in the smaller subtree. An illustration of the transformation of the basis tree during the update is given in Figure 4.3. The adaption of the primal variables is incorporated in steps (1) and (3). Concerning the update of the dual multipliers, we implement *Procedure 4.* as presented in Glover et al. (1979), that is, we update the dual variables of the smaller subtree independently after steps (1) to (4) have been carried out. The exact execution of the update of the primal and dual solution are given in the description of Phase II of the TPS in Section 4.3.3.

In order to take full advantage of the network interpretation of the pivot step, the XTI update is highly optimized. This includes in particular the exploitation of cancellation effects between steps (1) to (4) and, whenever possible, the merging of operations. This makes the description extensive and quite technical, especially with regard to the topological information which includes possible rerooting of subtrees. Since this would be beyond the scope of this chapter, we will hereinafter focus on the update of the primal and dual solution and refer to Glover et al. (1979) for the technical details concerning the update of the topology of the basis tree, i.e. $\mathbf{p}$, $\mathbf{s}$, $\mathbf{t}$, $\mathbf{f}$ and $\mathbf{r}$.

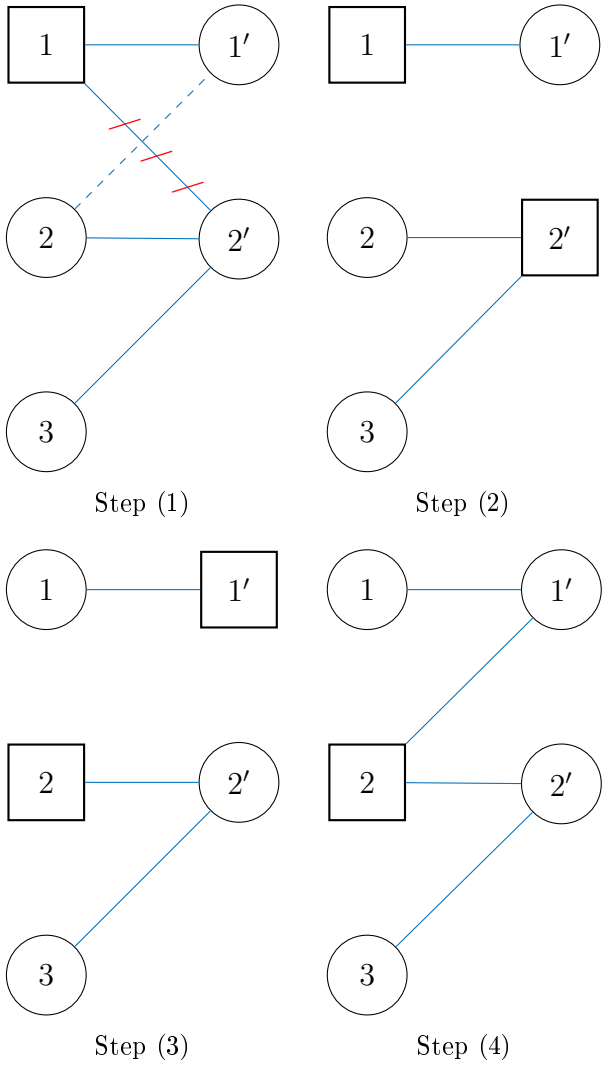


Figure 4.3: The update steps (1) to (4) of the XTI method are displayed for a basis tree $\mathcal{T}$ of the transportation graph with dimensions $m = 3$ and $n = 2$. In Step (1), the dashed entering edge $\overline{21}$ and the leaving edge $\overline{12}$, crossed by short red lines, are indicated. In Step (2) the resulting two disjoint subtrees $\mathcal{T} \setminus \mathbf{T}(2')$ and $\mathbf{T}(2')$ are depicted. The smaller tree $\mathcal{T} \setminus \mathbf{T}(2')$ is then rerooted in node 2 incident to the entering edge in Step (3). Finally the two trees are joined by the entering edge in Step (4) to form the new basis tree which accordingly has the root 2.

4.3.2 Phase I

The aim of Phase I is to find an initial primal feasible basic solution x and the corresponding dual multipliers (u, v).

Initial computation of the primal solution

In case of the TPS, the primal solution can be determined very elegantly by applying one of the primal basic heuristics presented in Chapter 3. This offers a wide range of basic solutions, which can be evaluated by two decisive criteria: the inherent cpu time of the heuristic and the number of pivots the Simplex subsequently requires to improve this solution to optimality. Additionally, former research indicated dependencies in the performance of heuristics and pivot rules, cf. Srinivasan and Thompson (1973) or Glover et al. (1974b). Accordingly, we initially combine all the heuristics presented in Chapter 3 with the proposed pivot rules in Section 4.3.4 in our numerical investigations. An exemplary initial basic solution is given in Figure 4.4.

Initial computation of the dual multipliers and the XTI representation

In addition, Phase I of the TPS also includes the calculation of the corresponding dual multipliers, which is carried out with the initialization of the other components of the XTI method due to our choice regarding the basis representation. The XTI initialization induces additional computational effort, especially with respect to the topology information $\mathbf{p}$, $\mathbf{s}$, $\mathbf{r}$, $\mathbf{t}$ and $\mathbf{f}$. However, this disadvantage is negligible in comparison with the faster execution of the pivot steps.

Naturally, the XTI representation is computed in a separate step after the primal solution is determined by a basic heuristic. Most importantly, this entails the computation of a depth-first traversal of the basis tree $\mathcal{T}_\mathcal{B}$ in order to determine the topology information $\mathbf{p}$, $\mathbf{s}$ $\mathbf{r}$, $\mathbf{t}$ and $\mathbf{f}$. The necessary calculations are described in detail in Glover et al. (1979); with regard to Section 4.3.5, observe that their method always yields a strongly feasible spanning tree, cf.

Ahuja et al. (1993).

When the above operations were executed, the dual multipliers (u, v) are determined. Therefore, the value of the dual variable corresponding to the root node q is customarily set to zero. By means of the depth-first sequence defined by $\mathbf{s}$, the values of the remaining variables can then be determined by traversing the basis tree in a cascading fashion and setting

$$u_i := c_{i\mathbf{p}(i)} + v_{\mathbf{p}(i)}$$

for each i in S and

$$v_j := u_{\mathbf{p}(j)} - c_{\mathbf{p}(j)j}$$

for all j in T, respectively. This ensures that the complementary slackness conditions (1.4) for x and (u, v) will be satisfied and processing the nodes in the order of a depth-first traversal guarantees that the dual variable of the predecessor of a node has already been set when the node is visited. An illustration of this process is given in Figure 4.5.

Integrated XTI heuristics

For certain heuristics, the computation of the XTI information can be further streamlined. In the execution of the NWCR, the edges of the basis tree are considered in the order of a depth-first search. Thereby, the computation of the XTI information as well as the dual solution can be integrated into the execution of the heuristic. However, since the NWCR generally provides very inefficient starting solutions, the TMR was proposed in Section 3.1. This constitutes an attempt to provide a heuristic that defines an efficient initial solution by means of a depth-first traversal. we highlight the two integrated heuristics by using the notation NWCRXTI and TMRXTI for the corresponding implementations.

4.3.3 Phase II

Provided the same initial solution and pivot rule are used, the TPS performs the same pivot steps as the LPS in the course of Phase II. The main difference

lies in the realization of pivot operations 2 and 3 by means of the XTI representation. In addition, the specific structure of the problem can be further exploited by choosing a suitable pivot rule.

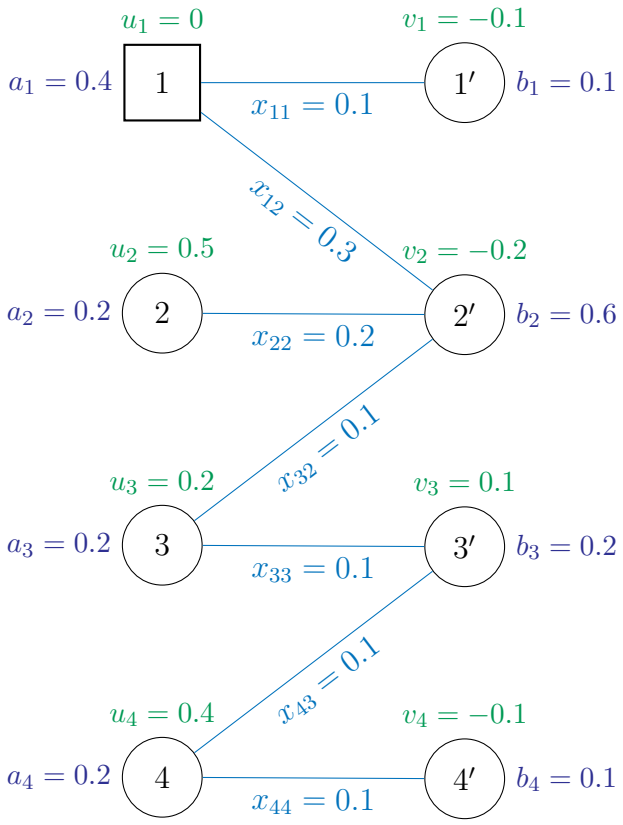
Assuming that a current basic solution x as well as the corresponding dual multipliers (u, v) and an XTI representation of the basis tree $\mathcal{T}_\mathcal{B}$ are given, we now present the TPS equivalents of the three operations executed in the pivot step of the LPS (cf. Algorithm 2). Thereby, we view the basis elements (i, j) as edges $\overline{ij}$ in the transportation graph G_{tp} and adapt the nomenclature accordingly. Hence, the three pivot operations constitute:

1. Determining the entering edge.
2. Determining the leaving edge.
3. Updating the basis.

we support our explanations by an exemplary pivot step of the TPS in Figures 4.4 to 4.8 and show the corresponding XTI representations in Tables 4.1 and 4.2.

Nodes	**p**	**x**	**y**	**s**	**r**	**t**	**f**
1	0	0	0	1'	4'	8	4'
2	2'	0.2	0.5	3	2'	1	2
3	2'	0.1	0.2	3'	2	4	4'
4	3'	0.1	0.4	4'	3'	2	4'
1'	1	0.1	-0.1	2'	1	1	1'
2'	1	0.3	-0.2	2	1'	6	4'
3'	3	0.1	0.1	4	3	3	4'
4'	4	0.1	-0.1	1	4	1	4'

Table 4.1: An XTI representation for the basic solution presented in Figure 4.4; the root node 1 is indicated by a square.

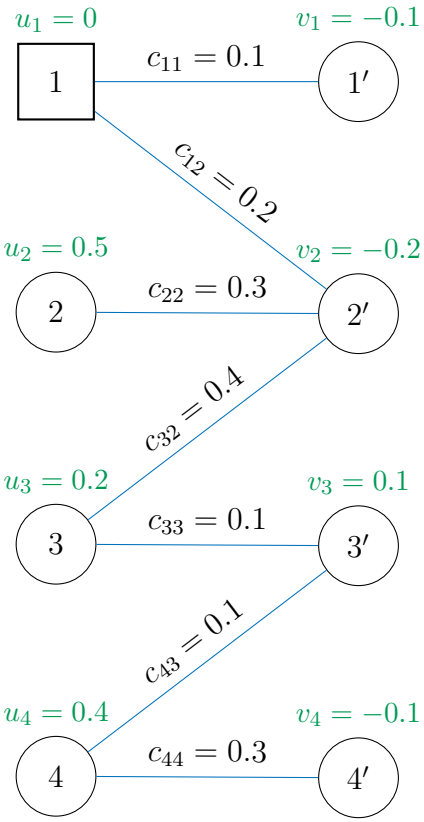


$$C = \begin{pmatrix} 0.1 & 0.2 & 0.6 & 0.3 \\ 0.8 & 0.7 & 0.1 & 0.7 \\ 0.6 & 0.4 & 0.1 & 0.4 \\ 0.8 & 0.9 & 0.3 & 0.5 \end{pmatrix}$$

$$X = \begin{pmatrix} \mathbf{0.1} & \mathbf{0.3} & 0 & 0 \\ 0 & \mathbf{0.2} & 0 & 0 \\ 0 & \mathbf{0.1} & \mathbf{0.1} & 0 \\ 0 & 0 & \mathbf{0.1} & \mathbf{0.1} \end{pmatrix}$$

$$B = \left\{ \begin{array}{l} (1,1'), (1,2'), (2,2'), \\ (3,2'), (3,3'), (4,3'), \\ (4,4') \end{array} \right\}$$

Figure 4.4: We continue the example considered in Figures 4.1 and 4.2, that is, we choose a basis B such that the previously introduced spanning tree represents the basis tree $\mathcal{T}_\mathcal{B}$ and add costs C as well as supplies a and demands b for a transportation problem of dimension $m = n = 4$. For these input parameters, a potential initial primal basic solution x is depicted. Furthermore, the corresponding dual multipliers (u, v) are given.



$$C = \begin{pmatrix} 0.1 & 0.2 & 0.6 & 0.3 \\ 0.8 & 0.7 & 0.1 & 0.7 \\ 0.6 & 0.4 & 0.1 & 0.4 \\ 0.8 & 0.9 & 0.3 & 0.5 \end{pmatrix}$$

$$C^{uv} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & 0.7 & 0.2 \\ 0.2 & \mathbf{0} & -0.3 & 0.1 \\ 0.3 & \mathbf{0} & \mathbf{0} & 0.1 \\ 0.3 & 0.3 & \mathbf{0} & \mathbf{0} \end{pmatrix}$$

$$X = \begin{pmatrix} \mathbf{0.1} & \mathbf{0.3} & 0 & 0 \\ 0 & \mathbf{0.2} & 0 & 0 \\ 0 & \mathbf{0.1} & \mathbf{0.1} & 0 \\ 0 & 0 & \mathbf{0.1} & \mathbf{0.1} \end{pmatrix}$$

Figure 4.5: Observe that the dual variables in Figure 4.4 have been assigned values that satisfy complementary slackness, i.e. , the reduced costs on the basic edges satisfy $c_{ij}^{uv} := c_{ij} - u_i + v_j = 0$. If u_1 is initially set to zero, this leads to uniquely defined values for the remaining dual variables. Furthermore, these are easily computable by traversing the tree via $\mathbf{s}$, i.e. $1 \xrightarrow{\mathbf{s}} 1' \xrightarrow{\mathbf{s}} 2' \xrightarrow{\mathbf{s}} 2 \xrightarrow{\mathbf{s}} 3 \xrightarrow{\mathbf{s}} 3' \xrightarrow{\mathbf{s}} 4 \xrightarrow{\mathbf{s}} 4'$. This ensures that each node is only visited once and whenever a node is considered, the dual variable of its predecessor is already computed. The resulting reduced costs for the non-basic edges are shown on the right side. Moreover, note that the matrices X and C^{uv} indeed satisfy complementary slackness. In this situation, the optimal solution is not yet achieved and the edge $\overline{23}$ is the only candidate for an entering edge, since all other reduced costs are non-negative.

Pivot operation 1: Determining the entering edge

As in the LPS, the selection of an entering edge or the alternative determination of optimality is carried out by employing a pivot rule, that is the computation of the reduced costs for a subset[5] of all edges. Since there are many suitable rules available, we will analyze this step individually in Section 4.3.4 where we consider several realizations that take into account the specific structure of the problem.

Pivot operation 2: Determining the leaving edge

This operation can be efficiently executed by traversing a cycle in the enlarged basis tree $\mathcal{T}_\mathcal{B}$ and comparing the flow on the visited edges. More precisely, this cycle, called *the basis equivalent path*, results from adding the entering edge $\overline{i_e j_e}$, obtained in pivot operation 1, to $\mathcal{T}_\mathcal{B}$ as shown in Figure 4.6. It can be efficiently identified by retracing the respective direct paths from i_e and j_e to the root by means of the predecessor function $\mathbf{p}$ until the initial intersection node is found. Furthermore, the flow value of the new edge $\overline{i_e j_e}$ can be computed simultaneously. Thereby, one considers all backward edges in the cycle, i.e. all edges $\overline{ij}$ for which the first visited node is j, if one assumes an orientation of the edge $\overline{i_e j_e}$ from its origin to its destination and traverses the cycle in the according direction. In order to maintain primal feasibility, an edge $\overline{i_l j_l}$ with minimum flow $x_{i_l j_l}$ is selected from these edges via $\mathbf{p}$ and $\mathbf{x}$. Whenever an edge $\overline{v\mathbf{p}(v)}$ (w.l.o.g. v is in S and $\mathbf{p}(v)$ is in T) is crossed via $\mathbf{p}$, its flow value is easily accessible since $\mathbf{x}(v)$ stores exactly $x_{v\mathbf{p}(v)}$. The update operations in pivot operation 3 will cause this edge to leave the basis tree.

Pivot operation 3: Updating the basis

Finally, the update of the basis is achieved by sending flow through the basis equivalent path (cycle) established in pivot operation 2 and modifying the remaining XTI functions accordingly.

[5] In the optimal case the subset will be equal to the complete set.

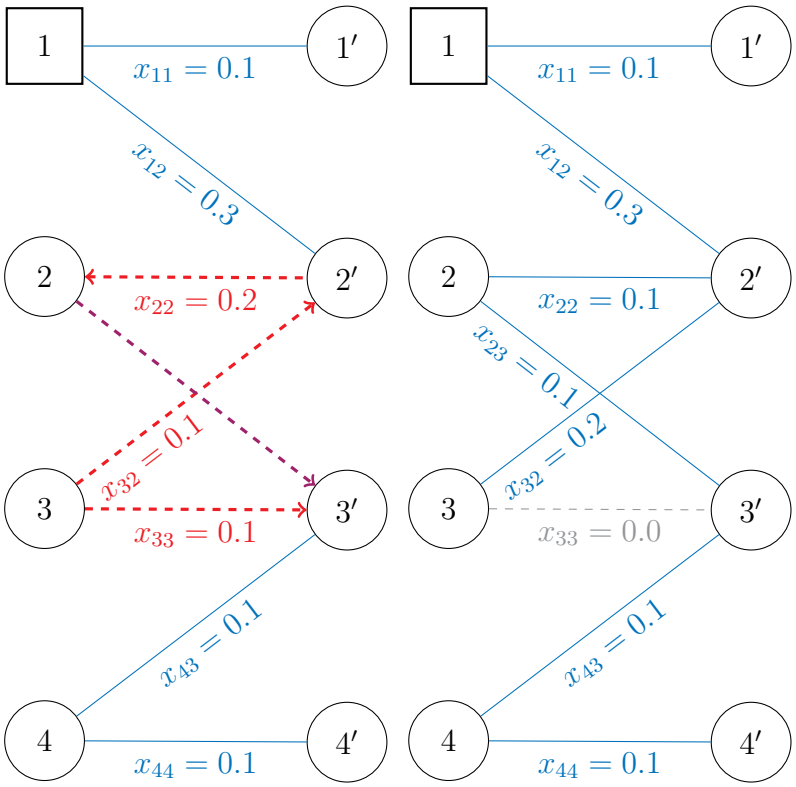


Figure 4.6: With the addition of the edge $\overline{23}$ to the basis tree $\mathcal{T}_\mathcal{B}$, a cycle, that is the basis equivalent path, can be identified. This is done via the predecessor function $\mathbf{p}$ by retracing the paths from 2 and $3'$ to the root until the first intersection is found, i.e. $2 \xrightarrow{\mathbf{p}} 2'$ and $3' \xrightarrow{\mathbf{p}} 3 \xrightarrow{\mathbf{p}} 2'$, since in this case the first intersection node is $2'$. While retracing the paths, the minimum flow on a backward edge is determined. In the figure, the backward edges are $\overline{22}$ and $\overline{33}$ and the minimum flow is hence $x_{33} = 0.1$. Pushing 0.1 units of flow through the cycle will cause the edge $\overline{33}$ to leave the basis while $\overline{23}$ enters it. Thereby, we have obtained a new primal feasible basic solution.

Thus, the primal basic solution is adjusted by sending $x_{i_l j_l}$ units of flow in the appropriate direction through the basis equivalent path (see Figure 3) caused by the entering edge. This is achieved by the functions **p** and **x** and will cause $\overline{i_l j_l}$ to leave and $\overline{i_e j_e}$ to enter the basis. Furthermore, a modification of **x** implies that the dual variables in **y** also need to be adjusted to restore the complementary slackness conditions (1.4) for the primal-dual pair x and (u, v) which is covered in detail the subsequent paragraph.

In line with the primal-dual solution, the functions representing topological information, that is **p**, **s**, **r**, **t** and **f**, are updated to represent the new basis tree. The essential steps of this update were given in Section 4.3.1. As already mentioned, a detailed description would be beyond the scope of this thesis, which is why we refer to the original publication of Glover et al. (1979).

The result of the exemplary pivot step, that is the updated basic solution, is illustrated in Figure 4.8 and Table 4.2.

Update of dual variables Finally, let us elaborate on the update of the dual variables. As mentioned before, a key observation in the update of the XTI representation is that most of the update operations for all functions can be restricted to the smaller of two subtrees of the basis which are determined by the function **t**. In particular, only the dual variables of this subtree need to be updated while the remaining dual variables remain unchanged.

In every pivot step variables (u, v) are chosen such that the complementary slackness condition is satisfied. This is achieved by ensuring that the reduced costs on basic edges are set to zero, i.e.

$$\forall\ \overline{ij} \in \mathcal{T}_\mathcal{B} : c_{ij}^{uv} \coloneqq c_{ij} - u_i + v_j = 0.$$

Dropping the leaving and the entering edge from the basis tree identifies two disjoint subtrees, see Figure 4.7. In the following we denote the larger subtree, i.e. the subtree that contains more nodes, by $\mathcal{T}_1$ and its counterpart by $\mathcal{T}_2$ where ties are split arbitrarily.

Let us assume that all dual variables of $\mathcal{T}_1$ remain unchanged in the update process and **p** represents the not yet updated predecessor function of the basis.

Thus, the reduced costs on the edges of $\mathcal{T}_1$ are zero by virtue of the previous pivot step. However, in this case, adding the entering edge $\overline{i_e j_e}$ demands a modification of the dual variables in $\mathcal{T}_2$.

In order to analyze this modification, we first assume that the first node i_e of the entering edge is in $\mathcal{T}_1$ and the second node j_e is in $\mathcal{T}_2$. Consequently, keeping u_{i_e} unchanged, we have to set v_{j_e} to $c_{i_e j_e} - u_{i_e}$ to achieve $c^{uv}_{i_e g g_e} = 0$. Since the former value of v_{j_e} was set to $c_{\mathbf{p}(j_e) j_e} - u_{\mathbf{p}(j_e)}$ this implies that v_{j_e} has to be increased by

$$\delta := u_{i_e} - c_{i_e g g_e} - u_{\mathbf{p}(j_e)} + c_{\mathbf{p}(j_e) j_e}.$$

In the opposite case, where j_e is contained in $\mathcal{T}_1$, this would require that $u_{\mathbf{p}(j_e)}$ must be increased by

$$\delta' := c_{i_e g g_e} + v_{j_e} - c_{i_e \mathbf{p}(i_e)} - v_{\mathbf{p}(i_e)}.$$

In both cases, the remaining variables of $\mathcal{T}_2$ can be updated by traversing $\mathcal{T}_2$ via the **updated** thread function $\mathbf{s}$ and increasing each variable in turn by δ (or δ'). This constitutes an advantage of the reduced cost formulation

$$c^{uv}_{ij} := c_{ij} - u_i + v_j$$

where the increase of one dual variable of an edge is canceled out by increasing the other edge. In case of the representation

$$c^{uv}_{ij} := c_{ij} - u_i - v_j,$$

commonly used for (TP), each node would be either increased or decreased by δ depending on its predecessor, which would entail an additionally inquiry for each node of $\mathcal{T}_2$. An example for the update process of the dual variables can be found in Figure 4.7.

4.3.4 Pivot rules

The choice of the pivot rule is a very important factor in the design of a Simplex Method. Its task is to find a non-basic element (edge) with negative reduced

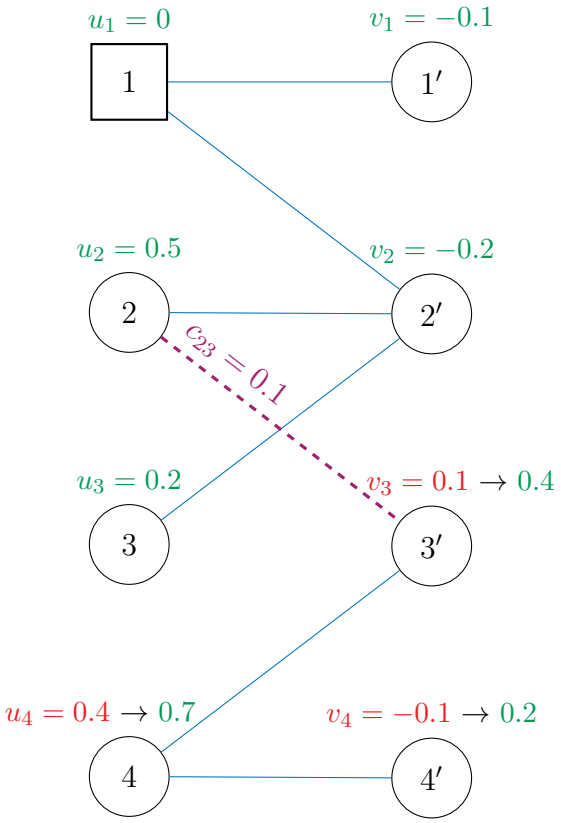
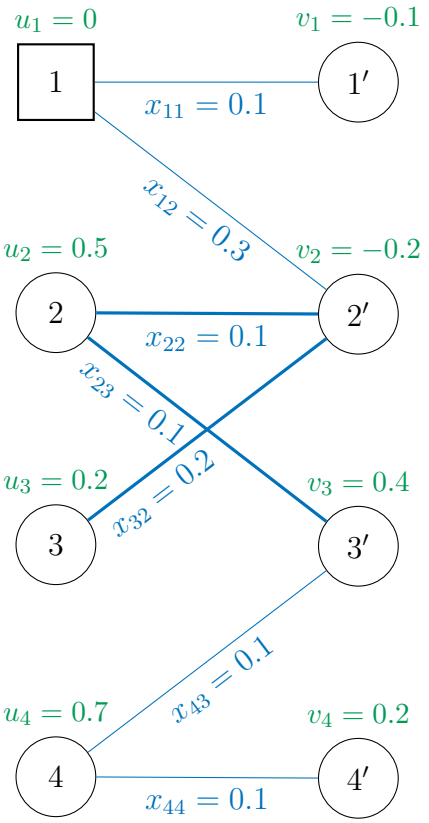


Figure 4.7: Eliminating the leaving edge $\overline{33}$ from the basis tree results in two disjoint subtrees that will be joined by the entering edge $\overline{23}$ to form the new basis tree. In this example, the tree $\mathbf{T}(3')$ (with respect to the predecessor function $\mathbf{p}$) is the smaller subtree. Since the dual variables in $V \setminus \mathbf{T}(3')$ remain unchanged, joining 2 and 3 implies an increase in v_3 to set the reduced costs to zero, i.e. $c_{23}^{uv} - u_2 + v_3 = 0$. By the definition of the reduced costs, it suffices to increase the remaining dual variables of the subtree by the same amount, i.e. 0.3, since the reduced costs of the corresponding edges have been zero in previous pivot. Afterwards, the complementary slackness conditions are satisfied for the complete new basis tree.



$$C = \begin{pmatrix} 0.1 & 0.2 & 0.6 & 0.3 \\ 0.8 & 0.7 & 0.1 & 0.7 \\ 0.6 & 0.4 & 0.1 & 0.4 \\ 0.8 & 0.9 & 0.3 & 0.5 \end{pmatrix}$$

$$C^{uv} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & 1.0 & 0.5 \\ 0.2 & \mathbf{0} & \mathbf{0} & 0.4 \\ 0.3 & \mathbf{0} & 0.3 & 0.4 \\ 0 & 0 & \mathbf{0} & \mathbf{0} \end{pmatrix}$$

$$X = \begin{pmatrix} \mathbf{0.1} & \mathbf{0.3} & 0 & 0 \\ 0 & \mathbf{0.1} & \mathbf{0.1} & 0 \\ 0 & \mathbf{0.2} & 0 & 0 \\ 0 & 0 & \mathbf{0.1} & \mathbf{0.1} \end{pmatrix}$$

Figure 4.8: Finally, we depict the updated basic solution. In this case, the root node is still 1 since it was not part of the updated subtree. Moreover, the updated reduced costs for the non-basic edges are all non-negative which implies that an optimal solution has been attained. Furthermore, we observe dual degeneracy with respect to the edges $\overline{41}$ and $\overline{42}$, i.e. $c^{uv}_{41} = c^{uv}_{41} = 0$. The corresponding XTI representation for this solution is given in Table 4.2. Note that the root node of $\mathcal{T}_\mathcal{B}$ is still given by 1 since it was contained in the larger subtree during the update, cf. Section 4.3.1.

Nodes	**p**	**x**	**y**	**s**	**r**	**t**	**f**
1	0	0	0	1'	3	4'	3
2	2'	0.1	0.5	3'	2'	4	4'
3	2'	0.2	0.2	1	4'	1	3
4	3'	0.1	0.7	4'	3'	2	4'
1	1	0.1	-0.1	2'	1	1	1'
2	1	0.3	-0.2	2	1'	2'	3
3	2	0.1	0.4	4	2	3	4'
4	4	0.1	0.2	3	4	1	4'

Table 4.2: The updated XTI representation for the new basic solution given in Figure 4.8; the root node 1 is indicated by a square.

costs. This constitutes a classical trade-off situation, as the cpu time per pivot step increases with the number of inspected elements but lower reduced costs of the entering element in a single pivot generally imply that fewer pivot steps are required overall, since the objective function is decreased by a larger margin in each iteration. Thus, two decisive variables for the quality of a pivot rule can be identified: the total number of pivot steps performed in Phase II and the number of elements for which the reduced costs are computed in a single pivot step. Observe that in the optimal case, every pivot rule has to consider all elements (the complete set E_{tp}) in order to verify that all reduced costs are non-negative.

Apart from possible deviations with regard to the data structures used to store variables and parameters, the pivot rules can be implemented in the same way for the LPS, NS and TPS. The only concession to the transportation problem is to choose specific pivot rules whose selection better exploits the problem structure. In the remainder of this section, we present a selection of such rules.

The evaluation of different pivot rules for the transportation problem has been the subject of extensive numerical investigations (see e.g. Srinivasan and Thompson (1973) and Glover et al. (1974b)). Since our focus of research is to gain insights in the potential of column generation in the context of piv-

oting in the TPS, we refrain from the reimplementation of all hitherto tested rules. Instead we focus on a selection of rules listed below which seemed most promising based on the results of Srinivasan and Thompson (1973), Glover et al. (1974b), Gottschlich and Schuhmacher (2014) and Kovács (2015). We will later introduce additional pivot rules in the context of column generation for the transportation problem in Chapter 5.

Hereafter we assume that a primal solution x and the corresponding dual multipliers (u, v) satisfying the complementary slackness conditions (1.4) are given.

Modified Row Most Negative (RMN) With regard to the results of Srinivasan and Thompson (1973), Glover et al. (1974b) and Gottschlich and Schuhmacher (2014), this seems to be the most promising classical rule for the TPS if there is no specific structure in the problem instances.

This rule cyclically evaluates reduced costs $c_{ij}^{uv} = c_{ij} - u_i + v_j$ with respect to the cost matrix C and is commonly implemented as follows: In each pivot step, the process starts with the first row not considered in the previous one. For a row i, one computes the reduced costs c_{ij}^{uv} corresponding to all non-basis elements $(i, j) \notin B$. Among these candidates, the most negative is chosen, or in the event that all reduced costs are non-negative, one proceeds by inspecting the subsequent row. The next row of a given row i is thereby the row $i + 1$ or 1 in case that $i = m$.

Based on our numerical analysis, we make a slight **modification** of the rule compared to the publications mentioned above: Instead of starting the inspection with the row $i + 1$, following the row i which was last considered in the previous step, we begin by investigating the row i again. This results in a reduction of the total cpu time of the TPS by approximately 10%, which is possibly explained as follows: A disadvantage of RMN is that at most one element of a considered row is added to the basis and a row is only inspected again after the rule has traversed the complete matrix again. Therefore, especially near the optimum, it makes sense to check if more eligible candidates remain in a row after the addition of one element before continuing with the next row.

Modified Column Most Negative (CMN) This rule constitutes the column-equivalent of the RMN rule, that is one executes the RMN rule on $C^\top$.

Row Block Search (RBLK) This rule proposed by Grigoriadis (1986) achieved the best results in the computational investigations of Kovács (2015) for the NS on general minimum cost flow problems. Starting with the first edge not considered in the previous iteration, this method also inspects sequences of edges $\overline{ij}$ and selects the one with the most negative reduced costs c_{ij}^{uv}. In contrast to the RMN and CMN, however, a sequence is not limited to a row or column, but results from the cyclic row-wise traversing of p_{blk} elements of the cost matrix C. In the case that all reduced costs of a sequence are non-negative, the rule continues traversing C until either a negative element was found, at which point it directly stops its investigations, or all elements have been inspected, i.e. optimality is verified.

Regarding the parameter p_{BLK}, our investigations supported the recommendation in Kovács (2015) to set $p_{BLK} := \lfloor\sqrt{mn}\rfloor$ in order to achieve the maximum efficiency. This also renders the investigation of two classical rules obsolete, namely the *matrix most negative rule* and the *first negative rule*, see Gottschlich and Schuhmacher (2014), since they represent special cases of the RBLK rule for $p_{BLK} := mn$ and $p_{BLK} := 1$. Moreover, for $p_{BLK} := \lfloor\sqrt{mn}\rfloor$, the rule is equivalent to the RMN in the symmetric case and will therefore exclusively be evaluated for asymmetric problem instances.

Column Block Search (CBLK) Analogously, we implement the column-equivalent of the RBLK rule where the matrix C is traversed column-wise. If in the following, we refer to the RBLK and the CBLK as the *Block Search rules*.

Row Shortlist (RSL) A new and (on certain problem classes) very successful type of pivot rule is introduced in the context of a TPS variant by Gottschlich and Schuhmacher (2014) called the *Shortlist Method*.

Its key innovation is to first solve a reduced problem on a subset of the edges. Therefore, Phase II is divided into two sub-phases; in the initial sub-phase, the algorithm limits the possible entering edges to a subset of candidates who are more likely to be in an optimal solution, thus reducing the calculation effort of the pivot steps. Only when an optimal solution for this reduced problem is found, the pivot rule considers the whole set of edges E_{tp}.

To this end, for each row i in the cost matrix C, a shortlist $L_i \subseteq J$ is created. More precisely, the row i of C is sorted in ascending order and the column indices of the best l_{SL} candidates (the l_{SL} edges with the smallest costs) are stored in the corresponding shortlist L_i. In the first sub-phase, only these shortlists L_i, $i \in I$, are considered in the selection of entering edges in pivot operation 1.

Moreover, as l_{SL} is chosen to be significantly smaller than n (see below), Gottschlich and Schuhmacher (2014) additionally introduce a specific selection process for the entering edge in the first sub-phase, henceforth referred to as *Shortlist selection*: Therefore, in each pivot step, the shortlists are consecutively traversed until either k_{SL} edges with negative reduced costs have been found or p_{SL} percent of all shortlists have been searched. As in the previously described rules, the element with the most negative reduced costs of the inspected sequence is chosen. If no negative element was found the search continues until the first negative element was found or all shortlists were searched. In the next iteration, one starts with shortlist not considered in the previous step. In the case that all checked edges have non-negative reduced costs, the problem is optimally solved with respect to the shortlists but not necessarily with respect to all edges (variables), since the computed dual multipliers might lead to negative reduced costs for edges not contained in the shortlists. Therefore, a second pivot rule is applied in the following sub-phase, which takes all edges into account and thus leads to an optimal solution of the complete (original) problem. Clearly, the choice of the parameter set l_{SL}, k_{SL} and p_{SL} is critical for the performance of the rule. We obtained good results for the values proposed in Gottschlich and Schuhmacher (2014) as "a first step to good

universal parameters", i.e.

$$l_{SL} := \begin{cases} \min\{15, n\} & \text{for } n \leq 200, \\ 15 + \lfloor 15 \cdot \log_2(n/200) \rfloor & \text{for } n > 200, \end{cases}$$

$$k_{SL} := l_{SL},$$

$$p_{SL} := 5\%.$$

Lastly, with regard to the final pivot rule applied in the second sub-phase of Phase II, we follow the recommendation of Gottschlich and Schuhmacher (2014) and apply the RMN rule stated above.

Observe that Gottschlich and Schuhmacher (2014) introduce the RSL in the context of a TPS implementation which additionally includes a specific heuristic to compute starting solutions and is represented by the RMR[6] in our setup. In order to combine it with different starting heuristics, we study the RSL independently of the Shortlist Method.

Column Shortlist (CSL) As in the two rules presented before, we implemented a column-variant of the Shortlist pivot rule, maintaining shortlists L_j for the columns $j \in J$, as well. Here we accordingly use the CMN as the global pivot rule in the second sub-phase of Phase II. If in the following statements refer to both the RSL and the CSL, we refer to the Shortlist Rules.

Implementation in Matlab We conclude with a note on the implementation of the presented methods. As explained in the implementation of the heuristics in Section 3.5, MATLAB uses column-major memory and therefore iterates faster over the columns of a matrix than over its rows. Due to the large number of pivot steps performed in the TPS, the impact of this is even more significant in the pivot rules. Hence, as for the heuristics, the row variants of the pivot rules

[6] The differences in the heuristic of Gottschlich and Schuhmacher (2014) and RMR had no practical implications in our numerical investigations.

are most effectively implemented as their column equivalents on the transposed problem, i.e. on $(C^\top, b, a)$ instead of (C, a, b).

Moreover, while the MATLAB code for RMN and CMN can be efficiently vectorized[7], we observed inferior performance of a vectorized variant of the Block Search Rules. This is explained by the structure of the reduced costs $c_{ij}^{uv} = c_{ij} - u_i + v_j$ which leads to a time-consuming replication of dual vectors to compute the reduced costs for sequences (blocks) which are overlapping several rows (or columns) of C. The time required for replication outweighs the time gained by vectorization. Furthermore, since the cost coefficients are inspected in ascending order and the order is therefore not consistent with the dual variables, vectorization of the Shortlist Rules is not practicable. This constitutes a MATLAB-specific advantage of the RMN and the CMN.

4.3.5 Cycling and degenerate basic solutions

Another important aspect in the implementation of the TPS is the phenomenon of *cycling* caused by the existence of degenerated basic solutions introduced in Section 1.5.6. In the context of the Simplex method, cycling refers to an infinite, repetitive sequence of *degenerate pivots*, that is pivot operations that replace a basis variable with zero flow and thereby change the basis, but leave the corresponding basic solution and the current objective value unchanged. In the worst case, this will prevent the algorithm from terminating after a finite number of iterations. Moreover, the practical implications of degenerate basic solutions go beyond the possibility of not terminating but potentially reduce the efficiency of the method in general (see below).

As mentioned before, the differences in pivoting between the TPS and the LPS are restricted to the execution of the pivot steps. Provided the initial solution and the pivot rules used are the same, they will visit exactly the same basic solutions in the course of Phase II. Consequently, the TPS inherits the problems introduced above by design. For instance, it is stated in Ahuja et al. (1993) that computational studies on the Network Simplex report that up to

[7] https://de.mathworks.com/help/matlab/matlab_prog/vectorization.html

90% of the pivot operations can be degenerate (not improving the current solution) in "commonplace" network problems.

However, this can be efficiently handled by enforcing the basis trees to be *strongly feasible* via the *leaving edge rule* (Ahuja et al. (1993)). Thereby, in the event that multiple edges qualify as a leaving edge, i.e. multiple backward edges in the basis equivalent path achieve the minimum flow value, one selects the edge which would last be encountered when traversing the created cycle in the basis tree in the outlined direction. This approach ensures the finiteness of the algorithm and in addition reduces the average computing time in general, see Ahuja et al. (1993).

4.3.6 Final algorithm

Having presented all components of the method, we summarize the TPS in Algorithm 3. Furthermore, we introduce a simple preprocessing technique that is run before Phase I and II are executed and specify the memory requirements of the algorithm.

Preprocessing

The preprocessing routine of Algorithm 3 operates as follows: In order to avoid unnecessary computations, we eliminate all origins i with supply $a_i < 10^{-12}$ as well as destinations j with demand $b_j < 10^{-12}$ and accordingly adapt the variables x and costs c of a given problem instance. The result is an equivalent problem of lower dimension.

In practice, the according reduction in dimension is negligible for the artificial problem classes. Recall that for the UNIFORM instances, the expected values as well as the variance of the components of a and b is given by $\frac{1}{m}$ and $\frac{m-1}{m^2(m+1)}$ or $\frac{1}{n}$ and $\frac{n-1}{n^2(n+1)}$ respectively, cf. Section 2.3.2, and the interval of dimensions considered in our numerical analysis was $m, n \in [10^3, 16 \cdot 10^3]$. In the case of SOLGEN instances, the generation process of a and b is slightly modified, see Section 2.3.4. However, in practice we observed the same behavior, i.e.

Algorithm 3 The Transportation Simplex (TPS)

Preprocessing

1. Delete zero entries in a and b and adapt x and c accordingly.

Phase I

2. Determine a primal feasible basic solution x by means of a primal basic heuristic of Chapter 3.
3. Compute the XTI information including corresponding dual multipliers (u, v).

Phase II

4. Compute reduced costs c_{ij}^{uv} by means of a pivot rule of Section 4.3.4 resulting in one of the two following alternatives:
   a) Selection of a non-basic entering edge $\overline{i_e j_e}$ with $c_{i_e,j_e}^{uv} < 0$.
   b) Observation of optimality, i.e. $c_{ij}^{uv} \geq 0$ for all $\overline{ij} \in E_{tp}$ and thereby the termination of the algorithm.
5. Determine the leaving edge $\overline{i_l j_l}$ and the new basic solution by sending flow through the basis equivalent path.
6. Update the XTI representation and go to 4.

the role of preprocessing in solving is negligible. In contrast, preprocessing plays a decisive role for the subclasses "Shapes" and "MicroscopyImages" of DOTMARK where the total number of supplies and demands is reduced by up to 30%.

Memory requirement

Finally, we state the memory requirement of the algorithm. For this purpose, recall that the size of the input parameters C, a and b is in $\mathcal{O}(mn)$. Apart from

this, the only information which needs to be stored is the XTI representation, which is in $\mathcal{O}(m+n)$, cf. Section 4.3.1. Hence, the matrix C dominates the total memory requirement which amounts to $\mathcal{O}(mn)$.

4.4 Differences to the Network Simplex (NS)

After the TPS has been introduced in Section 4.3, we discuss the differences between the TPS and the NS for general minimum cost flow problems (MCF_u) and (MCF_c) in this section. Observe that a modern implementation of the NS is very well covered in the literature, see e.g. Ahuja et al. (1993) and Kovács (2015). Therefore, we refrain from a comprehensive introduction of the general NS and will only give a comparative presentation of the algorithm with respect to the TPS. In particular, we point out which elements of the NS are implemented differently.

4.4.1 Phase I

One of the main differences of the TPS and the NS constitutes the determination of the initial primal solution. The heuristics of Chapter 3 are specifically designed for the bipartite structure of the transportation problem (TP). While some of them can be generalized to (MCF_u) and (MCF_c), see e.g. Glover et al. (1974a), it is more common to artificially extend the underlying network such that an initial basic solution is known. Thereby, an artificial root node with arcs of infinitely high costs and sufficient capacities to all other nodes is introduced, cf. Bradley et al. (1977), Ahrens and Finke (1980) and Kovács (2015). The NS then continues with Phase II and solves the extended problem. If, in the bounded case, all artificial arcs are removed in this process, an optimal solution for the original problem is obtained; otherwise the original problem is infeasible. Furthermore, observe that the artificial arcs will not be included in a negative cost cycle in the unbounded case, see Section 4.4.2.

In case of the TPS, the approach above is probably best represented by the

NWCR since the computational effort (each node is examined exactly once) is identical and in both cases a solution is determined whose distance from the optimal solution (in pivot steps) is unknown. A minor disadvantage of the artificial edges, however, is that at least these must be removed from the basis before the optimum (for feasible problems) can be achieved.

Additionally, Ahuja et al. (1993) mention the possibility to obtain an initial solution by solving a maximum flow problem. However, this approach seems to be less common in practice, cf. Kovács (2015).

Compute XTI information and the corresponding dual solution

After the primal solution is determined, the XTI information as well as the corresponding dual multipliers are obtained as in the TPS, i.e. by identifying a depth-first traversal of the basis tree and subsequently determining the remaining parameters of the XTI representation.

4.4.2 Phase II

The differences in Phase II are more subtle. Thereby, we first consider uncapacitated minimum cost flow problems, i.e. (MCF_u), for comparison.

The uncapacitated case

Assume a uncapacitated problem (MCF_u) is given for the graph $G = (V, \vec{E})$, cf. Section 1.4.3. we describe how Phase II of the TPS generalizes to the NS in this case in the following: Firstly, recall that we have adopted the representation (1.8) of reduced costs for (TP) in the context of the TPS which ensures that dual variables are computed as in general minimum cost flow problems. Thus, we implicitly solve the problem formulation (1.6) of (TP) and, as described in Section 1.4, the only difference between (1.6) and (MCF_u) lies in the underlying graph for which a minimum cost flow is established. Furthermore, as indicated by the proof of Theorem 1.4.2, the one-to-one correspondence between bases

of the incidence matrix and spanning trees generalizes to general minimum cost flow problems, see e.g. Ahuja et al. (1993). Observe further that in case of the XTI representation and its update, no restrictions on the underlying graph were made and, in particular, it was not required that the graph is bipartite. Thereby the representation of the basis in (MCF_u) is exactly the same as for (TP), i.e. a spanning tree (stored by the XTI method) and the corresponding primal and dual solution. Moreover, since all operations performed on the basis tree, i.e. pivot operations 2 and 3, are restricted to the basis tree itself and the entering edge, these are carried out almost in the same way. The only difference is that the direction of edges in the TPS is implicitly given by the order of the incident nodes, while this direction has to be explicitly stored for general networks.

Thereby, the only significant adjustment in the implementation of Phase II for the NS is in pivot operation 1 (Algorithm 2). Here a pivot rule exploiting the special structure of the respective underlying graph should be chosen. In Section 4.3.4 we present such rules for the transportation problem, whereby with the Block Search Rules we also apply a rule originally introduced for general minimum cost flow problems in Grigoriadis (1986).

Moreover, as indicated above, a general difference lies in the storage of the topology of the underlying graph. Since the bipartite graph of the transportation graph is complete, all the relevant information is given by the matrix C as well as the supplies a and demands b. In contrast, one has to store, which nodes are connected by arcs along with the direction of these arcs for general uncapacitated minimum cost flow problems.

Lastly, Phase II yields another possible outcome, since the (MCF_u) is not guaranteed to be unbounded in our setup. Thereby, a minimum cost flow problem is unbounded if and only if it contains a directed cycle where the sum of the arc costs is negative (Ahuja et al. (1993)). Since there is no capacity on the arcs, unlimited amount of flow can be sent through the cycle to achieve arbitrary small objective function values.

The capacitated case

The additional introduction of capacities on the arcs entails further modifications in Phase II of the NS. To this end, assume a capacitated problem (MCF_c) is given for the graph $G = (V, \vec{E})$, cf. Section 1.4.3. First of all, the differences described for the uncapacitated case apply for this case as well. Furthermore, this naturally implies that along with the costs c_{ij}, the capacities o_{ij} of the arcs must be stored. It also implies a slight extension of the basis representation to account for arcs in which the flow is saturated, i.e. $x_{ij} = o_{ij}$. Therefore, basic solutions correspond to a triple of sets $(\mathcal{T}, Z, O)$ with $\mathcal{T} \dot\cup Z \dot\cup O = \vec{E}$ where it holds

$$\begin{aligned} 0 < \quad x_{ij} &< o_{ij} \iff \vec{ij} \in \mathcal{T} \\ x_{ij} &= 0 \iff \vec{ij} \in Z, \\ x_{ij} &= o_{ij} \iff \vec{ij} \in O. \end{aligned}$$

and $\mathcal{T}$ is a spanning tree of the underlying graph.

The elements of $\mathcal{T}$ are basic arcs while the elements in O and Z are non-basic arcs. Thus, the existence of non-basic arcs with positive flow in O yields that arcs with positive reduced costs are potential candidates to improve the current basic solutions when their flow is reduced as they enter the basis. This slightly alters the selection process of pivot operation 1 and the transition from one primal solution to the next in pivot operation 2. For a thorough and comprehensive explanation of the Network Simplex and XTI operations for (MCF_c), see e.g. Ahuja et al. (1993) and Glover et al. (1979).

4.5 Numerical analysis

In the following we want to numerically evaluate the TPS. To this end, the method was implemented as described above with the only variable components being the starting heuristics and the pivot rules. In order to keep our description of the results concise, we refer to the respective TPS implementations by the heuristics and pivot rule used, e.g. we denote the TPS with the NWCRXTI heuristics and the RMN pivot rule as NWCRXTI/RMN.

In doing so, we pursue two main objectives: The primary objective of this chapter is to identify the best implementation of the TPS. For this purpose, we analyze different combinations of heuristics and pivot rules an test them on the problem classes introduced in Section 2.3. The most promising combinations will then serve as a reference point for the pivot rules proposed in Chapter 5 which introduce concepts of Column Generation to the TPS. As former studies recommend to test pivot rules and heuristics together, cf. Srinivasan and Thompson (1973) and Glover et al. (1974b), we will incorporate all heuristics of Chapter 3 in combination with the pivot rules of Section 4.3.4 in our analysis. In case of the NWCR and the TMR, we only include the integrated variants, i.e. NWCRXTI and TMRXTI, as they proved to be slightly faster.

As a secondary objective, we want to quantify the overall advantage of a Network Simplex implementation specifically tailored to the transportation problem (the TPS), over the general NS and will therefore include the Network Simplex of ILOG CPLEX 12.6.2, henceforth referred to as CPLX, in our analysis. This represents a highly efficient implementation of the NS and further – as a widely used commercial solver – increases the comparability of our results to other studies. Since this method was clearly superior to all other CPLEX and GUROBI LP-solvers in our numerical investigations on the transportation problem, we refrain from presenting the results of further commercial LP-solvers. This decision is also based on the fact that similar observations were made in Schrieber et al. (2017). In the course of determining the benefit of a specialized NS implementation for transportation problems, we also examine the NWCR starting heuristic, which is well known to deliver poor results. However, as described in Section 4.4.1, it provides a good reference for the benefit of using a starting heuristic in Phase I.

We begin our analysis with basic tests on symmetrical and asymmetrical problem instances to identify the best combinations of heuristics and pivot rules for further investigations.

4.5.1 Basic symmetric evaluation

First, we will perform initial tests on symmetric instances of the problem that include all heuristics given in Chapter 3 and all pivot rules described in Section 4.3.4, except the Block Search Rules, which are equivalent to the Modified Row/Column Rules on symmetric instances and are therefore omitted.

Accordingly, 10^3 instances of dimension $m = n = 1024$ of the artificial problem classes SOLGEN and UNIFORM were solved. In addition, we created all problem instances of the DOTMARK class for the fixed resolution 32×32, which constitutes 450 additional transportation problems of the same size $m = n = 1024$, see Section 2.3.5. Consequently, each solved instance of a transportation problem (TP) included $2 \cdot 1024 \approx 2 \cdot 10^3$ nodes (constraints) and $1024^2 \approx 10^6$ edges (variables).

We averaged the results of each TPS combination and CPLX over all solved problems of a given class. The respective cpu time is presented in Figure 4.9 as well as Tables 4.3, 4.5 and 4.7. Furthermore, we documented the number of pivot steps performed and, if relevant, the percentage of pivot steps which were executed by the Shortlist Rules in Tables 4.4, 4.6 and 4.8.

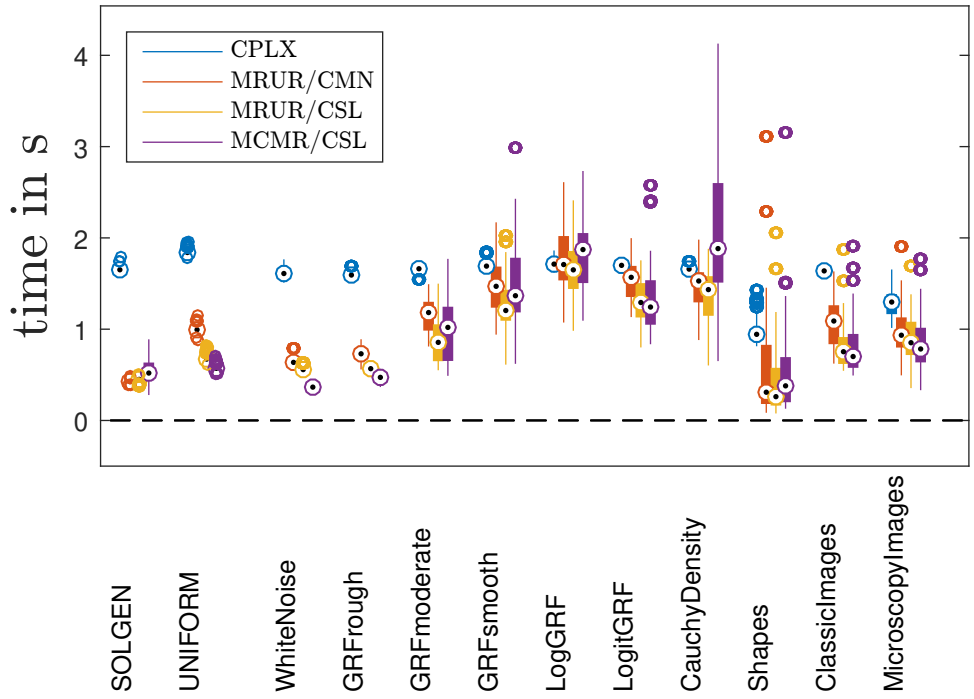


Figure 4.9: A boxplot of the cpu time for the symmetric instances of dimension $m = n = 1024$ of the best TPS combinations and CPLX on all problem classes is shown. It is observed that while the degree of difficulty is rather constant for the artificial classes, whereby the UNIFORM instances require on average more cpu time than the SOLGEN instances, the cpu time for the DOTMARK classes is subject to a higher variance. Moreover, whereas the TPS solvers are clearly superior on average, the CPLX is more robust and subject to less variance in cpu time.
Due to the large number of problem classes and algorithms, MATLAB's compact style is used for boxplots.

SOLGEN	RMN	CMN	RSL	CSL
NWCRXTI	0.79	0.78	0.76	0.75
TMRXTI	0.58	0.58	0.58	0.58
RMR	0.65	0.62	0.64	0.63
CMR	0.63	0.64	0.64	0.63
MRMR	0.55	0.53	0.54	0.54
MCMR	0.53	0.54	0.54	0.53
MMR	0.61	0.61	0.62	0.61
MRUR	**0.44**	**0.43**	**0.45**	**0.45**
LALC	0.77	0.77	0.77	0.76
D2PR	0.50	0.49	0.52	0.51

Table 4.3: Average cpu time on 10^3 symmetric SOLGEN instances of dimension $m = n = 1024$, for all combinations of starting heuristics (rows) and pivot rules (columns). For CPLX we documented an average running time of 1.65 seconds which is about four times slower than the best TPS combination MRUR/CMN with 0.43 seconds. In particular, dominance of the MRUR is observed which combines well with all pivot rules and halves the cpu time compared to the slowest heuristics. In contrast, the involved pivot rules perform at a surprisingly similar level.

SOLGEN		RMN	CMN	RSL	CSL
NWCRXTI	Pivot steps / 10^3	17.86	17.86	16.58	16.59
	SL steps in %	–	–	27	27
TMRXTI	Pivot steps	12.25	12.25	11.72	11.72
	SL steps	–	–	17	17
RMR	Pivot steps	13.04	12.81	12.48	12.28
	SL steps	–	–	15	28
CMR	Pivot steps	12.81	13.05	12.26	12.46
	SL steps	–	–	28	15
MRMR	Pivot steps	10.51	10.29	9.82	9.85
	SL steps	–	–	28	28
MCMR	Pivot steps	10.29	10.51	9.83	9.80
	SL steps	–	–	28	28
MMR	Pivot steps	9.58	9.58	9.33	9.31
	SL steps	–	–	18	18
MRUR	Pivot steps	5.37	5.36	5.02	5.03
	SL steps	–	–	30	29
LALC	Pivot steps	11.78	11.78	11.01	11.01
	SL steps	–	–	30	30
D2PR	Pivot steps	5.73	5.74	5.41	5.42
	SL steps	–	–	29	29

Table 4.4: Evaluation of the pivot steps, averaged over 10^3 samples of dimension $m = n = 1024$ of the SOLGEN class, for all combinations of starting heuristics (rows) and pivot rules (columns). The total number of pivot steps is given in thousands where the percentage of steps performed by the Shortlist Rules is shown below. In accordance with computation time, the MRUR and D2PR significantly reduce the number of required pivot steps of the TPS compared to the other heuristics. Interestingly, the Shortlist Rules do not significantly reduce the number of pivot steps and only account for a maximum share of 30% of the total pivots.

UNIFORM	RMN	CMN	RSL	CSL
NWCRXTI	2.00	1.99	1.17	1.16
TMRXTI	0.97	0.97	0.61	0.61
RMR	1.02	0.96	0.63	0.61
CMR	0.97	1.01	0.62	0.62
MRMR	0.91	0.90	**0.58**	**0.57**
MCMR	0.90	0.89	**0.58**	**0.57**
MMR	0.99	0.98	0.67	0.66
MRUR	1.00	1.00	0.68	0.67
LALC	0.91	0.90	0.67	0.67
D2PR	0.98	0.97	0.68	0.68

Table 4.5: Average cpu time on 10^3 symmetric UNIFORM instances of dimension $m = n = 1024$, for all combinations of starting heuristics (rows) and pivot rules (columns). The average cpu time of CPLX for the same problem instances amounted to 1.84 seconds. Whereas the heuristics (except the NWCRXTI) perform approximately on the same level, the Shortlist Rules clearly dominate the other pivot rules and achieve the best cpu time with 0.57 to 0.58 seconds in combination with the MCMR and the MRMR.

UNIFORM		RMN	CMN	RSL	CSL
NWCRXTI	Pivot steps / 10^3	46.94	46.90	22.62	22.61
	SL steps in %	–	–	100	100
TMRXTI	Pivot steps	19.79	19.80	9.24	9.23
	SL steps	–	–	100	100
RMR	Pivot steps	20.42	19.10	9.12	8.84
	SL steps	–	–	100	100
CMR	Pivot steps	19.12	20.44	8.85	9.15
	SL steps	–	–	100	100
MRMR	Pivot steps	17.66	17.67	8.06	8.05
	SL steps	–	–	100	100
MCMR	Pivot steps	17.60	17.63	8.03	8.03
	SL steps	–	–	100	100
MMR	Pivot steps	17.33	17.31	7.87	7.87
	SL steps	–	–	100	100
MRUR	Pivot steps	17.42	17.41	7.92	7.92
	SL steps	–	–	100	100
LALC	Pivot steps	13.35	13.35	6.07	6.08
	SL steps	–	–	100	100
D2PR	Pivot steps	16.04	16.05	7.23	7.23
	SL steps	–	–	100	100

Table 4.6: Evaluation of the pivot steps, averaged over 10^3 samples of dimension $m = n = 1024$ of the UNIFORM class, for all combinations of starting heuristics (rows) and pivot rules (columns). The total number of pivot steps is given in thousands where the percentage of steps performed by the Shortlist Rules is shown below. In line with computation time, the number of pivot steps is generally significantly higher than for SOLGEN instances. In particular, it is observed that all Shortlist pivots always amount to 100% of the total pivots, that is, the solution on the shortlists was already optimal in all cases.

DOTMARK	RMN	CMN	RSL	CSL
NWCRXTI	1.74	1.77	1.40	1.45
TMRXTI	1.62	1.71	1.29	1.36
RMR	2.66	1.89	1.82	1.51
CMR	1.87	2.80	1.48	1.87
MRMR	1.30	1.30	1.07	1.11
MCMR	1.22	1.29	1.06	1.08
MMR	1.72	1.82	1.38	1.44
MRUR	1.16	1.14	**0.99**	**0.98**
LALC	1.35	1.45	1.15	1.19
D2PR	1.67	1.72	1.40	1.43

Table 4.7: Evaluation of the cpu time, averaged over 450 instances with dimension $m = n = 1024$ of the DOTMARK class, for all combinations of starting heuristics (rows) and pivot rules (columns). The average cpu time of CPLX for the same problem instances amounted to 1.56 seconds. The best TPS combination is given by the MRUR and the RSL/CSL with a total cpu time of 0.99 and 0.98 seconds. Generally, the MRUR and the Shortlist Rules deliver the best results for this problem class.

DOTMARK		RMN	CMN	RSL	CSL
NWCRXTI	Pivot steps / 10^3	34.43	34.89	25.89	27.05
	SL steps in %	–	–	55	54
TMRXTI	Pivot steps	29.35	31.76	23.22	24.99
	SL steps	–	–	59	58
RMR	Pivot steps	51.86	36.16	33.79	27.47
	SL steps	–	–	54	56
CMR	Pivot steps	35.71	55.22	26.90	35.62
	SL steps	–	–	58	55
MRMR	Pivot steps	23.41	24.22	18.51	19.80
	SL steps	–	–	62	62
MCMR	Pivot steps	22.25	23.45	18.37	19.19
	SL steps	–	–	63	62
MMR	Pivot steps	29.06	31.64	22.47	24.27
	SL steps	–	–	59	58
MRUR	Pivot steps	18.36	18.44	14.07	14.24
	SL steps	–	–	55	55
LALC	Pivot steps	22.11	24.13	17.16	18.32
	SL steps	–	–	63	62
D2PR	Pivot steps	28.40	29.19	22.09	23.35
	SL steps	–	–	56	55

Table 4.8: Evaluation of the pivot steps, averaged over 450 instances of dimension $m = n = 1024$ of the DOTMARK class, for all combinations of starting heuristics (rows) and pivot rules (columns). The total number of pivot steps is given in thousands where the percentage of steps performed by the Shortlist Rules is shown below. Generally, it required the highest number of pivot steps ranging from 14000 for MRUR with the Shortlist Rules to 55000 for CMR/CMN. Moreover, the percentage of pivot steps performed by Shortlist Rules is higher compared to the SOLGEN instances and amounts to 54% – 63%.

Results

Before we specifically review the performance of the methods on each of the three problem classes, we give our general observations, which, for reasons of clarity, we divide into major and minor observations:

Major observations

1. While we observed that the results depend on the problem classes and that no universally best TPS combination could be determined, the heuristics MRUR, MRMR and MCMR were competitive on all problem classes. Since we explicitly preselected efficient methods in Section 4.3.4, all pivot rules performed at a high level, as expected. However, a significant reduction of cpu time could be observed when applying the Shortlist Rules for the UNIFORM and DOTMARK instances.

2. Moreover, the TPS compared favorably to CPLX in this first evaluation. The total cpu time of the best combinations was between 3 to 4 times faster on the artificial problem classes and 1.5 times faster on the DOTMARK instances. As an illustration, we show the average cpu time of MRUR/CSL compared to CPLX below:

Times in s	SOLGEN	UNIFORM	DOTMARK
MRUR/CSL	0.45	0.67	0.98
CPLX	1.65	1.84	1.56

   Conversely, CPLX proved to be significantly more robust and contained less variance in the solution time, see Figure 4.9. In the further course of our numerical analysis we examine how this comparison develops in higher dimensions. Lastly, a very interesting observation was that DOTMARK problems constitute the easiest problem class for CPLX, while they were the hardest to solve for TPS methods. Since we do not know the details of the CPLX implementation, we cannot explain this observation entirely. However, one reason is certainly the robustness of CPLX, whereas the average time of the TPS methods suffers from some outliers, see 4.9.

3. In terms of total cpu time, the DOTMARK instances are subject to the largest variance and contain the most difficult as well as the easiest problems. The artificially constructed problems are more homogeneous in degree of difficulty, whereby the UNIFORM instances (on average) are somewhat more difficult to solve than the SOLGEN instances. We illustrate this in the boxplot of Figure 4.9.

4. We included the NWCRXTI method in our study to measure the general impact of starting heuristics, in particular in comparison to the application of the general NS to the transportation problem, see Section 4.4.1. Comparing the best runtime of a pivot rule combined with all starting heuristics and the runtime of this pivot rule with the NWCRXTI, the cpu time was reduced by 25% to 50% in all cases. This constitutes a first argument to apply the TPS instead of the NS when solving transportation problems and a possible explanation for the promising performance compared to CPLX.

5. While RMN and CMN showed satisfactory results, the Shortlist Rules proved particularly promising. As expected, they achieved the best results with the classes UNIFORM and DOTMARK, as they were specifically developed for the cost structure of these problems. The first subphase of the Shortlist Rule, see Section 4.3.4, is restricted to a subset of the variables (the shortlists), which requires fewer pivot steps to optimally solve this reduced problem. Accordingly, by design, the Shortlist Rule excels whenever an optimal solution is already included in the shortlists. More precisely, if the shortlists contain all variables that assume non-zero values in an optimal solution (e.g. all basic variables of an optimal basic solution), the shortlist solution is also optimal for the full set of variables. In this case, the dimension of the problem is effectively reduced for the complete solution process, which significantly reduces the number of necessary pivot steps. Furthermore, beyond this extreme case, it is reasonable to assume that the effectiveness of the Shortlist Rules generally increases when the percentage of pivot steps executed on the shortlists is large. This means that a larger part of the solution process is

performed on a reduced problem that can be solved in fewer pivot steps, which will also cause the number of total pivot steps to be smaller. The closer[8] the solution on the shortlists is to an optimal solution, the fewer pivot steps are required on the full set of variables.

In particular, the uniformly distributed costs of the UNIFORM class generate such a structure, since an optimal solution is included in the shortlists for every problem instances, see Tables 4.5 and 4.6. Naturally, this structure is also likely for the DOTMARK instances, where the origins i and destinations j are represented by pixels in two images and the respective costs c_{ij} are given by the squared euclidean distance of the 2-dimensional pixel positions. This implies that, for each pixel, the adjacent and close pixels in the other image yield promising variables of low costs. As a result, the Shortlist pivots account for a high percentage between 54% and 63% of the total pivots in Table 4.8. The least share of Shortlist pivots, i.e. up to 30%, is observed for the SOLGEN instances, which suggests that the optimal solutions generated tend to contain a larger share of variables with comparably high costs and, hence, these variables are excluded from the shortlists. This observation constitutes the essential motivation for the pivot rules introduced in the context of Column Generation in Chapter 5.

6. Apart from the obvious fact that faster heuristics combine favorably with the Shortlist Rules for all instances where the Shortlist Rule (almost) optimally solves the problem, we did not determine any significant dependencies between applied heuristics and pivot rules. This contrast to the earlier studies of Srinivasan and Thompson (1973) and Glover et al. (1974b) is likely explained by the preselection (and thus the small number) of efficient pivot rules.

   In particular, we did not observe a significant interrelation between row and column variants of successively applied heuristics and pivot rules in general. The only exception is given by the RMN and CMN on the

[8] Here the term close is used in the sense that only a few pivot steps are needed to convert this solution into an optimal one.

DOTMARK classes. Here, combining the row and column variants of the heuristic and the pivot rules yields much better results than using two rows or column variants in one algorithm, as discussed in the detailed review for the DOTMARK instances below. In addition, this phenomenon can be observed to a minimal extent for RMN, CMN, MRMR and MCMR. However, this interrelation is not part of the central research questions of this thesis. Furthermore, the RMN and CMN in general achieve poor results and the impact for the MRMR and MCMR is negligible. Hence, we will not discuss this phenomenon for symmetric problems in the further course of this thesis.

Minor observations

1. The heuristics MRUR and D2PR apply a similar approach where the primal solution is found by minimizing the complementary slackness to a given dual solution. While the according dual solution in the D2PR is feasible, the dual solution in the MRUR is chosen to mimic the multipliers in the Simplex and is in general not feasible. This this similarity is reflected in the cpu time on the artificial problem instances. However, on the DOTMARK instances the MRUR interestingly performs significantly better than the D2PR.

2. A note on LALC: As mentioned in Section 3.5, this method could not be implemented optimally in MATLAB. The pivot steps documented in Tables 4.12 and 4.14 show that the corresponding starting solutions require the fewest pivot steps in order to be converted into an optimal solution for the UNIFORM class and also to achieve low numbers of pivot steps for the DOTMARK class. Therefore, this method is a suitable candidate for further investigations in the context of other programming languages.

3. Integrating the XTI initialization in the TMRXTI heuristic results in approximately a 5% reduction of the cpu time. However, this effect is canceled out by the selection of basis variables in the TMRXTI, which proved to be ineffective compared to the best performing heuristics. Accordingly, the method only performs competitive in some cases, which

leads us to not investigate this approach any further.

4. Due to the column-major memory design of MATLAB, we implemented row variants of the heuristics and pivot rules by executing their column equivalents on the transposed problem, as described in Sections 3.5 and 4.3.4. This is reflected in a minimal increase in the cpu time of the row variants, which is based on the transposition of C and the swapping of a and b.

SOLGEN Let us discuss the results on the SOLGEN instances in detail: The best overall time, i.e. 0.43 seconds, was achieved by the MRUR/CMN combination. This is about four times as fast as the average time of CPLX which amounted 1.65 seconds and was undercut by all combinations of pivot rules and heuristics where the worst combination was given by the NWCRXTI/RMN with 0.79 seconds.

Surprisingly, the total cpu time is almost independent of the pivot rules used, with the largest difference being 0.04 seconds between NWCR/RMN and NWCR/CSL. In particular, we did not observe a significant dependence in the starting heuristics and the pivot rules for the SOLGEN instances. In contrast to that, we observe a rather surprising dominance of the MRUR rule compared to the remaining heuristics with almost constant total cpu time of 0.43 to 0.45 seconds for all pivot rules. The second best time by the D2PR, ranges from 0.49 seconds to 0.52 seconds and is followed by the MRMR and MCMR with cpu time between 0.53 and 0.55 seconds and the TMRXTI with 0.58 seconds.

The average number of pivot steps ranges from approximately 5000 to 18000. In accordance with the cpu time, the MRUR realizes the smallest number of pivot steps. Interestingly, the application of the Shortlist Rules did not significantly reduce the total number of pivots steps, despite the fact that they account for up to 30% of all pivots. Moreover, this share of pivot steps for the Shortlist Rules was significantly smaller compared to the other problem classes.

UNIFORM As expected, the UNIFORM instances proved to be somewhat more difficult than the SOLGEN instances. Excluding the NWCRXTI, all starting heuristics performed approximately on the same level with total cpu time ranging between 0.57 and 1.02 seconds. The best overall time is achieved by a combination of MRMR and MCMR with the Shortlist Rules and amounts to 0.57 or 0.58 seconds. The average time of CPLX, 1.84 seconds, exceeded this by factor 3 and was at least 80% higher than for all TPS combinations that did not contain the NWCRXTI. The number of pivot steps also increased compared to the SOLGEN instances. Here, the NWCRXTI represents again the extreme case with up to 45000 steps, whereas all other heuristics achieve numbers approximately in the range 6000 to 20000.

In particular, we observe a rather dominant performance of the Shortlist Rules, which achieved the best results for all starting heuristics and was almost twice as fast as the concurrent pivot rules. This fact can be easily explained by Table 4.6, where it is observed that in fact all instances were solved optimally by the Shortlist Rules, i.e. an optimal solution was already contained in the shortlists. Accordingly, the number of necessary pivot steps is approximately halved.

Consequently, in comparison to the SOLGEN instances, we observe the contrary situation, i.e. the results are (somewhat) independent of the selected heuristics but strongly affected by the pivot rule used. Thus, the combination of heuristics and pivot rules is a minor factor for the UNIFORM class: As the problem is quickly solved by the Shortlist Rules, the inherent cpu time of the heuristics plays a more important role and, hence, faster heuristics generally produce the better overall results. For instance the LALC method is among the best heuristics with respect to the RMN and CMN rules but yields some of the worst results (with the exception of the NWCRXTI) when it is combined with the Shortlist Rules.

DOTMARK Finally, we evaluate the results on the DOTMARK instances: Generally, this class required the highest average cpu time and number of pivot steps as it includes the most difficult problem instances. Here, the best TPS combination is again given by the MRUR/CSL with an averaged cpu time of

0.98 seconds, which is 50% faster than the 1.56 seconds of CPLX. Moreover, as for the SOLGEN instances, the MRUR generally dominates the other starting heuristics with cpu time between 0.98 and 1.16 seconds. Comparably good results are only achieved by the MRMR and MCMR with cpu time ranging from 1.06 to 1.30 seconds. Interestingly, and in contrast to the observations on the SOLGEN instances, the D2PR performs significantly worse than the MRUR although they apply very similar concepts, see minor observation 1.

Surprisingly, the worst combinations do not include the NWCR method but are represented by RMR/RMN and CMR/CMN. Generally, we observe that applying a row (column) pivot rule after RMR (CMR) delivers worse results than combining row and column methods. Since the RMR/CMN and CMR/RMN perform significantly better (but still worse than the NWCR heuristics), this is the only case for all problem classes in which the examination of columns or rows has a significant influence on performance for symmetric problems. More interestingly, this observation is not matched for the MRMR and MCMR which perform approximately on the same level for the row and column variants of the pivot rules. Hence, this phenomenon is due to the RMN (CMN) distributing all the supply (demand) of one origin (destination) before considering the next one.

This class required the highest number of pivot steps, i.e. 14000 when combining the MRUR with Shortlist Rules and 55000 for CMR/CMN. Moreover, the percentage of pivot steps performed by Shortlist Rules is higher compared to the SOLGEN instances and amounts to 54% – 63%.

4.5.2 Basic asymmetric evaluation

To complete our basic evaluation, we perform tests on asymmetric instances of the transportation problem. For this purpose, we include the Block Search rules RBLK and CBLK, as they were introduced exclusively for these problems (see Section 4.3.4), in addition to the heuristics and pivot rules already applied in the symmetric case.

For DOTMARK, we created asymmetric problems of dimension $m = 4096$

and $n = 1024$, generated by comparing images of size 64×64 and 32×32, which implies approx. $5 \cdot 10^3$ nodes (constraints) and approx. $4 \cdot 10^6$ edges (variables) in (TP). As in the symmetric case, we generated all 45 instances for each of the 10 DOTMARK classes for these resolutions. As the rest of this thesis will focus on the symmetric case, we try to keep the asymmetric evaluation concise. Hence, in order to keep the asymmetric evaluation clear, we go to a higher degree of asymmetry for the investigation of the artificial problem classes. So we generate 1000 problems instances of size $m = 4096$ and $n = 64$, i.e. $m = n^2$, for SOLGEN and UNIFORM, which accordingly results in approx. $4 \cdot 10^3$ nodes (constraints) and approx. $2.6\dot{1}0^5$ edges (variables) in (TP). We averaged the results each TPS combination and CPLX over all solved problems of a given class. As in the symmetric case, we documented the cpu time in Tables 4.9, 4.11, 4.13 and Figure 4.10 as well as the pivot steps in Tables 4.10, 4.12 and 4.14.

Results

Let us summarize the results on the asymmetric problem instances. As expected, the cpu time is generally higher for the DOTMARK instances, as these problems involve more constraints and variables. Interestingly, while the cpu time of the TPS combinations for the UNIFORM instances was, as in the symmetric case, generally higher than for SOLGEN instances, the corresponding cpu time of CPLX was almost identical, see Figure 4.10. Furthermore, in general, we find that column variants of heuristics and pivot rules are more efficient than their row counterparts. This holds for the artificial instances with higher degree of asymmetry as well as for the DOTMARK instances and is clearly explained by the fact that the column variants consider larger sequences of variables in the pivot step (size is a function of m) than the row variants (size is a function of n). As in the symmetric case, the MRUR and MCMR as well as the CSL delivered the best overall results. Surprisingly, the TMRXTI/CSL achieved the best cpu time for the UNIFORM instances.

While the TPS was competitive to CPLX, this comparison was significantly worse than in the symmetrical case, see Figure 4.10. In particular the Block

Search rules could not achieve good results and only performed reasonably well on the SOLGEN instances. Moreover, a comparison with an unvectorized implementation of CMN indicates that this is only to a very small extent caused by the implementation in MATLAB, but is mainly due to the inherent concept of the rule. Regarding the Shortlist Rules, their good performance in comparison to RMN and CMN is somewhat surprising, as they were specifically introduced for symmetric problems in Gottschlich and Schuhmacher (2014). Accordingly, we assume that the selection of parameters l_{SL}, k_{SL} and p_{SL} has potential for improvement, since these have been optimized exclusively for the symmetrical case, see Gottschlich and Schuhmacher (2014).

However, to reduce the scope of this analysis, we will hereinafter refrain from a more thorough analysis of asymmetric problems and suitable implementations of the TPS in this case. This is in particular due to the fact that the pivot rules in Chapter 5 will be proposed solely for symmetric problems.

4.5.3 Consequences

On the basis of these initial observations, we draw the following consequences before we continue with more extensive investigations:

1. Since the Block Search rules did not yield satisfactory results and we refrain from an extensive optimization of the Shortlist parameters for different degrees of asymmetry, we restrict the further analysis to symmetric problem instances. Consequently, as they were introduced specifically for the asymmetric case, we also omit the Block Search rules in further investigations.
2. As they are slightly faster, we restrict our investigations to the column variants of heuristics and pivot rules – see Section 3.5, 4.3.4 and minor observation 4 for symmetric problems.
3. Lastly, on the basis of our results, we select the heuristics MRUR and MCMR as well as the pivot rules CMN and CSL for further study.

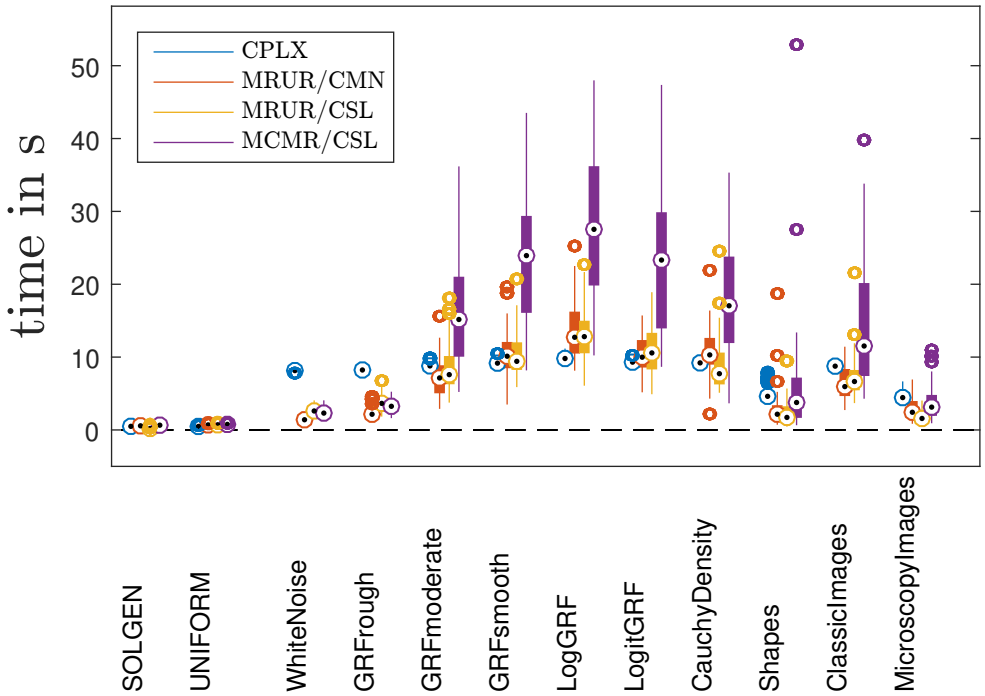


Figure 4.10: A boxplot of the cpu time for the asymmetric instances of dimension $m = 4096$ and $n = 64$ for the SOLGEN as well as the UNIFORM instances and $m = 4096$ and $n = 1024$ for the DOTMARK instances. As expected, the cpu time is generally higher for the DOTMARK instances, as they involve more constraints and variables. Moreover, the TPS solvers perform worse here than on the symmetric instances and are comparable to CPLX in average computing time, but are significantly less robust.

SOLGEN	RMN	CMN	RBLK	CBLK	RSL	CSL
NWCRXTI	2.05	0.78	1.76	0.97	1.63	0.63
TMRXTI	1.15	0.53	1.10	0.62	1.00	0.52
RMR	1.55	0.61	1.33	0.75	1.33	0.55
CMR	1.26	0.64	1.20	0.74	1.09	0.62
MRMR	1.53	0.61	1.32	0.74	1.32	0.55
MCMR	0.38	0.23	0.34	0.28	0.35	0.24
MMR	0.88	0.46	0.84	0.53	0.82	0.45
MRUR	0.29	**0.20**	0.25	0.24	0.27	**0.20**
LALC	1.25	1.08	1.23	1.13	1.23	1.08
D2PR	0.34	0.24	0.29	0.28	0.32	0.23

Table 4.9: The cpu time, averaged over 10^3 instances of the SOLGEN class with dimension $m = n^2 = 4096$ and $n = 64$, for all combinations of starting heuristics (rows) and pivot rules (columns). The average cpu time of CPLX for the same problem instances amounted to 0.49 seconds. The MRUR is again clearly dominant while the CMN and CSL perform at a similar level.

SOLGEN		RMN	CMN	RBLK	CBLK	RSL	CSL
NWCRXTI	Pivot steps / 10^3	26.65	10.26	27.73	14.64	23.19	8.84
	SL steps in %	–	–	–	–	47	36
TMRXTI	Pivot steps	15.72	6.40	15.93	8.22	14.18	6.39
	SL steps	–	–	–	–	41	2
RMR	Pivot steps	17.98	6.34	18.80	9.17	16.03	5.90
	SL steps	–	–	–	–	41	35
CMR	Pivot steps	15.83	6.45	16.10	8.21	13.94	6.46
	SL steps	–	–	–	–	47	3
MRMR	Pivot steps	17.78	6.27	18.43	8.98	15.92	5.82
	SL steps	–	–	–	–	42	35
MCMR	Pivot steps	1.94	0.95	2.41	1.21	1.80	0.96
	SL steps	–	–	–	–	51	17
MMR	Pivot steps	9.69	3.94	10.32	5.12	9.32	3.95
	SL steps	–	–	–	–	35	7
MRUR	Pivot steps	0.63	0.43	0.76	0.54	0.57	0.41
	SL steps	–	–	–	–	52	31
LALC	Pivot steps	2.80	1.31	3.44	1.68	2.66	1.33
	SL steps	–	–	–	–	54	14
D2PR	Pivot steps	1.07	0.74	1.32	0.90	0.98	0.70
	SL steps	–	–	–	–	49	31

Table 4.10: Evaluation of the pivot steps, averaged over 10^3 samples of dimension $m = n^2 = 4096$ and $n = 64$ of the SOLGEN class, for all combinations of starting heuristics (rows) and pivot rules (columns). The total number of pivot steps is given in thousands, whereas the percentage of steps performed by the Shortlist Rules is shown below. In accordance with cpu time, the MRUR method significantly reduces the number of required pivot steps of the TPS compared with the other heuristics. Interestingly, the Shortlist Rules do not significantly reduce the number of pivot steps and only account for a maximum share of 30% of the total pivots.

UNIFORM	RMN	CMN	RBLK	CBLK	RSL	CSL
NWCRXTI	8.91	1.34	2.37	2.08	1.99	1.12
TMRXTI	1.58	0.70	0.84	1.06	**0.65**	**0.66**
RMR	3.30	0.73	1.26	1.13	0.85	0.71
CMR	1.67	0.80	0.94	1.17	0.75	0.77
MRMR	3.09	0.72	1.17	1.11	0.82	0.70
MCMR	1.51	0.72	0.86	1.05	0.68	0.72
MMR	1.51	0.73	0.87	1.06	0.69	0.73
MRUR	1.51	0.73	0.87	1.06	0.69	0.72
LALC	1.39	1.14	1.23	1.26	1.15	1.15
D2PR	1.73	0.75	0.91	1.10	0.72	0.74

Table 4.11: The cpu time, averaged over 10^3 instances of the UNIFORM class with dimension $m = n^2 = 4096$ and $n = 64$, for all combinations of starting heuristics (rows) and pivot rules (columns). As for the SOLGEN instances, the average cpu time of CPLX for the same problem instances amounted to 0.49 seconds.

UNIFORM		RMN	CMN	RBLK	CBLK	RSL	CSL
NWCRXTI	Pivot steps / 10^3	147.92	18.41	38.35	33.59	34.18	16.04
	SL steps in %	–	–	–	–	100	24
TMRXTI	Pivot steps	23.72	8.61	12.31	15.31	9.32	8.48
	SL steps	–	–	–	–	100	17
RMR	Pivot steps	49.02	8.02	17.88	15.27	11.37	8.02
	SL steps	–	–	–	–	100	18
CMR	Pivot steps	23.72	8.57	12.33	15.26	9.31	8.50
	SL steps	–	–	–	–	100	17
MRMR	Pivot steps	45.76	7.84	16.38	14.92	10.89	7.92
	SL steps	–	–	–	–	100	18
MCMR	Pivot steps	20.79	7.47	11.04	13.39	8.11	7.81
	SL steps	–	–	–	–	100	20
MMR	Pivot steps	20.77	7.45	11.03	13.38	8.10	7.79
	SL steps	–	–	–	–	100	20
MRUR	Pivot steps	20.73	7.45	11.03	13.37	8.10	7.71
	SL steps	–	–	–	–	100	21
LALC	Pivot steps	5.15	2.17	3.71	3.67	2.01	2.34
	SL steps	–	–	–	–	100	16
D2PR	Pivot steps	23.90	7.57	11.42	13.75	8.35	7.77
	SL steps	–	–	–	–	100	18

Table 4.12: Evaluation of the pivot steps, averaged over 10^3 samples of dimension $m = n^2 = 4096$ and $n = 64$ of the UNIFORM class, for all combinations of starting heuristics (rows) and pivot rules (columns). The total number of pivot steps is given in thousands where the percentage of steps performed by the Shortlist Rules is shown below. In line with cpu time, the number of pivot steps is generally significantly higher than for SOLGEN instances. In particular, it is observed that all pivots are Shortlist pivots, that is, the solution on the shortlists was already optimal in all cases.

DOTMARK	RMN	CMN	RBLK	CBLK	RSL	CSL
NWCRXTI	12.43	9.56	14.23	12.41	9.30	8.58
TMRXTI	12.24	9.31	14.96	12.79	9.99	8.18
RMR	20.19	10.93	23.80	17.06	14.50	10.05
CMR	13.58	17.45	17.46	23.21	11.44	13.91
MRMR	12.62	7.82	16.44	12.72	9.03	**6.86**
MCMR	9.30	7.02	11.48	11.11	7.52	**6.37**
MMR	12.68	9.80	15.19	13.21	10.12	8.58
MRUR	7.84	7.08	12.79	12.26	**6.63**	**6.16**
LALC	10.27	7.45	12.45	11.53	8.02	**6.83**
D2PR	12.90	9.88	15.02	13.10	10.56	8.84

Table 4.13: The cpu time, averaged over 10^3 instances of the DOTMARK class with dimension $m = 4096$ and $n = 1024$, for all combinations of starting heuristics (rows) and pivot rules (columns). The average cpu time of CPLX for the same problem instances amounted to 8.12 seconds.

DOTMARK		RMN	CMN	RBLK	CBLK	RSL	CSL
NWCRXTI	Pivot steps / 10^3	128.87	72.72	51.54	47.05	81.71	66.06
	SL steps in %	–	–	–	–	56	39
TMRXTI	Pivot steps	110.56	65.74	46.66	39.72	84.82	63.83
	SL steps	–	–	–	–	59	47
RMR	Pivot steps	201.76	80.78	79.34	49.63	130.08	77.95
	SL steps	–	–	–	–	59	44
CMR	Pivot steps	131.79	135.27	56.10	63.43	99.47	111.62
	SL steps	–	–	–	–	59	39
MRMR	Pivot steps	115.14	53.98	48.94	37.39	75.79	51.79
	SL steps	–	–	–	–	59	48
MCMR	Pivot steps	82.26	46.53	37.29	33.57	60.68	47.49
	SL steps	–	–	–	–	63	52
MMR	Pivot steps	109.85	65.73	45.42	38.37	81.78	62.62
	SL steps	–	–	–	–	59	47
MRUR	Pivot steps	63.79	47.05	39.73	32.08	47.10	41.97
	SL steps	–	–	–	–	55	39
LALC	Pivot steps	81.98	43.34	36.63	31.65	55.95	43.48
	SL steps	–	–	–	–	63	51
D2PR	Pivot steps	115.20	66.26	45.22	38.73	85.23	63.51
	SL steps	–	–	–	–	56	42

Table 4.14: Evaluation of the pivot steps, averaged over 10^3 samples of dimension $m = 4096$ and $n = 1024$ of the DOTMARK class, for all combinations of starting heuristics (rows) and pivot rules (columns). The total number of pivot steps is given in thousands, whereas the percentage of steps performed by the Shortlist Rules is shown below.

4.5.4 Asymptotic symmetric evaluation

Let us analyze the asymptotic behavior of the best TPS combinations compared to CPLX on symmetric problem instances. As explained in Section 2.3.5, we restrict the DOTMARK evaluations to a maximal resolution of 64×64, i.e. $m = n = 4096$, for CPLEX and 128×128, i.e. $m = n = 16384$ for the TPS. Symmetric problems for resolutions 256×256 could not be solved on our test system due to memory limitations, see Section 2.1. Hence, there are only three problem sizes available for the DOTMARK classes; we will present the results for the two higher resolutions in the following section. The complexity analysis will be limited to the artificial classes where more data points can be generated. Here, we have solved instances of SOLGEN and UNIFORM up to a dimension of $m = n \leq 1.5 \cdot 10^4$.

Results for DOTMARK

We begin by presenting the results for the DOTMARK classes in Figures 4.11 and 4.12. While the TPS (MRUR/CSL) was superior in all DOTMARK classes in our basic evaluation (see Section 4.5.1) for resolution 32×32 and $m = n = 1024$, CPLX is faster on 6 out of 10 classes for the increased dimension $m = n = 4096$. The variance of the CPLX is again much lower than with TPS, its average computation time is close to 50 seconds or less. In contrast, the average cpu time of the TPS varies between 5 and 150 seconds, where LogGRF was the class with the highest complexity. If the individual dotmark classes are sorted according to their solution time with the TPS, we obtain the same results as for the basic evaluations.

These results are consistent with Schrieber et al. (2017) and reiterate the advantages of using specific start heuristics in the TPS implementation by the following argument: The exact implementation of CPLX is not publicly available. However, it is known that there is no special implementation for transportation problems and it can be assumed that the pivot steps are implemented very efficiently. The latter assumption is supported by the high memory load (see Section 4.5.4). Thus, the good results of the TPS for $m = n = 1024$ are

most likely explained by the benefit of the starting heuristic. This advantage is outweighed by the efficient implementation of pivot steps as the number of pivot steps increases.

Lastly, the low memory requirements of the TPS enables us to report the cpu time for $m = n = 16384$ in Figure 4.12 – the largest instances solved for this thesis. The average computation time increased to range from 10 seconds to 600 seconds. We observe the same order of complexity for the individual classes; LogGRF is again the most difficult class.

Results for artificial problem classes

For a complexity analysis of our algorithms compared to CPLX, we solved 100 instances of the SOLGEN and UNIFORM class for dimensions $m \leq 6 \cdot 10^6$. The averaged cpu time is displayed in Table 4.15 and Figure 4.13.

Assuming a relation $t = k \cdot m^\alpha$ between the cpu time t and the number of supplies m, we estimated the following parameter values by linear regression: In case of SOLGEN, the exponents α were either 2.16 or 2.17 for all TPS variants, while we obtained $\alpha = 2.3$ for CPLX. On the UNIFORM class we observed a larger interval for the TPS exponents ranging from 2.29 to 2.36 – the best variant being MRUR/CSL. All TPS variants were again able to beat CPLX which achieved $\alpha = 2.56$ – see Table 4.16 for a comprehensive display of the results for both classes.

Largest solved instances and memory requirements For our large-scale analysis, we restrict our investigations to the SOLGEN instances, since these could be solved fastest on average. Moreover, we could not solve symmetric instances with CPLX, where m was significantly larger than $6 \cdot 10^3$ due to memory limitations[9]. Therefore, we only present the cpu time of the MRUR/CSL[10] for dimension $10^3 \leq m \leq 1.5 \cdot 10^4$ in Figure 4.14. Thereby, we obtained an even better regression exponent of $\alpha = 2.13$ for MRUR/CSL for SOLGEN.

[9] This is with respect to the hardware specifications given in Section 2.1.

[10] The results of the other TPS combinations presented in Figure 4.13 were comparable.

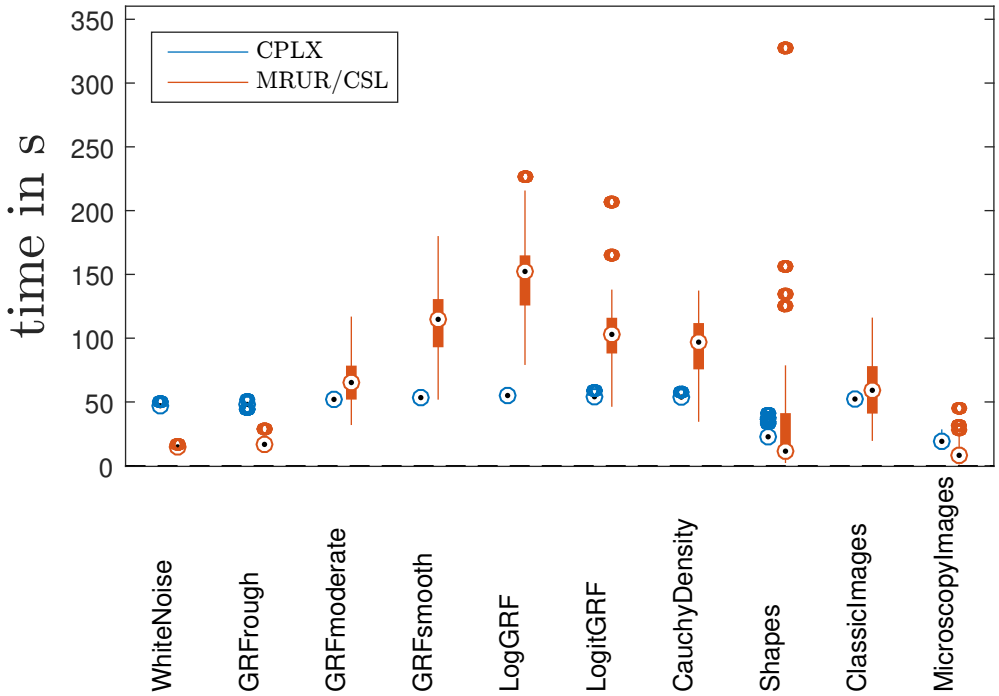


Figure 4.11: A boxplot of the cpu time for the symmetric instances of dimension $m = n = 4096$ of TPS and CPLX on all DOTMARK classes is shown. It is observed that while the degree of difficulty is rather constant for CPLX, the average cpu time of TPS is subject to a higher variance. Moreover, CPLX is now clearly superior to the TPS in terms of cpu time. The order of the complexity of the individual classes for TPS is comparable to Figure 4.9 and in accordance with Schrieber et al. (2017).

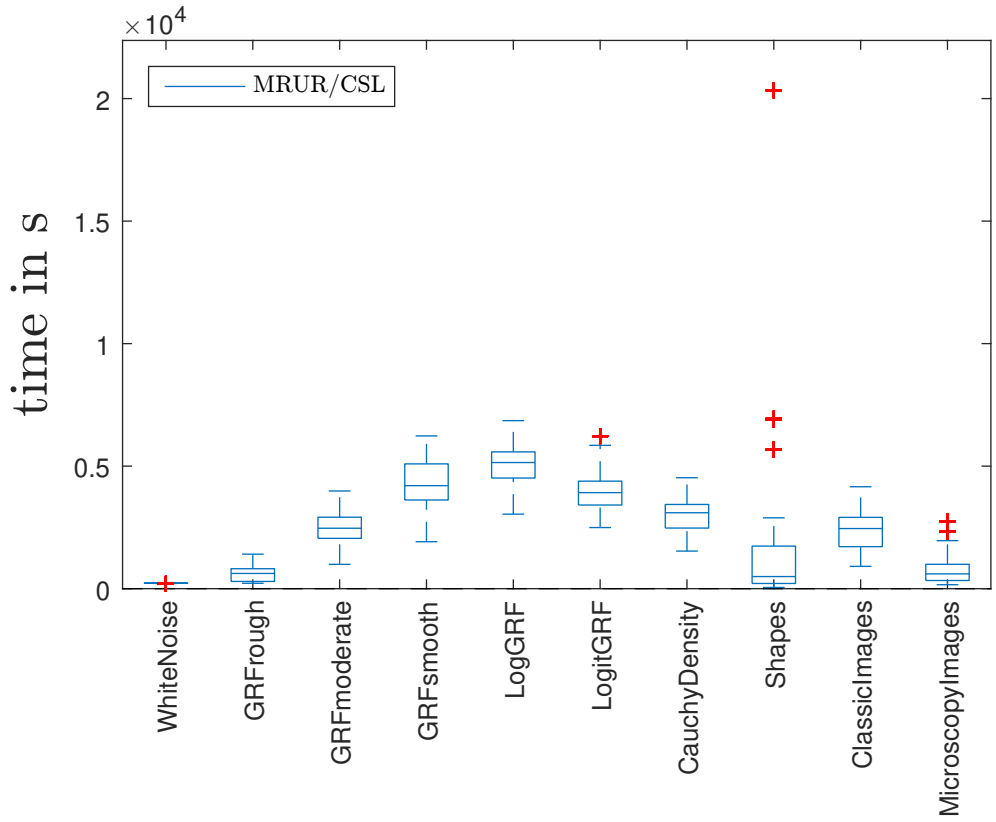


Figure 4.12: A boxplot of the cpu time for the symmetric instances of dimension $m = n = 16384$ on all DOTMARK classes is shown. Due to memory limitations, only TPS cpu time is reported for this problem dimension. The order of the complexity of the individual classes is again comparable to Figure 4.11.

SOLGEN								
m / 10^3	0.1	0.5	1	2	3	4	5	6
CPLX	0.01	0.25	1.25	5.94	15.14	30.86	49.80	76.74
MCMR-CMN	0.01	0.13	0.48	2.32	4.97	9.87	19.70	27.45
MCMR-CSL	0.01	0.12	0.47	2.36	5.05	9.88	19.15	26.67
MRUR-CMN	0.01	0.10	0.38	1.82	4.07	7.82	15.03	21.04
MRUR-CSL	0.01	0.10	0.39	1.93	4.28	8.17	15.23	21.01

UNIFORM								
m / 10^3	0.1	0.5	1	2	3	4	5	6
CPLX	0.01	0.28	1.49	8.26	24.10	53.61	94.92	158.84
MCMR-CMN	0.01	0.20	0.76	4.49	10.19	20.97	45.43	64.36
MCMR-CSL	0.01	0.11	0.51	3.18	7.23	13.15	24.47	34.52
MRUR-CMN	0.01	0.22	0.86	4.88	11.10	22.59	48.04	67.97
MRUR-CSL	0.01	0.14	0.61	3.58	8.21	14.90	27.18	38.40

Table 4.15: Given is the averaged cpu time in seconds on 100 symmetric (i.e. $m = n$) problem instances of the best TPS combinations compared to CPLX of the classes SOLGEN (top) and UNIFORM (bottom). The number of supplies and demands m and n are given in thousands. A log-log plot is provided in Figure 4.13. The corresponding regression parameters are given in Table 4.16.

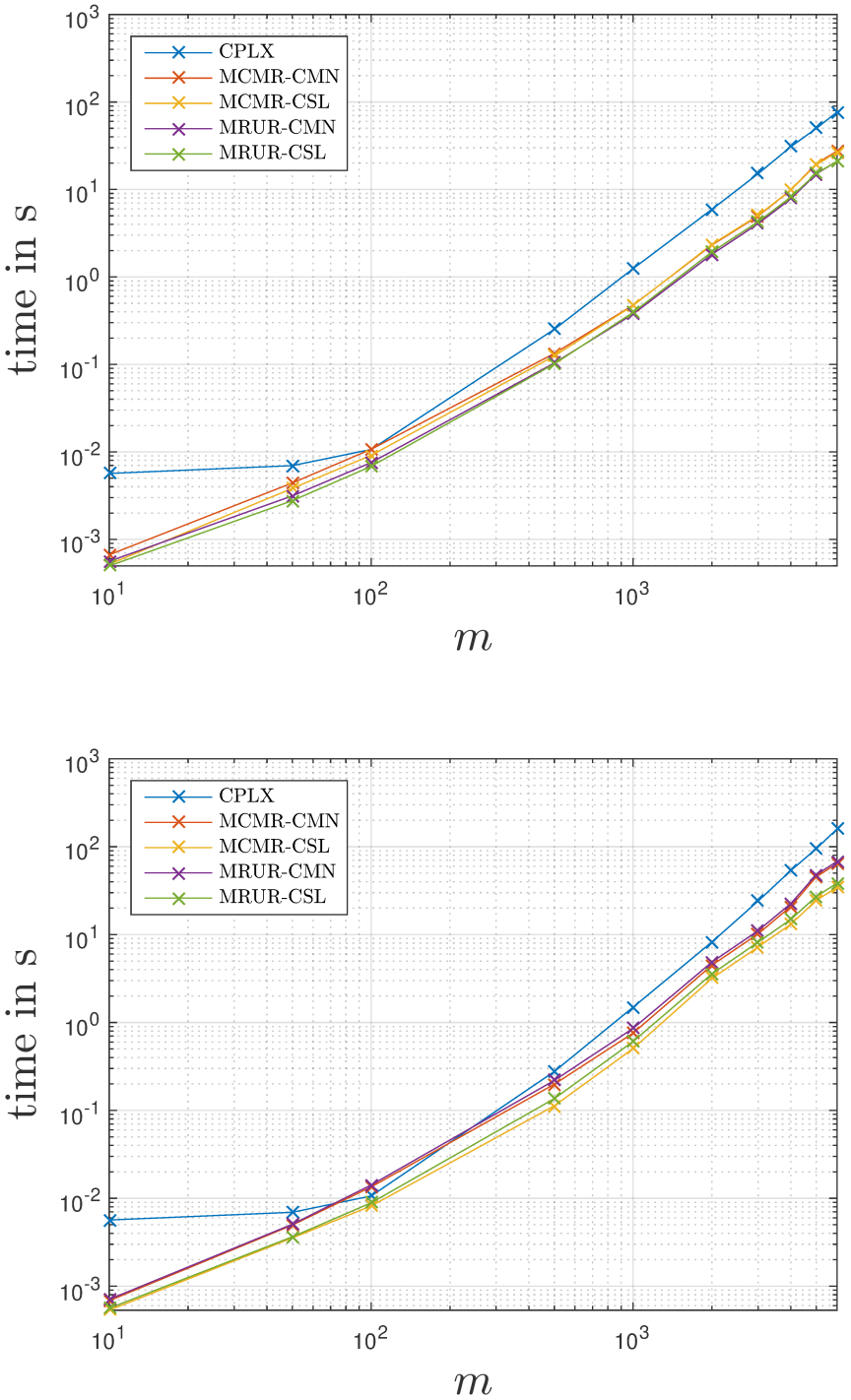


Figure 4.13: Displayed is a log-log plot corresponding to the cpu time of Table 4.15 for the classes SOLGEN (top) and UNIFORM (bottom).

Empirical Complexity		CPLX	MCMR CMN	MCMR CSL	MRUR CMN	MRUR CSL
SOLGEN	α	**2.30**	**2.16**	**2.17**	**2.16**	**2.17**
	$k \cdot 10^6$	0.15	0.17	0.15	0.14	0.13
UNIFORM	α	**2.56**	**2.36**	**2.33**	**2.33**	**2.29**
	$k \cdot 10^6$	0.03	0.07	0.06	0.10	0.09

Table 4.16: Assuming a relation $t = k \cdot m^\alpha$ between the cpu time t in seconds and the problem size $m = n$, we depict the parameters k and α. Omitting all values $m \leq 100$, these parameters were obtained by a linear regression on $\log(t) = \alpha \cdot \log(m) + \log(k)$. All p-values were less than $0.3 \cdot 10^{-6}$.

With regard to memory requirements, we therefore come to the following findings: As shown before, the largest DOTMARK (and overall) class solved was of dimension $m = 1.6384 \cdot 10^4$ where the most difficult class LogGRF was solved in 600 seconds on average. In terms of the product[11] $m \cdot n$, this is seven times larger than the largest instance ($m = 6 \cdot 10^3$) solved by CPLX.

SOLGEN										
m / 10^3	1	2	3	4	5	6	8	10	12	15
MRUR-CSL	0.39	2	4	8	15	21	35	58	81	123

Table 4.17: Displayed is the averaged cpu time in seconds of MRUR/CSL on symmetric (i.e. $m = n$) problem instances of the SOLGEN class. To obtain the according values, we solved 100 instances for $m \leq 6 \cdot 10^3$ and 10 instances for $m > 6 \cdot 10^3$. The number of supplies m and demands n are given in thousands.

[11] The memory requirement is dominated by the cost matrix $C \in \mathbb{R}^{m \times n}$ and is therefore $\mathcal{O}(mn)$, as shown in Section 4.3.

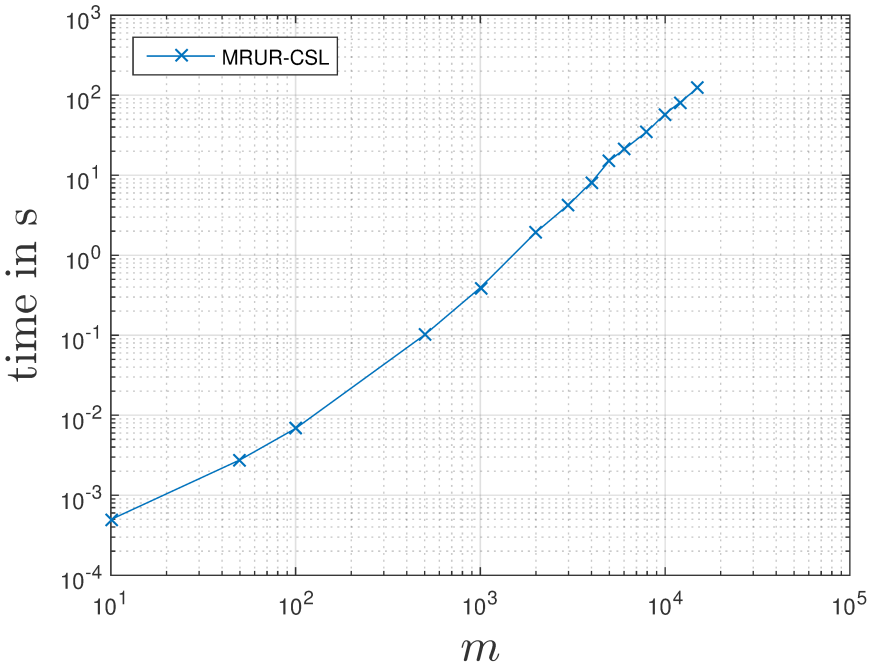


Figure 4.14: Displayed is a log-log plot for the cpu time of Table 4.17. The corresponding regression parameters $\alpha = 2.13$ and $k = 1.7 \cdot 10^{-7}$ were obtained as described in Table 4.16.

4.6 Conclusion

In summary, we have clearly succeeded in developing an efficient implementation of the TPS in MATLAB. Regarding the starting heuristics and pivot rules evaluated in the context of this implementation, the MRUR, MRMR and MCMR represent a universally good choice and in combination with CMN, RMN or especially the Shortlist rules provide the best results for the covered problem classes. In a detailed analysis of various problem classes, the cpu time of the TPS was competitive to a commercial Network Simplex of CPLEX implemented in C/C++. This solver is referred to as CPLX in the context of this dissertation. Furthermore, these results were achieved with a significantly lower memory load compared to the commercial software. The largest instances solved by the TPS were seven times larger than those of CPLX in terms of the cost matrix. Thus we have created a suitable basis for our investigations in the following chapter.

In the *asymmetric* case, we only performed basic tests where the performance of the TPS was still comparable to CPLX on medium-sized problem instances. A detailed analysis of the asymmetric case was beyond the scope of this work and will be left to further research.

On medium-sized *symmetric* problem instances of dimension $m, n \approx 10^3$, our solver was superior for all problem classes, where the cpu time of CPLX ranged from 150% to 400% of the time spend by the TPS. The scaling properties of the algorithm were studied exclusively for the symmetric case. A regression analysis was performed on the artificial problems: Assuming a relation $t = k \cdot m^\alpha$ between the cpu time t and the symmetric problem size $m = n$, we obtained exponents α in the interval $[2.13, 2.17]$ for the TPS compared to 2.3 for CPLX for the SOLGEN class. Similarly, the regression analysis yielded exponents in the interval $[2.29, 2.36]$ for the TPS and 2.56 for CPLX on the UNIFORM instances. Interestingly, these results undercut the general theoretical complexity of Section 2.2. Here, the best complexity of a strictly polynomial algorithm was $\mathcal{O}(m^3 \log m)$. The results for artificial problems, however, are in some contrast to the results for the non-artificial classes. While the TPS was superior on

medium-sized DOTMARK problems, it was slower than CPLX on 6 out of 10 non-artificial classes for dimension $m, n \approx 4 \cdot 10^3$. On the other hand, the TPS was able to solve instances of size $m, n \approx 1.5 \cdot 10^4$ while CPLX could only solve instances where $m, n \leq 6 \cdot 10^3$ due to memory limitations.

While we do not have access to implementation details for CPLX, it can be assumed that Phase II of the algorithm, i.e. the execution of the pivot steps, is implemented very efficiently. Accordingly, we conclude that the advantages in computation time of the TPS are indeed caused by the problem-specific modifications, that is, the starting heuristics in Phase I and the pivot rule. These advantages are clearly dependent on the type of transportation problem and are outweighed by the more efficient implementation of pivot steps in CPLX as the number of pivot steps grows. Hence, assuming a trade-off between computing time and memory requirements, the comparison may be even more in favor of the TPS if the same implementation of the pivot steps is used. A clear advantage of the commercial solver CPLX is that it has proven to be much more robust on the DOTMARK instances. One reason for the higher variance in cpu time of the TPS is certainly that the special pivot rules and start heuristics of the TPS considerably reduce the computing time on average, but depend strongly on the special structure of individual instances and are therefore more prone to outliers. Since CPLX methods for preprocessing and solving general network flow problems have been developed, it seems reasonable to assume that they make less use of specific structures in transport problems, which aligns the total cpu time across different problems.

We further investigated the influence of the starting heuristics by a comparison for medium-sized problems, which was performed to quantify the benefit of using specialized starting heuristics instead of a common Phase I of the Network Simplex. The observed difference between the best and worst start heuristics was between 25% and 50% of the total computation time.

4.7 Outlook

We close this chapter by listing open questions for further research: While our MATLAB-implementation indicates algorithmic advantages of our approach, these advantages could be better quantified by implementing the algorithm in a low-level programming language, instead of interfacing CPLEX into MATLAB. Furthermore, as discussed in minor observation 2, the LALC may constitute an effective starting heuristic when implemented in a different programming language.

As briefly mentioned, while the TPS was competitive for symmetric problem instances, our shot analysis of the asymmetric case was less promising. Whether this phenomenon is due to the specific implementation of the algorithm or the inherent structure of the problem remains open. Moreover, with regard to asymmetric problem instances, our attempt to provide an efficient pivot rule by the Block Search method has failed.

Additionally, to reduce the scope of this chapter and increase the comparability of our results, we refrained from optimizing the parameters of the Shortlist approach for general problem instances and instead used the configuration proposed by Schrieber et al. (2017) for transportation problems created for computational image comparison. A re-optimization of these parameters for general transportation problems may lead to further improvement of the average computation time.

Lastly, we would like to point out another solution approach, which we have investigated in our numerical studies but not included in this doctoral thesis. Motivated by the work of Shor et al. (1985), we observed promising results of bundle-methods applied to a non-smooth objective function which is derived from the following dual reformulation:

$$\begin{aligned} \max_{\lambda \in \mathbb{R}^n} \Theta(\lambda) \quad := \quad & \min_{x \in \mathbb{R}^{mn}} \quad \langle c, x \rangle - \langle \lambda, Bx - b \rangle \\ & \text{s.t.} \quad Ax \;=\; a \\ & \qquad\quad x \;\geq\; 0. \end{aligned}$$

The inner problem (constrained minimization) can be solved analytically to

obtain an unconstrained convex minimization problem with piecewise linear objective function $-\Theta(\lambda)$. Taken into account the recent development of non-smooth optimization methods with the potential exploitation of parallel computation, this approach could be a valuable alternative for high degrees of asymmetry, i.e. $m \gg n$. In fact, we observed promising results using a basic, non-optimized implementation of a bundle method in MATLAB.

5 Column Generation in the Transportation Simplex

In the following chapter we present a new pivot rule for the TPS, which represents a Column Generation approach to the transportation problem. Our method builds upon the Shortlist Rule of Gottschlich and Schuhmacher (2014), see Section 4.3.4, and the TPS implementation presented in Chapter 4. As we will explain in Section 5.2.1, this method is developed specifically for symmetric problems, since it benefits from the large ratio of the number of variables versus constraints of these problems. Accordingly, our numerical analysis will be limited to the symmetric case. Nevertheless, the method is also applicable in the asymmetric case.

The initial motivation for this approach stems from our numerical studies in Section 4.5 and was already indicated in major observation 5 of Section 4.5.1. In summary, the Shortlist pivot rules excelled on the UNIFORM and DOTMARK classes and produced comparable results to the other pivot rules on the SOLGEN instances. Considering the basic symmetric evaluation in Section 4.5.1, this observation is easily explained by the percentage of pivot steps performed on the shortlists. On average these are 100% for the UNIFORM instances, 54% to 63% for the DOTMARK instances and up to 30% for the SOLGEN instances, see Tables 4.4, 4.6 and 4.8. If this share is large, the dimension of the problem is effectively reduced for the majority of the solution process. As observed in Section 4.5.1, the number of total pivot steps correspondingly becomes significantly smaller. In the extreme case of the UNIFORM class, the total number of pivots is halved, which is clearly reflected in total cpu time. Thus, while the Shortlist Rule is generally successful, it could

possibly be improved for DOTMARK and SOLGEN instances.

Clearly, a straightforward approach to achieve this would be to readjust the parameters of the Shortlist Rule, e.g. increase the length l_{SL} of the shortlists. However, this would limit the transferability of our results to further problem classes, as these parameters depend on the problem instances studied. Instead, we strive for a method that dynamically adapts to given problem instances. In addition, using the same parameters ensures the comparability to the results of Gottschlich and Schuhmacher (2014) and Schrieber et al. (2017).

Therefore, we want to investigate an approach that is less dependent on the parameters of the original Shortlist Rule and, even more importantly, ensures that the performance on the UNIFORM class is at least as good as that of the original version. With DOTMARK and especially SOLGEN instances, a large portion of the pivot steps have to be executed on the full set of variables before the optimal solution is reached. This implies that optimal solutions of these instances tend to contain variables with comparably high costs, which are accordingly excluded from the shortlists. To improve the performance of the Shortlist Rule for such problems while maintaining the excellent results for the UNIFORM instances, we propose dynamic extension of the shortlists. As we will see in Section 5.2.1, this approach represents a very natural extension of the implementation of the TPS proposed in Section 4.3. To the best of our knowledge, it has not yet been applied to the transportation problem and evaluated numerically.

5.1 Enhancement of the Shortlist Rule

In the following we describe an enhancement of the Shortlist Rule presented in Section 4.3.4. As for the original method, we only give the detailed description for the row variant without loss of generality. We begin by introducing an extension procedure for shortlists in Section 5.1.1 and state the new pivot rules in Section 5.1.2. Before we proceed, some nomenclature and notation: Hereinafter the term Shortlist Rules includes the RSL and CSL as well as all

possible extensions of these rules or, in other words, any method that operates on a shortlist before it ultimately invokes a pivot rule on the full set of variables. If we want to refer explicitly to the RSL and CSL, we use the term original Shortlist Rules. Moreover, as the length of the shortlists will not be constant anymore, we hereinafter denote the current length of the shortlist $L_i \subseteq J$, corresponding to the i-th row of C, by $l_i \leq n$.

The basic idea of the approach is to (repeatedly) increase the size l_i of the shortlists whenever an optimal solution for the shortlists has been obtained. Furthermore, before such an extension is executed, we verify whether the current solution is optimal on the full set of variables. This implies in particular that in the event that an optimal solution is already included in the initial shortlists, the enhanced approach is equivalent to the RSL and therefore maintains the excellent performance of this rule for such problems.

Furthermore, we need criteria for the selection of the elements to be added. Clearly, a straightforward approach is to select further variables with low costs. However, in terms of costs, the best variables are already included in the initial selection of the shortlists. Therefore, we propose to further select variables based on the feasibility of the current dual multipliers, i.e. the reduced costs. In this way, variables can be added that have high costs but still assume positive values in an optimal solution.

5.1.1 Extension procedure

All methods have in common that first the original Shortlist Rule is executed, i.e. an optimal solution is determined on the shortlists of initial lengths $l_i = l_{SL}$ for all i. As we describe the extension procedures in detail, we assume that such a primal-dual optimal solution (x, u, v) was determined for a given transportation problem (C, a, b). The extension procedure is then divided into three successive steps:

Optimality check: Initially, the complete reduced cost matrix C^{uv} with respect to the optimal dual solution is computed. In the event that no candidates

with negative reduced costs are found, i.e. $C^{uv} \geq 0$, the solution is also optimal for the full set of variables and the TPS stops. In the contrary case, the candidate (i_e, j_e) with the most negative reduced costs, i.e.

$$(i_e, j_e) = \underset{(i,j)\in I\times J}{\arg\min}\, c_{ij}^{uv},$$

is selected as the entering variable and j_e is added to the corresponding shortlist L_{i_e}. As mentioned above, this ensures the equivalence to RSL, in case an optimal solution was already included in the initial shortlists L_i. Furthermore, this approach provides an entry variable with maximum negative reduced costs in the contrary case.

In the non-optimal case, the shortlists are further extended by means of the following two procedures:

Extension procedure 1: For the first procedure, we naturally extend the methodology of the Shortlist pivot rule: For each shortlist L_i, we add the Δc elements $j \in J$ with the lowest corresponding costs in the row i of C which are not yet included in the shortlist. More precisely, for all $i \in I$, assume σ_i to be a bijective mapping

$$\begin{aligned} \sigma_i : \quad & \{1, \ldots, n\} \to J \\ & k \mapsto j \end{aligned}$$

such that it holds $c_{i\sigma_i(1)} \leq c_{i\sigma_i(2)} \leq \cdots \leq c_{i\sigma_i(n)}$ where, if necessary, ties are split arbitrarily. Then, for all $i \in I$, we add the first Δc elements in

$$\left(\sigma_i^{-1}(1), \ldots, \sigma_i^{-1}(n)\right)$$

which are not yet contained in L_i. In the case that less than Δc elements are eligible, we add the maximum possible number. However, for our choice of parameters, see Section 5.3, this was only the case for small $n \leq 200$.

Extension procedure 2: To complement the selection of variables with low cost, we introduce a method based on the reduced costs. Here, we proceed as follows: For each $i \in I$, we consider the elements of the i-th row of the reduced cost matrix C^{uv} in the order of the bijective mapping σ_i introduced above and

add the first Δuv negative elements. Accordingly, we successively compute the elements of the vector

$$\left(c^{uv}_{i\sigma_i(1)}, \ldots, c^{uv}_{i\sigma_i(n)}\right)$$

until either Δuv elements $c^{uv}_{i\sigma_i(k)} := c_{i\sigma_i(k)} - u_i + v_{\sigma_i(k)} < 0, k \in J$, were found or the vector was completely generated. In the event that less than Δuv negative elements were found, we add the maximum possible number. Hereby, observe that for all $i \in I$, (u, v) is feasible with respect to all $j \in L_i$ and, hence, it holds $c^{uv}_{ij} \geq 0$ for these elements. Consequently, $c^{uv}_{i\sigma_i(k)} < 0$ implies $\sigma_i(k) \notin L_i$, which ensures that elements are not added to the shortlists more than once.

A clear advantage of this method is that it uses the information (in the form of dual variables) obtained in the initial solution process and is not exclusively based on low cost. The hope is to integrate variables into the shortlists that have comparatively high costs but are still included in optimal solutions and thereby complement the initial composition of the shortlists.

Finally, we would like to point out that we have investigated two variants of this procedure. First, we considered the elements in the natural order, i.e. for a given row i, we successively computed the elements of the i-th row of C^{uv}. Second, we tried sorting this row in ascending order, i.e. adding the elements with the most negative reduced costs. However, somewhat surprisingly, the first approach provided the better numerical results.

5.1.2 Enhanced Shortlist Rules

By virtue of the previous explanations, we finally state the new pivot rules. As before, we restrict the detailed description to the row variant of the rule:

Enhanced Row Shortlist Rule (ERSL) Until an initial solution on the shortlists has been found, the rule is equivalent to the RSL. In particular,

it uses the same parameter configuration, i.e.

$$l_{SL} := \begin{cases} \min\{15, n\} & \text{for } n \leq 200, \\ 15 + \lfloor 15 \cdot \log_2(n/200) \rfloor & \text{for } n > 200, \end{cases}$$

$$k_{SL} := l_{SL},$$

$$p_{SL} := 5\%,$$

and initially $l_i := l_{SL}$ for all $i \in I$. In addition, the parameter e_{SL} is introduced, which specifies the maximum number of extensions to the shortlists, as well as the parameters Δc and Δuv, which limit the number of elements that are added per shortlist in one extension.

Whenever an optimal solution (x, u, v) was computed for the shortlists, and less than e_{SL} extensions have been performed, the extension procedure described in Section 5.1.1 is applied. In the opposite case, the rule is stopped and – analogous to the original Shortlist Rules – a second pivot rule is invoked over the full set of variables to ultimately solve the problem.

Lastly, observe that the Shortlist Rules and in particular the Enhanced Shortlist Rules do not significantly increase the memory utilization of the TPS. Although additional storage space is required for the corresponding lists, this is at most the order of magnitude of the cost matrix (which dominates the memory load) and on average significantly smaller.

Enhanced Column Shortlist Rule (ECSL) As in the case of the pivot rules of Section 4.3.4, we have also implemented the corresponding column variant of the Enhanced Shortlist Rule.

5.2 Column Generation

In the following section, we illustrate that the TPS implemented with the Shortlist Rules (in particular the ERSL or ECSL) uses concepts similar to Col-

umn Generation. To this end, we briefly discuss a generic Column Generation approach and then illustrate the adaptation to the transportation problem.

5.2.1 Generic Column Generation for linear programs in standard form

Consider a generic Column Generation method, as described in more detail in Section 6.2. Naturally, this approach can be equivalently applied to linear programs in standard form, i.e. a primal problem

$$\begin{aligned} v(K) := \quad & \min_{x \in \mathbb{R}^{|K|}} \quad \sum_{k \in K} c_k x_k \\ & \text{s.t.} \quad \sum_{k \in K} \mathbf{d}_k x_k \;=\; \mathbf{b} \\ & \qquad\qquad x_k \;\geq\; 0 \qquad \forall\, k \in K \end{aligned} \tag{P_K}$$

with $\mathbf{b} \in \mathbb{R}^p$ and $\mathbf{d}_k \in \mathbb{R}^p$ for all $k \in K$ and the corresponding dual problem

$$\begin{aligned} v(K) = \quad & \max_{y \in \mathbb{R}^p} \quad \mathbf{b}^\top \mathbf{y} \\ & \text{s.t.} \quad \langle \mathbf{d}_k, \mathbf{y} \rangle \;\leq\; c_k \qquad \forall\, k \in K \\ & \qquad\qquad \mathbf{y} \;\leq\; 0. \end{aligned} \tag{D_K}$$

For the readers convenience, we summarize the Column Generation approach for the standard form in the following: To avoid solving (P_K) for the complete set K, one solves restricted problems (P_{K_l}) for a finite sequence of subsets $K_0 \subsetneq K_1 \subsetneq \ldots \subsetneq K_L \subseteq K$. For these restricted problems one applies an LP solver providing a primal-dual optimal pair such that the primal and dual solution satisfy complementary slackness conditions (1.4). The most common variant is the Simplex. For each computed optimal solution of a restricted problem (P_{K_l}), one either verifies that it is already optimal for (P_K) or adds additional variables to K_l to get K_{l+1} and proceeds with the next restricted problem$(P_{K_{l+1}})$. More precisely, assuming $\mathbf{x}^*_{K_l}$ is an optimal solution of (P_{K_l}) and $\mathbf{y}^*_{K_l}$ the corresponding optimal dual solution, one computes $h_k := \langle \mathbf{d}_k, \mathbf{y}^*_{K_l} \rangle - c_k$ for all k in K. If $h := (h_1, \ldots h_{|K|})^\top \leq 0$, $\mathbf{y}^*_{K_l}$ is feasible for (D_K) which implies that $\mathbf{x}^*_{K_l}$ and $\mathbf{y}^*_{K_l}$ are optimal for (P_K) and (D_K), see

Section 1.3. In the other case, all k with $h_k > 0$ qualify as entering indices for K_{l+1} and $(\mathbf{x}^*_{K_l}, \mathbf{y}^*_{K_l})$ is used as a starting point for the problem $(P_{K_{l+1}})$. Observe that it necessary to choose K_0 such that (P_{K_0}) is feasible.

5.2.2 Application to the Transportation Simplex

Generally, the Column Generation approach is applied when the number of variables $|K|$ is much larger than the number of constraints p, e.g. $|K|$ is exponential in p. In case of the transportation problem, we have $p = m + n$ and $|K| = m \cdot n$, i.e. $|K| = (p/2)^2$ for $m = n$. For asymmetric problems the ratio between variables and constraints becomes even smaller up to the point where $p > |K|$ for extreme degrees of asymmetry. This led us to propose our approach solely for the symmetric case and make one adjustment in comparison to the generic Column Generation. Instead of computing a single entering variable via a subproblem in each iteration, we explicitly consider all dual constraints and allow $|K_{l+1} \setminus K_l| > 1$. Hence, compared to the generic method, we perform few[1] but expensive extensions which add multiple variables per iteration.

Including this adjustment, the TPS follows exactly the approach described above when the set of potential entering variables in the pivot rule is restricted. Adding an index k to the restricted problem generally requires the *generation* of the corresponding column $\mathbf{d}_k$, which means that the column is added to the representation of the constraint matrix used by the LP solver. As discussed in Section 4.3, the TPS uses the inherent graph structure of the transportation problem to perform efficient pivot steps. Thus, the constraint matrix is never explicitly generated, but the columns are handled as edges in a graph. The only stored information besides costs, supplies and demands is which edges, i.e. index tuples[2], are in the current basis and new edges are introduced via the pivot rule. Thus, if the selection of the pivot rule is limited to a subset of

[1] In the extreme case, only one extension.

[2] Recall that the indices of the variables x_{ij} in the transportation problem are given by tuples (i, j) which also define edges in the transportation graph G_{tp}, see Section 1.4.

index tuples, the problem is solved[3] for the corresponding subset of variables[4].

This approach is realized in the Shortlist Rules by restricting the solution process to the shortlists. In the extreme case of the original Shortlist Rules, a two-step Column Generation is performed, in which the initial quantity K_0 defined by the shortlists is directly extended to $K = I \times J$ after an optimal solution on the shortlists has been found. For the Enhanced Shortlist Rules we perform $e_{SL} + 1$ Column Generation steps where the sets K_l, $l = 0, \ldots, e_{SL}$, are gradually increased by means of the extension procedure of Section 5.1.1.

5.3 Numerical analysis

To evaluate the performance of the Enhanced Shortlist Rules we solved a series of test problems. As explained in Sections 4.5.3 and 5.2.1, we limit our investigation to symmetric problems, i.e. $m = n$ and the column variant ECSL. Furthermore, since the MRUR was the best heuristic for the incorporated problem classes, we only evaluate the performance of MRUR/ECSL.

The enhanced rules are equivalent to the original Shortlist Rules for the UNIFORM instances, since an optimal solution is already included in the initial shortlists (cf. major observation 5 in Section 4.5.1) and no extension of the lists is necessary. Therefore, we only evaluated SOLGEN and DOTMARK instances and refer to the results of Section 4.5 for the UNIFORM class.

Our analysis is again divided into two parts. First, we examine a larger parameter set for problems of the dimension $m = 32^2 = 1024$. Then, based on the results of this first experiment, we limit the possible parameter configurations when we investigate larger problems of dimension $m = 64^2 = 4096$ and $m = 128^2 = 16384$[5].

[3] We assume that the subset of variables defines a feasible problem.

[4] We assume that the index tuples of the initial basis computed in Phase I are included in this subset. Otherwise we consider without loss of generality the union of these sets.

[5] The selection of problem dimensions is based on the available DOTMARK problems.

5.3.1 Basic analysis

In our first experiment we solve 1000 instances of the SOLGEN class as well as all 450 instances of DOTMARK for $m = 1024$, i.e. resolution 32×32 in case of the DOTMARK problems. We tested the following parameter configurations:

$$\begin{aligned} \Delta c &\in \{0, \tfrac{l_{SL}}{2}, l_{SL}\}, \\ \Delta uv &\in \{0, \tfrac{l_{SL}}{2}, l_{SL}\}, \\ e_{SL} &\in \{1, 2, 3\}. \end{aligned}$$

The results are presented in Tables 5.1, 5.2, 5.3, and 5.4. In order to evaluate the enhanced rule, the upper left corner of each tables represents the results of the original Shortlist Rule presented in the prior chapter, see Section 4.3.4.s

The enhanced rules perform well on the DOTMARK instances, where it improves the average computation time of the hitherto best combination MRUR/CSL by more than 11%. This was achieved by the parameter configuration $\Delta c = 0$, $\Delta uv = l_{SL}$ and $e_{SL} = 1$. Both extension procedures succeeded in decreasing the total number of pivot steps by increasing the percentage of pivot steps on the shortlists. However, setting $\Delta c > 0$ had only a small influence on the total number of pivots, so that this advantage was nullified by the cpu time required for the extensions. On the other hand, increasing the parameter Δuv let to significantly smaller total cpu time on DOTMARK instances, see Tables 5.1, 5.2 and 5.3. In order to be able to make a more detailed statement about the performance on the individual DOTMARK classes, we further add Table 5.4. Here it can be observed that the extended approach reduces the cpu time for all classes except WhiteNoise and GRFrough and achieves comparable results on these two classes.

As mentioned before, executing the extensions introduces a small computational overhead to the approach. This overhead also explains the fact that the best cpu time is achieved for $\Delta c = 0$ and $\Delta uv = 1$, although the combination $\Delta c = 1$ and $\Delta uv = 1$ achieves the smallest number of pivot steps.

In contrast, the enhanced rules failed to decrease the cpu time of the MRUR/CSL on the SOLGEN instances. The reason for this can be observed in

Table 5.2, where we documented the pivot steps. In particular, increasing Δuv resulted in a significantly larger share of shortlists pivots as well as a significant reduction of the total pivots on the DOTMARK instances. In comparison, the increase of Δc and Δuv only slightly raised the percentage of pivots on the shortlists and even increased the total number of pivots for the SOLGEN class. This implies that even with our extension methods, we fail to include critical variables. As a result, optimization on the shortlists progresses even slower toward the optimal solution than the pivot rule on the entire set of variables. Lastly, observe that an investigation of $e_{SL} > 3$ did not produce better results for both problem classes, as is already indicated by the fact that the cpu time for $e_{SL} = 1$ is the best in all cases. Likewise, our numerical investigations have shown that the results can not be improved by setting Δc and Δuv greater than l_{SL}.

5.3.2 Large problems

Since the results of the ECSL on larger SOLGEN problems were identical to the first experiment, i.e. delivered inferior results with regard to cpu time, we exclude this class from further analysis. Furthermore, based on the results of the first experiment, we fix the number of extension steps to $e_{SL} = 1$, that is, we restrict the parameter configurations to

$$\begin{aligned} \Delta c &\in \{0, \tfrac{l_{SL}}{2}, l_{SL}\}, \\ \Delta uv &\in \{0, \tfrac{l_{SL}}{2}, l_{SL}\}, \\ e_{SL} &= 1, \end{aligned}$$

for $m = 4096$. We present the results over all 450 DOTMARK instances in Tables 5.5 and 5.6. It is observed that the extended approach is even more effective for this larger dimension; it beats the original Shortlist Rule on all subclasses and reduces the average computation time by 50%. Moreover, it closes the gap to CPLX as can be observed in Figure 5.1.

Finally, to reduce the computational burden, we fix $\Delta c = 0$ and $e_{SL} = 1$, for

	SOLGEN			DOTMARK		
$e_{SL}=1$	$\Delta c \backslash \Delta uv$	0	0.5	1	0	0.5	1
	0	**0.43**	0.47	0.47	0.97	0.87	**0.86**
	0.5	0.45	0.47	0.48	1.01	0.88	0.87
	1	0.45	0.48	0.48	1.01	0.90	0.89
$e_{SL}=2$	$\Delta c \backslash \Delta uv$	0	0.5	1	0	0.5	1
	0	**0.43**	0.51	0.53	0.97	**0.89**	**0.89**
	0.5	0.47	0.52	0.54	1.06	0.93	0.93
	1	0.49	0.53	0.55	1.08	0.97	0.97
$e_{SL}=3$	$\Delta c \backslash \Delta uv$	0	0.5	1	0	0.5	1
	0	**0.43**	0.57	0.59	0.97	**0.92**	**0.92**
	0.5	0.51	0.59	0.61	1.13	0.98	0.97
	10	0.53	0.61	0.63	1.18	1.03	1.01

Table 5.1: Average cpu time (seconds) for dimension $m = 1024$ of MRUR/ECSL on 1000 symmetric SOLGEN instances (left) and all 450 symmetric DOTMARK instances (right). Each row of tables represents one value of $e_{SL} \in \{1, 2, 3\}$. Regarding the small tables, the values of Δc (rows) and Δuv (columns) are given as multiples of l_{SL}. Accordingly, the upper left cell of each table represents the result, when no extension is executed, i.e. the result of the original MRUR/CSL which was investigated in the prior Chapter, see Section 4.3.4. While the enhanced approach fails for the SOLGEN instances, we observe that they are superior to the original Shortlist Rules on the DOTMARK instances, when the extension method based on the dual variables, i.e. $\Delta uv > 0$ is used. In contrast, the total cpu time increases if we set a $\Delta c > 0$.

SOLGEN

$e_{SL} = 1$

$\Delta c \backslash \Delta uv$	0		0.5		1	
0	**5.09**	34%	5.20	49%	5.22	56%
0.5	5.11	38%	5.20	52%	5.23	58%
1	5.12	42%	5.20	54%	5.22	60%

$e_{SL} = 2$

$\Delta c \backslash \Delta uv$	0		0.5		1	
0	**5.09**	34%	5.32	63%	5.34	74%
0.5	5.12	42%	5.31	68%	5.35	77%
1	5.16	48%	5.30	72%	5.34	80%

$e_{SL} = 3$

$\Delta c \backslash \Delta uv$	0		0.5		1	
0	**5.09**	33%	5.41	73%	5.45	85%
0.5	5.14	44%	5.43	79%	5.45	89%
1	5.18	52%	5.39	83%	5.41	91%

Table 5.2: We present the pivot steps corresponding to the cpu time in Table 5.1 for the SOLGEN class. In each cell, the total number of pivots steps is given in thousands together with the percentage of pivot steps performed on the shortlists. While all extension procedures succeed in increasing the percentage of pivot steps on the shortlists, the number of overall pivots is also slightly higher for all parameter configurations. Thus, the extension approach failed for the SOLGEN instances.

DOTMARK

$e_{SL} = 1$

$\Delta c \backslash \Delta uv$	0		0.5		1	
0	14.24	54%	10.66	83%	10.21	86%
0.5	13.55	66%	10.39	87%	9.90	90%
1	12.79	73%	10.15	90%	**9.70**	92%

$e_{SL} = 2$

$\Delta c \backslash \Delta uv$	0		0.5		1	
0	14.24	54%	9.97	96%	9.73	98%
0.5	13.16	74%	9.84	97%	9.59	99%
1	12.12	84%	9.71	98%	**9.47**	99%

$e_{SL} = 3$

$\Delta c \backslash \Delta uv$	0		0.5		1	
0	14.24	54%	9.94	99%	9.75	100%
0.5	12.93	80%	9.82	99%	9.60	100%
1	11.81	90%	9.69	100%	**9.48**	100%

Table 5.3: We present the pivot steps corresponding to the cpu time in Table 5.1 for the DOTMARK class. In each cell, the total number of pivots steps is given in thousands together with the percentage of pivot steps performed on the shortlists. All extension procedures succeed in increasing the percentage of pivot steps on the shortlists and decrease the total number of pivot steps. However, increasing the parameter Δc has only a mild effect on the total number of pivot steps. Thereby, the advantage gained by reducing the pivot steps is outweighed by the overhead in cpu time for executing the extensions. In contrast for $\Delta uv > 0$, the percentage of pivot steps on the shortlists increases enough to significantly reduce the total number of pivot steps and thereby speed up the method.

Δuv	0	0.5	1
WhiteNoise	0.55	0.56	0.56
GRFrough	0.58	0.59	0.59
GRFmoderate	0.91	0.85	0.86
GRFsmooth	1.28	1.11	1.12
LogGRF	1.67	1.45	1.40
LogitGRF	1.32	1.12	1.10
CauchyDensity	1.37	1.22	1.19
Shapes	0.43	0.36	0.32
ClassicImages	0.85	0.81	0.81
MicroscopyImages	0.89	0.71	0.70

Table 5.4: We differentiate the cpu time of Table 5.1 for the DOTMARK classes and $m = 1024$: The parameter Δuv is given analogously to the tables above as a multiple of l_{SL} while $\Delta c = 0$ and $e_{SL} = 1$ are fixed, that is, the first column represents the results of MRUR/CSL. It is observed that the enhanced approach of the Shortlist Rule MRUR/ECSL reduces the computation for all but the first two classes, where it still provides comparable results to the original approach MRUR/CSL.

cpu time

$\Delta c \backslash \Delta uv$	0	0.5	1
0	70.40	39.90	**35.23**
0.5	67.22	40.37	35.80
1	64.13	41.44	37.08

pivot steps

$\Delta c \backslash \Delta uv$	0		0.5		1	
0	210.00	35%	100.47	86%	85.73	94%
0.5	183.81	53%	96.37	89%	83.41	96%
1	164.51	62%	93.89	91%	**82.16**	96%

Table 5.5: Average cpu time in seconds (top) and pivot steps (bottom) of MRUR/ECSL on all 450 symmetric DOTMARK instances of dimension $m = 4096$. The parameters Δc and Δuv are chosen as in the previous tables, while $e_{SL} = 1$ is fixed. Again, the upper left cell of each table represents the results of MRUR/CSL. For this dimension, the ECSL is even more efficient than for $m = 1024$, see Table 5.1. We observe that the cpu time of the original Shortlist Rules can be approximately halved when the enhanced approach is applied. Analogously, we observe that the extension introduces a computational overhead, which is why $\Delta c = 1$ and $\Delta uv = 1$ led to the smallest number of pivots but a slightly higher cpu time than $\Delta c = 0$ and $\Delta uv = 1$.

Δuv	0	0.5	1
WhiteNoise	15.28	15.22	15.17
GRFrough	17.38	17.17	17.18
GRFmoderate	71.69	44.87	43.41
GRFsmooth	120.42	80.69	70.77
LogGRF	160.96	119.86	104.84
LogitGRF	109.43	70.99	61.50
CauchyDensity	100.71	72.93	65.82
Shapes	34.77	27.74	25.00
ClassicImages	63.89	43.36	39.90
MicroscopyImages	11.31	8.36	7.87

Table 5.6: Depicted is the cpu time of Table 5.5 for the individual DOTMARK classes when $m = 4096$: As before, the parameter Δuv is given as a multiple of l_{SL} while $\Delta c = 0$ and $e_{SL} = 1$ are fixed, that is, the first column represents the results of MRUR/CSL. Here, the enhanced approach of the Shortlist Rule (MRUR/ECSL) reduces the cpu time of all classes.

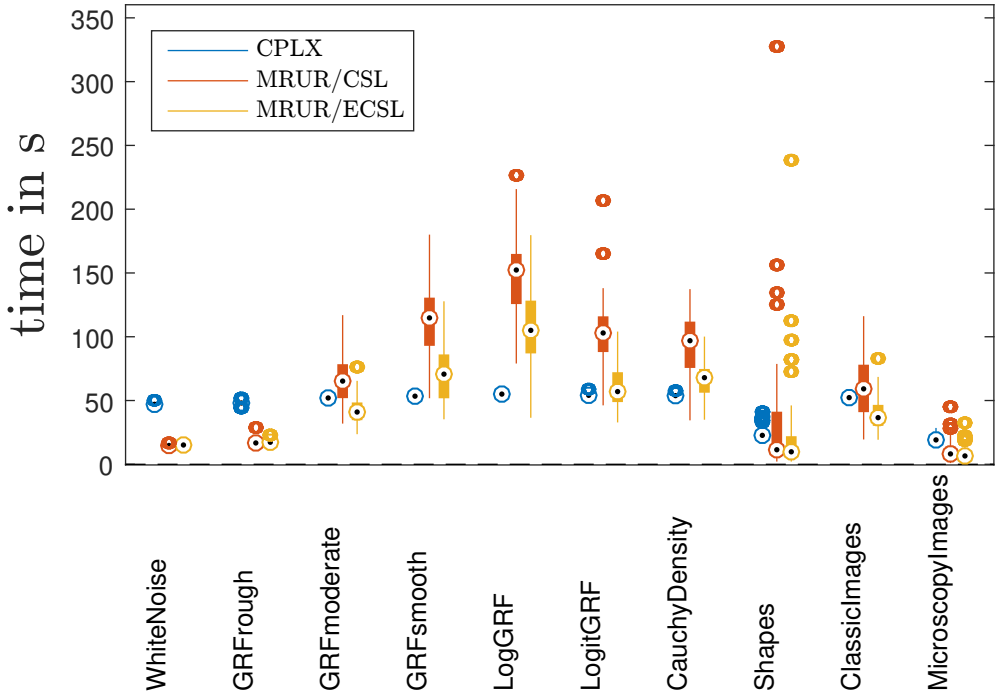


Figure 5.1: Depicted is a boxplot of CPLX, MRUR/CSL and MRUR/ECSL cpu time for DOTMARK instances of dimension $m = 4096$. Here, we observe that the ECSL clearly improves the CSL approach. While the TPS is slower than CPLX on 6 out of 10 classes when the original shortlist approach is used, this number reduces to 3 out of 10 when the results on class LogitGRF are considered a draw. This leaves the three classes GRFsmooth, LogGRF and CauchyDensity, where the enhanced approach is still slower than CPLX regarding the total cpu time. This difference in the average cpu time, however, is caused by only a few instances where the TPS takes really long. These outliers also account for the large variance in cpu time. Accordingly, the enhanced approach is faster than CPLX on most of the instances in these three classes as well.

our largest problem dimension $m = 16384$:

$$\begin{aligned} \Delta c &= 0, \\ \Delta uv &\in \{0, \tfrac{l_{SL}}{2}, l_{SL}\}, \\ e_{SL} &= 1. \end{aligned}$$

We present the results in Tables 5.7 and 5.8 as well as Figure 5.2. Here, CPLX is excluded from our investigation due to memory limitations. Again, we observe that the ESCL approach clearly improves upon the Shortlist Rule. Furthermore, the results indicate that the cpu time could be improved even further by setting $\Delta uv > l_{SL}$ or $e_{SL} > 1$.

cpu time

Δuv	0	0.5	1
	2586	2038	**1855**

pivot steps

Δuv	0		0.5		1	
	4196	22%	2981	39%	**2581**	44%

Table 5.7: Average cpu time in seconds (top) and pivot steps (bottom) of MRUR/ECSL on all 450 symmetric DOTMARK instances of dimension $m = 16384$. The parameters Δc and $e_{SL} = 1$ are fixed. Note that the left cell of each table represents the results of MRUR/CSL. Again, the ECSL clearly improves upon the original Shortlist Rule, although the effect is not as large as for $m = 4096$. The reason for this can be observed in the percentages of pivot steps performed on the shortlists which is 44% at the most. This is a clear indicator, that either Δuv or e_{SL} should be increased for higher dimensions.

5.4 Conclusion and outlook

Using the inherent graph operations of the algorithm, we profitably introduced concepts from Column Generation to the Transportation Simplex. While our approach, based on the Shortlist Rule of Gottschlich and Schuhmacher (2014),

Δuv	0	0.5	1
WhiteNoise	224	229	230
GRFrough	636	443	444
GRFmoderate	2665	1768	1605
GRFsmooth	4563	3572	3112
LogGRF	5362	4743	4411
LogitGRF	4186	3302	2983
CauchyDensity	3119	2553	2352
Shapes	1716	1453	1350
ClassicImages	2511	1603	1436
MicroscopyImages	883	719	633

Table 5.8: Depicted is the cpu time of Table 5.7 for the individual DOTMARK classes when $m = 16384$: As before, the parameter Δuv is given as a multiple of l_{SL} while $\Delta c = 0$ and $e_{SL} = 1$ are fixed, that is, the first column represents the results of MRUR/CSL. As for $m = 4096$, the enhanced approach of the Shortlist Rule (MRUR/ECSL) reduces the cpu time of all classes.

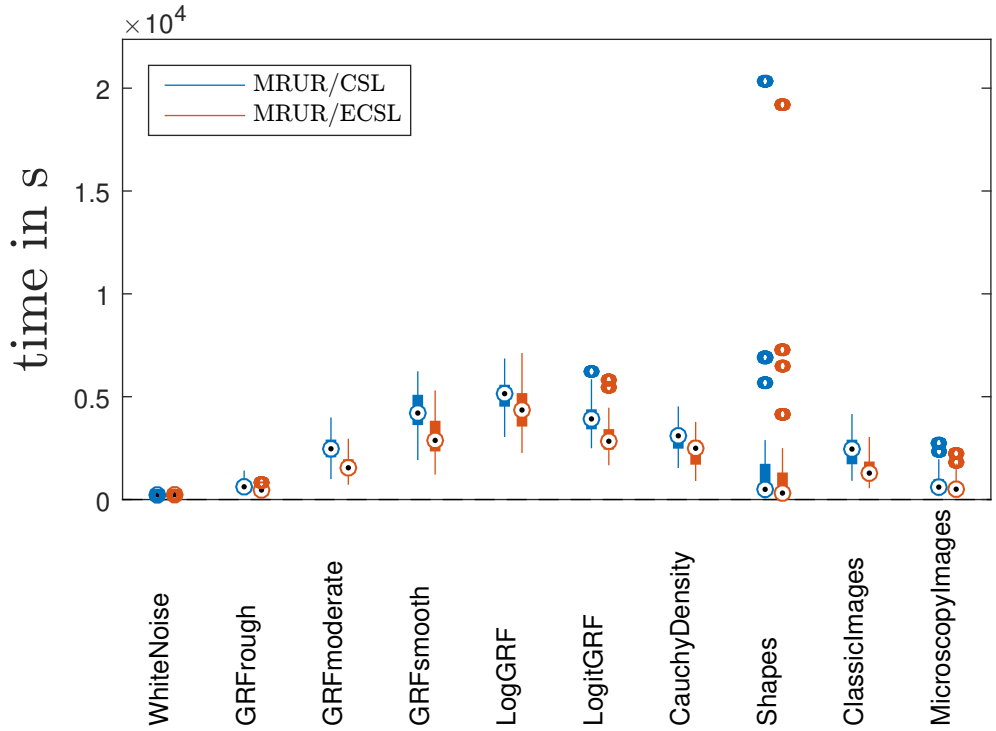


Figure 5.2: Finally, we show a comparison of MRUR/CSL and MRUR/ECSL cpu time for DOTMARK instances of dimension $m = 16384$. CPLX is excluded, since it could not solve problems of this dimension due to memory reasons. Again, we observe that the ECSL clearly improves the CSL approach. In comparison to Figure 5.1, we observe less variance and fewer outliers.

failed to improve the cpu time on artificial problem instances, we observed promising results on DOTMARK, a class of transportation problems that is solved regularly in applications, see e.g. Gottschlich and Schuhmacher (2014) and Schrieber et al. (2017). Here, we reduced the average cpu time by 10% to 50% for the symmetric problem sizes $m = n = 1024$, $m = n = 4096$ and $m = n = 16384$. Thereby, the algorithm compares favorably to a commercial Network Simplex of Cplex (CPLX). While both algorithms provide comparable results with regard to cpu time, the TPS achieves these results with significantly less memory load, see Section 4.5.4.

There are various possible approaches for further improvements: In terms of SOLGEN, we clearly failed to include enough relevant variables into the shortlists to make our method efficient. This is somewhat surprising, since the SOLGEN instances are generally easy to solve, see. e.g. Section 4.5 and Schwinn and Werner (2018). While it is reasonable to assume that the variance in the costs of the variables in the corresponding optimal solutions is higher than, for example, with the UNIFORM instances, a mathematically rigorous explanation of this fact remains an open question.

Another interesting point for further research is the high variance of the Column Generation approach and the TPS in general on the DOTMARK instances. Initial assumptions towards possible explanations have been made, but a thorough analysis remains to be done. In particular, it would be interesting to investigate which instances produce outliers and whether this can be traced back to certain structures in the corresponding transportation problems.

Another approach that should be pursued in the future is the selection of shortlists on the basis of minimum reduced costs instead of minimum actual costs. Especially if one takes into account the good performance of the MRUR heuristic, which implements a similar strategy, on SOLGEN instances, see Sections 3.1 and 4.5.

Apart from these considerations, we have observed shorter cpu time for the SOLGEN instances when the internal pivot rule of the Shortlist Rule was replaced by RMN or CMN, which indicates that altering the selection process

within the Shortlist iterations may be beneficial.

Finally, to reduce the scope of our analysis we refrained from changing the parameter configuration of the original Shortlist Rule. As these were optimized for a fixed length l_{SL} of the shortlists, and specific problem classes of the transportation problem, this may yield further improvement. Regarding the parameters of the enhanced approach, other configurations (e.g. $e_{SL} > 1$, $\Delta c > 0$, $\Delta uv > l_{SL}$) could perform better on very large problems, when the overhead of the extensions is canceled out by the reduction of pivot steps. This is indicated in particular by the results in Table 5.7.

6 Membership testing on the Bernoulli polytope

In the final chapter of this thesis, we apply the Column Generation approach to a problem originating in multivariate statistics. The resulting algorithm was developed within the scope of Krause et al. (2018); accordingly, the following results are based on this work and in particular Sections 6.2, 6.3.1, 6.3.2, 6.3.6, 6.3.7 and 6.5 are taken from Krause et al. (2018), with minor changes. Furthermore, we adopt the numerical results in Section 6.4 with some changes in the text and structure of the presentation in Section 6.4.2.

The corresponding research question is as follows: For a given matrix $B \in \mathbb{R}^{d \times d}$ decide (numerically) whether it is a *Bernoulli matrix* which – as shown in Proposition 6.1.1 – is equivalent to testing if B is a member of the *Bernoulli polytope*. The interest in this problem extends across several communities ranging from probability and operations research to applications in various disciplines; see Krause et al. (2018) for a detailed review of the corresponding publications. In short, the motivation for this work is probably best illustrated by two statements made in the context of probabilistic investigations. The first is found in Embrechts et al. (2016) which end with the statement:

"*Concerning future research, an interesting open question is how one can (theoretically or numerically) determine whether a given arbitrary nonnegative, square matrix is a tail-dependence or Bernoulli-compatible matrix. To the best of our knowledge there are no corresponding algorithms available.*"

Furthermore, with regard to the exponential increase of the parameters in d, it is mentioned in Qaqish (2003) that

"*However, specifying or computing p becomes impractical for n greater than about* 15."[1]

These two statements are theoretically supported by results that originate in the field of operations research where the Bernoulli polytope is known as the *correlation polytope*. In this community, the polytope was introduced by Pitowsky (1991) who also proved that the membership problem is NP-complete. Moreover, Krause et al. (2018) consider an observation that can be traced back to Deza and Laurent (1997) implying that any polynomial algorithm for the membership problem on the Bernoulli polytope would induce a polynomial algorithm for the *binary quadratic problem* (BQP) which is known to be NP-hard, see, e.g., Padberg (1989). Hence, it cannot be expected that there is any method that solves the problem for large d (i.e. $d > 40$, cf. Kochenberger et al. (2014)) in reasonable time on a standard personal computer. However, exploiting the connection to binary quadratic programming in the reverse direction, we aim to develop a method that solves at least medium sized instances (i.e. $20 \leq d \leq 40$) within a few minutes of cpu time. More precisely, both in order to reduce the memory requirement and cpu time for $d \geq 20$, we propose a Column Generation approach where BQPs are solved as subproblems to introduce new variables to the problem.

The rest of this chapter is structured as follows: In the first section, we introduce the problem mathematically and provide an equivalent formulation as a membership problem on the Bernoulli polytope. Furthermore, this problem, in turn, yields a straightforward reformulation as a linear program which constitutes the central problem of this chapter. To solve this LP, we review a generic Column Generation method in Section 6.2, which is then customized to our application in Section 6.3. This includes a proof that the inherent subproblem can be solved as a BQP as well as the introduction of a novel dual bound for Column Generation. Moreover, we exploit specific necessary and sufficient conditions for Bernoulli matrices in order to compute a suitable Slater point and include additional stopping criteria in our algorithm. Finally, we illustrate our method in Section 6.3.7 before we conclude with the numerical results.

[1] Our parameter d corresponds to the parameter n in Qaqish (2003).

6.1 Problem formulation

More precisely, we consider the following research question: For a given symmetric matrix $B \in \mathbb{R}^{d\times d}$, decide if there exists a d-dimensional random vector X on some probability space $(\Omega, \mathcal{A}, \mathbb{P})$ such that each component X_i, $i = 1, \ldots, d$ is Bernoulli-distributed and such that

$$B = \mathbb{E}_{\mathbb{P}}[XX^\top]. \tag{B}$$

In this case, B is called *Bernoulli-compatible* (or *Bernoulli matrix* in short), otherwise B is called *Bernoulli-incompatible*.

Following the ideas of Theorem 2.2 in Embrechts et al. (2016), or Fiebig et al. (2017), this problem can naturally be reformulated as a membership problem, since it clearly holds

$$\mathbb{E}_{\mathbb{P}}[XX^\top] = \sum_{\mathbf{p}\in\{0,1\}^d} \mathbb{P}[X = \mathbf{p}]\, \mathbf{p}\mathbf{p}^\top$$

on each probability space $(\Omega, \mathcal{A}, \mathbb{P})$.

Proposition 6.1.1
A matrix $B \in \mathbb{R}^{d\times d}$ is Bernoulli-compatible if and only if $B \in \mathcal{B}_d$, where $\mathcal{B}_d$ denotes the Bernoulli polytope

$$\mathcal{B}_d := \operatorname{conv}\left(\{\mathbf{p}\mathbf{p}^\top \mid \mathbf{p} \in \{0,1\}^d\}\right).$$

6.1.1 Preparatory definitions

With regard to the formulation of the optimization problem in the next section, let us introduce some additional preparatory definitions. We denote by $\mathbf{p}(i)$ the natural bijection between all integers from 0 to $2^d - 1$ and all $\{0,1\}$-vectors of dimension d, i.e.

$$\mathbf{p} : \{0, \ldots, 2^d - 1\} \to \{0,1\}^d,$$

defined as the inverse of the function

$$\begin{aligned} \mathbf{i}: \quad \{0,1\}^d &\to \{0,\dots,2^d-1\} \\ p &\mapsto \sum_{j=1}^{d} p_j \cdot 2^{j-1}, \end{aligned}$$

mapping $\{0,1\}$-vectors bijectively onto the integers $\{0,\dots,2^d-1\}$, see Section 6.3.7 for an illustration.

Furthermore, we denote the rank-one matrices which are defined by the dyadic product of these vectors by $\mathbf{B}_i := \mathbf{p}(i)\mathbf{p}(i)^\top$ for all i in $\{0,\dots,2^d-1\}$ to obtain an (ordered) representation of the vertices of the Bernoulli polytope.

6.1.2 Membership testing by linear programming

By virtue of the aforementioned definitions, it immediately follows from Proposition 6.1.1 that testing $B \in \mathcal{B}_d$ is equivalent to solving the following optimization problem:

$$vp(B) := \min_{a \in \Lambda_{2^d}} |||\sum_{i=0}^{2^d-1} a_i \mathbf{B}_i - B|||_\infty, \tag{6.1}$$

where $|||A|||_\infty$ denotes the matrix max-norm of the matrix A, $\Lambda_m := \{\lambda \in \mathbb{R}_+^m \mid \lambda^\top \mathbf{1}^m = 1\}$ and as in the previous chapters $\mathbf{1}^m := (1,\dots,1)^\top$. This approach follows the main idea of Lee (1993) and can also be found in Embrechts et al. (2016); the intention is to find the convex combination of the vertices $\mathbf{B}_i$ of the Bernoulli polytope which is as close as possible to the given matrix B in matrix max-norm.

Moreover, problem (6.1) can easily be reformulated as a linear program:

$$\begin{aligned} vp(B) = \min_{\substack{a \in \Lambda_{2^d} \\ \alpha \in \mathbb{R}}} \quad & \alpha \\ \text{s.t.} \quad & B \le \sum_{i=0}^{2^d-1} a_i \mathbf{B}_i + \alpha E \\ & \sum_{i=0}^{2^d-1} a_i \mathbf{B}_i - \alpha E \le B, \end{aligned} \tag{PB}$$

where $E = \mathbf{1}^d(\mathbf{1}^d)^\top$. As part of our numerical approach, we also consider the dual problem to (PB):

$$
\begin{aligned}
vd(B) := \quad & \max_{\substack{Y,\, Z \,\in\, \mathbb{S}_d \\ Y,\, Z \,\geq\, 0 \\ \gamma \,\in\, \mathbb{R}}} && \langle B, Z - Y\rangle + \ \gamma \\
& \text{s.t.} && \langle \mathbf{B}_i, Z - Y\rangle + \gamma \leq 0 \quad i = 0, \ldots, 2^d - 1 \\
& && \langle E, Z + Y\rangle = 1
\end{aligned}
\tag{DB}
$$

where $\mathbb{S}_d$ denotes the Hilbert space of all real symmetric $d \times d$ matrices with scalar product $\langle G, H\rangle := \text{trace}(G^\top H)$ and its corresponding Frobenius norm. The interpretation of the dual problem is as follows: Whenever we have $vp(B) = vd(B) > 0$, the optimal solution of the dual program (DB) represents a hyperplane, in the form of a matrix $G = Z - Y$, separating the matrix B from the Bernoulli polytope. In the case that $vp(B) = vd(B) = 0$ no such hyperplane exists. More precisely, the following theorem holds:

Theorem 6.1.2 (Membership testing by linear programming)
The following statements hold for any matrix $B \in \mathbb{S}_d$ with $B \in [0,1]^{d\times d}$:

1. *Problems (PB) and (DB) are both feasible and it holds $0 \leq vd(B) \leq vp(B) \leq 0.5$.*
2. *Problems (PB) and (DB) possess a primal optimal solution (a^*, α^*) and a dual optimal solution (Y^*, Z^*, γ^*), respectively, and strong duality holds for (PB) and (DB), i.e. $vp(B) = |||\sum_{i=0}^{2^d-1} a_i^* \boldsymbol{B}_i - B|||_\infty = \langle B, Z^* - Y^*\rangle + \gamma^* = vd(B)$.*
3. *$B \in \mathcal{B}_d \iff vp(B) = 0 \iff vd(B) = 0$.*
4. *$vd(B) > 0 \iff$ the dual optimal solution $G^* = Z^* - Y^*$ strictly separates B from the Bernoulli polytope.*

Proof. See Krause et al. (2018). □

Observe that the two conditions $B \in \mathbb{S}_d$ and $B \in [0,1]^{d\times d}$ are obviously necessary for any matrix B to be Bernoulli-compatible, cf. Proposition 6.3.2, hence,

we can restrict ourselves to such matrices in the above theorem.

Remark 6.1.3 (Early termination criterion for (DB))
An important implication of Theorem 6.1.2.(4) is that any dual feasible point with strictly positive objective value already provides a certificate that the matrix B is not Bernoulli-compatible. In particular, the dual program does not need to be solved to optimality for testing purposes in this case.

Remark 6.1.4 (Extension complexity of the correlation polytope)
Let us point out that the primal problem (PB) *has 2^d variables and, accordingly, the dual problem* (DB) *has the same number of constraints. Furthermore, it is theoretically established that* (DB) *cannot be reformulated with significantly less constraints. For instance Kaibel and Weltge (2015) demonstrated in a easily accessible way that there is no formulation with less than $\boldsymbol{O}(1.5^d)$ constraints.*

6.1.3 Sparse solutions and efficient simulation of Bernoulli vectors

In solving (PB) via successively applying the Simplex algorithm, we also provide a sparse representation of B which in particular yields an efficient simulation approach for the corresponding multivariate distribution. For $B \in \mathcal{B}_d$, we know that there is at least one representation of B which needs at most $d(d+1)/2+1$ vertices due to the well-known Theorem of Carathéodory. Furthermore, with regard to our numerical approach, we know the Simplex method will compute a basic optimal solution a for (PB), which – by straightforward application of the fundamental theorem of linear programming – implies that at most $d(d+1)$ of the a_i will be strictly positive. Moreover, a thorough inspection of the symmetric structure of the problem yields that indeed a maximum of $d(d+1)/2+1$, i.e. the same number as in Carathéodory's Theorem, entries will be non-zeros. The according results are summarized in the following corollary:

Corollary 6.1.5 (Sparse representation)
Let $B \in \mathcal{B}_d$. Then it holds:

1. *There exists a* sparse representation $B = \sum_{i=0}^{2^d-1} a_i \boldsymbol{B}_i$ *with* $|\{i \mid a_i > 0\}| \leq d^2 + d$.
2. *Algorithm 4 applied to* (PB) *always yields a sparse representation.*

Proof. The first statement follows from the Theorem of Carathéodory. The second statement follows from the fact that Column Generation builds upon repeated calls of the Simplex algorithm, which is well known to compute basic solutions, and the fundamental theorem of linear programming. □

Remark 6.1.6 (Efficient simulation of Bernoulli vectors)
In particular, Corollary 6.1.5 yields an efficient simulation algorithm for the corresponding multivariate distribution by applying a standard mixture model to the sparse representation, see Krause et al. (2018).

6.1.4 A first numerical approach

Since LPs are often considered to be the most easy-to-solve optimization problems, the straightforward approach is to apply standard LP-solvers to solve problems (PB) and (DB). Accordingly, we applied IBM's ILOG CPLEX for various LP-formulations of (PB) and (DB), see Section 6.4.2, and illustrate the cpu time and typical[2] memory usage in Figure 6.1. For small dimensions, i.e. $d \leq 17$, we were able to solve all instances within a second. However, due to memory limitations, we were not able to solve any problem instance for $d > 20$. To address these difficulties, we propose a Column Generation approach.

[2]It is very difficult to exactly measure the average memory usage of an algorithm at any given time, thus we present approximated (slightly overestimated for small dimensions) values at this point.

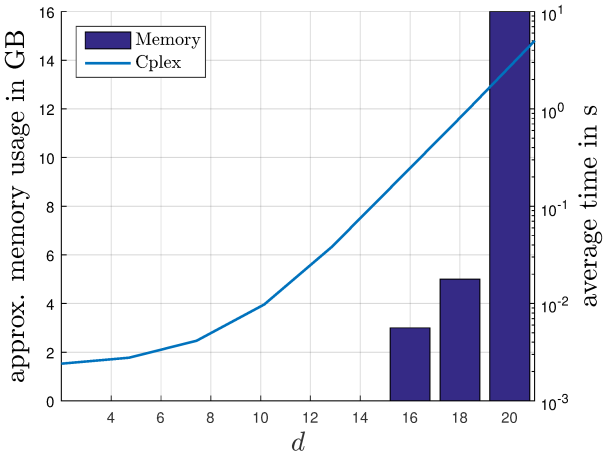


Figure 6.1: Typical memory usage and cpu time of CPLEX, averaged over all problem classes. For $d = 20$ the maximum available memory of 16 GB is reached.

6.2 Generic Column Generation

In the following, let us recall the generic Column Generation method for linear programs in canonical form; see for example Lübbecke (2010) for a detailed presentation. For this purpose, we consider a linear optimization problem (P_J) in the following form, called the *master problem*:

$$\begin{aligned} v(J) := \min_{x \in \mathbb{R}^n} \quad & \sum_{j \in J} c_j x_j \\ \text{s.t.} \quad & \sum_{j \in J} \mathbf{d}_j x_j \leq \mathbf{b} \\ & x_j \geq 0 \qquad \forall\, j \in J \end{aligned} \tag{P_J}$$

with $J = \{1, \ldots, n\}$, $\mathbf{d}_j \in \mathbb{R}^m$ for $j \in J$ and $\mathbf{b} \in \mathbb{R}^m$, where n is much larger than m. The corresponding dual[3] problem (D_J) is given by

$$\begin{aligned} v(J) = \max_{y \in \mathbb{R}^m} \quad & \mathbf{b}^\top \mathbf{y} \\ \text{s.t.} \quad & \mathbf{d}_j^\top \mathbf{y} \leq c_j \qquad \forall\, j \in J \\ & \mathbf{y} \leq 0. \end{aligned} \tag{D_J}$$

[3]In this section, we assume that the primal master problem is feasible and bounded. Hence, by strong duality, the same holds for the dual problem, and both optimal values coincide.

As Figure 6.1 shows directly solving the master problem becomes intractable beyond $n \approx 10^6$ because the large number of variables causes memory issues. Therefore, one must resort to iteratively solving *restricted master problems* (P_{I_k}) for $k = 1, \ldots, K$:

$$\begin{aligned} v(I_k) := \quad & \min_{x \in \mathbb{R}^n} \quad \sum_{j \in I_k} c_j x_j \\ & \text{s.t.} \quad \sum_{j \in I_k} \mathbf{d}_j x_j \quad \leq \quad \mathbf{b} \\ & \qquad\qquad x_j \quad \geq \quad 0 \qquad \forall\, j \in I_k. \end{aligned} \tag{P_{I_k}}$$

The sets $I_k \subset J$, usually called *inner sets*, represent subsets of indices (i.e. primal variables) which are used for the optimization – the remaining variables are simply set to 0 and thus excluded from the optimization. Starting from an initial inner set I_0, variables (columns) are then added (generated) to improve the current optimal solution when advancing from I_k to I_{k+1}. Thus, in the course of the algorithm, a finite sequence of (small) subsets $I_0 \subsetneq I_1 \subsetneq \ldots \subsetneq I_K \subseteq J$ is considered. Due to the fundamental theorem of linear programming, there always exists an optimal *basic* solution for P_J and, by construction, an optimal solution for the master problem is obtained in the restricted problem as soon as an optimal basis for the master problem is included in a set I_k. The number of elements in such a basis is m, which is assumed to be much smaller than n. Hence, in practice, I_K will hopefully be substantially smaller than J for the majority of problem instances. For our application this is indeed the case, as Figure 6.2 shows, the average number of Column Generation steps grows only moderately with the dimension d and does not exceed 700 in all our examples for $d \leq 30$.

6.2.1 The subproblem

In each iteration, the variable to be added is determined by solving a subproblem. To this end, denote by $\mathbf{x}^*_{I_k}$ an optimal solution of (P_{I_k}) and by $\mathbf{y}^*_{I_k}$ the corresponding optimal dual solution. Since $\mathbf{x}^*_{I_k}$ is feasible for (P_J), $\mathbf{x}^*_{I_k}$ is an optimal solution for the master problem if and only if $\mathbf{y}^*_{I_k}$ is feasible for (D_J).

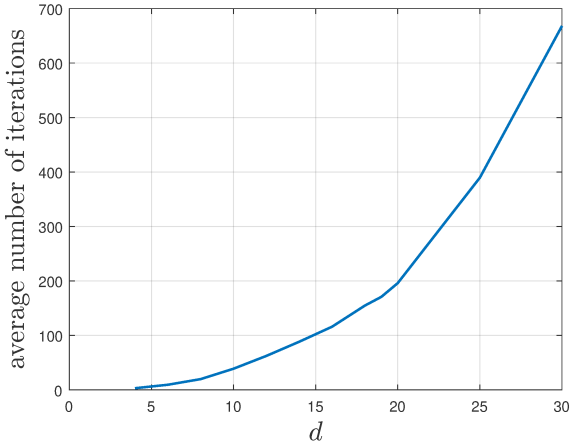


Figure 6.2: Average number of iterations of the pure Column Generation method, averaged over 100 instances of problem classes 1 to 4, see Section 6.4. It can be observed that the number of Column Generation steps only increases mildly with the dimension d.

The dual feasibility of $\mathbf{y}^*_{I_k}$ can be determined by means of the *subproblem* (SP_k):

$$h^*_{I_k} := \max_{j \in J} \; h_j(\mathbf{y}^*_{I_k}), \tag{SP_k}$$

where for some (not necessarily feasible) point $\mathbf{y}$, the violation of the j-th dual constraint is given by

$$h_j(\mathbf{y}) := \mathbf{y}^\top \mathbf{d}_j - c_j.$$

By construction, $h^*_{I_k} \leq 0$ implies $\mathbf{y}^*_{I_k}$ to be dual feasible and thus provides optimality of the current solution $\mathbf{x}^*_{I_k}$. In the case of $h^*_{I_k} > 0$ one sets $I_{k+1} := I_k \cup \{j^*_{I_k}\}$, that is, one adds the corresponding maximizing column $j^*_{I_k}$ in (SP_k) and sets $k := k+1$. Repeating this process, an optimal solution for the master problem is found after a finite number of steps.

6.2.2 Algorithm

Altogether, we obtain Algorithm 4 for the generic Column Generation:

Algorithm 4 Generic Column Generation

1. Choose an initial subset $I_0 \subset J$ such that the restricted master problem is feasible and bounded and set $k := 0$.

2. Solve the restricted master problem (P_{I_k}) to obtain $\mathbf{x}^*_{I_k}$ together with dual multipliers $\mathbf{y}^*_{I_k}$.

3. Solve the subproblem (SP_k) to obtain $h^*_{I_k}$ and a corresponding maximizer $j^*_{I_k}$.

4. If $h^*_{I_k} \leq 0$: $\mathbf{x}^*_{I_k}$ solves (P_J), stop.

   Else, set $I_{k+1} := I_k \cup \{j^*_{I_k}\}$, $k := k+1$ and go to 2.

Remark 6.2.1

Let us emphasize a few important aspects of Algorithm 4:

(i) For finite J, if (P_{I_0}) is feasible and bounded, the algorithm inherits the finiteness and correctness properties of linear programming, cf. Lübbecke (2010). This means, for some $K \leq n$, the iterate $x^*_{I_K}$ has to be an optimal solution for (P_J). Note that in the extreme case this may lead to $I_K = J$. In practice, as can be depicted from Figure 6.2, the average number of iterations only increases rather mildly with the dimension d.

(ii) By construction, any two restricted optimal solutions $\boldsymbol{x}^*_{I_k}$ and $\boldsymbol{x}^*_{I_l}$ with $k < l$ satisfy $\boldsymbol{c}^\top \boldsymbol{x}^*_{I_k} \geq \boldsymbol{c}^\top \boldsymbol{x}^*_{I_l}$, i.e. the optimal values of the restricted problems converge from above to the optimal value of the master problem.

(iv) It is critical to the overall efficiency of the Column Generation approach that both the restricted LP as well as the subproblem of determining the most violating constraint can be solved efficiently. In most successful applications of Column Generation, the problem structure of the subproblem can be exploited to avoid solving by full enumeration. For more details on the efficient solution of the subproblem in the present context let us refer to Section 6.3.1.

6.3 Column Generation in (PB)

On the basis of Algorithm 4, we develop a Column Generation method specifically tailored to the membership problem on the Bernoulli polytope in its linear formulation (PB). To this end, we start by considering the corresponding subproblem and introduce a novel dual bound by means of shifting infeasible dual iterates towards a Slater point. Furthermore, based on necessary and sufficient conditions for Bernoulli matrices, we present two methods for computing initial sets I_0 and incorporate specific stopping criteria in our approach. Finally, we summarize our numerical approach in Algorithms 5 and 6.

6.3.1 Efficient solution of (SP_k)

As mentioned above, an efficient implementation of the Column Generation is obtained if subproblem (SP_k) is solved efficiently. To avoid the full enumeration of all constraints, let us now exploit the specific structure of (DB): Given a dual variable $\mathbf{y} = (Y, Z, \gamma)$ its maximum dual violation can be computed as follows:

$$\begin{aligned}\max_{j\in J}\ h_j(\mathbf{y}) &= \max_{j\in J}\ \gamma + \langle \mathbf{B}_j, Z - Y\rangle \\ &= \gamma + \max_{j\in J}\ \langle \mathbf{p}(j)\mathbf{p}(j)^\top, Z - Y\rangle \\ &= \gamma + \max_{\mathbf{p}\in\{0,1\}^d}\ \mathbf{p}^\top (Z - Y)\mathbf{p}\end{aligned}$$

The index of the corresponding maximizer $\mathbf{p}^*$ can be immediately computed via (6.1.1). Therefore, finding the most violating constraint and computing the maximum violation boils down to solving the binary quadratic program

$$\max_{\mathbf{p}\in\{0,1\}^d}\ \mathbf{p}^\top G\mathbf{p}, \tag{SP-BQP}$$

with $G = Z - Y$. For this problem, it is well-known that it is NP-hard, as long as no special structure in G can be assumed, see, e.g., Padberg (1989), which is the case in the present situation. Until today, exact solution methods seem to be limited to a few hundred variables at most, see, for instance, Kochenberger

et al. (2014). In our implementation, we have solved (SP-BQP) of dimension $d \leq 40$ by CPLEX, which led to reasonable cpu time and was much more efficient than full enumeration, see Figure 6.3.

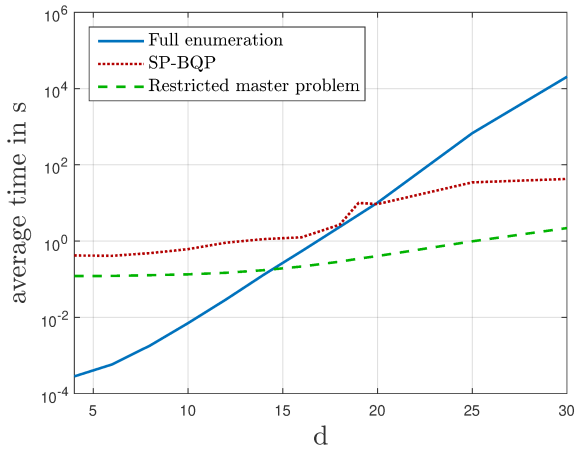


Figure 6.3: Comparison of average cpu time for one full enumeration, the solution of one binary quadratic subproblem, and the solution of one restricted master problem. The average is taken over all problem instances. It can be observed that full enumeration is much slower than solving the BQP subproblem. Further, the restricted master problem and the BQP subproblem have roughly the same computational workload.

Furthermore, one observes that the time spent for solving the restricted master problem roughly equals the time needed for finding the most violating constraint. This indicates that Algorithm 4 can only be improved in terms of cpu time, if both the LPs and the binary quadratic subproblems can be solved much faster.

6.3.2 Dual bound

As mentioned in Remark 6.1.3, Algorithm 4 can be terminated early in case a separating hyperplane is found. By linear duality we know that any feasible solution to (D_J) provides a lower bound to the optimal value of (P_J). Unfortunately, the dual solution $\mathbf{y}^*_{I_k}$ provided at step k is — in general, and excluding the optimal case — infeasible for (D_J). Therefore, we consider an additional dual bound based on a Slater point of the dual problem in the gist of Daum and Werner (2011).

Proposition 6.3.1
Let $\boldsymbol{y}$ be infeasible for (D_J). Further, let $\boldsymbol{y}_s$ be a Slater point for (D_J), i.e. $h^*(\boldsymbol{y}_s) < 0$. Then there exists some $\bar{\mu} \in]0,1[$ such that $\bar{\boldsymbol{y}} = \mu \boldsymbol{y} + (1-\mu)\boldsymbol{y}_s$ is a Slater point, i.e. $h^*(\bar{\boldsymbol{y}}) < 0$, for all $0 \leq \mu < \bar{\mu}$. A suitable choice for $\bar{\mu}$ is given by

$$\bar{\mu} = \frac{-h^*(\boldsymbol{y}_s)}{h^*(\boldsymbol{y}) - h^*(\boldsymbol{y}_s)}.$$

This implies that we can shift any infeasible iterate $\mathbf{y}^*_{I_k}$ along the line towards the Slater point $\mathbf{y}_s$ to a feasible iterate $\bar{\mathbf{y}}_{I_k}$. Whenever $\bar{\mathbf{y}}_{I_k}$ has a dual function value strictly greater than zero, it constitutes a separating hyperplane in the sense of Remark 6.1.3. As we will see in the following, this additional dual bound allows for a much earlier termination of the Column Generation and thus decreases cpu time for Bernoulli-incompatible matrices significantly.

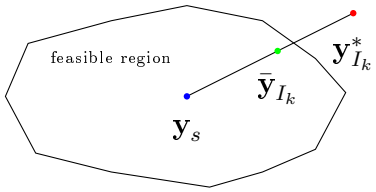


Figure 6.4: By means of a Slater point $\mathbf{y}_s$, shrink iterate $\mathbf{y}^*_{I_k}$ to a dual feasible iterate $\bar{\mathbf{y}}_{I_k}$ to obtain a lower bound for the optimal objective function value.

6.3.3 Necessary and sufficient conditions

The essential part of our numerical approach is based on the explicit solution of the LPs (PB) and (DB) by Algorithm 4. However, considerable time savings can be achieved if easy-to-verify necessary and sufficient criteria for Bernoulli compatibility of a matrix B are checked beforehand. Moreover, Propositions 6.3.3 and 6.3.4 will play a central role in the construction of suitable starting points and efficient dual solutions for stopping criteria. In the following, we introduce a selection[4] of necessary and sufficient criteria which are used in the course our numerical approach; for a more complete list as well as some novel results on bounding probabilities and scaling properties for Bernoulli-compatible matrices, we again refer the reader to Krause et al. (2018).

We start by collecting several easy-to-verify necessary conditions of Bernoulli-compatible matrices:

Proposition 6.3.2 (Necessary conditions)
Let $B \in \mathbb{R}^{d\times d}$. Then each of the following conditions is necessary for B to be Bernoulli-compatible.

1. $B \in \mathbb{S}^d$.
2. $B \in [0,1]^{d\times d}$.

[4] The majority of the corresponding criteria are found in Embrechts et al. (2016), Fiebig et al. (2017), and Deza et al. (1993); Deza and Laurent (1997).

3. *B is positive semidefinite.*

4. *B satisfies the Fréchet–Hoeffding bounds, i.e.*

$$\max(0, B_{ii} + B_{jj} - 1) \leq B_{ij} \leq \min(B_{ii}, B_{jj}).$$

Proof. See, for example, (Embrechts et al., 2016, Proposition 2.1) or Fiebig et al. (2017). □

In addition, we present a novel, but easily verifiable, necessary condition for the Bernoulli compatibility of a B matrix. With regard to our numerical approach, a dual formulation of this necessary condition will be of particular interest, as it provides an efficient way to compute suitable Slater points for shifting dual iterates as described in Proposition 6.3.1.

For this purpose, observe that for all $k \in \{0, \ldots, d\}$ and $i \in \{0, \ldots, 2^{d-1}\}$ we have

$$(\mathbf{1}^d)^\top \mathbf{p}(i) = k \implies \begin{array}{rcl} \operatorname{trace}(\mathbf{B}_i) & = & k \\ \langle E, \mathbf{B}_i \rangle & = & k^2 \end{array} \tag{6.2}$$

Then grouping binary vectors via (6.2) yields the following straightforward necessary condition: Whenever $B \in \mathbb{R}^{d \times d}$ is Bernoulli-compatible, i.e.

$$\exists a \in \Lambda_{2^d} : B = \sum_{i=0}^{2^d - 1} a_i \mathbf{B}_i,$$

we have

$$\exists b \in \Lambda_{d+1} : \begin{array}{rcl} \operatorname{trace}(B) & = & \sum_{k=0}^{d} b_k \cdot k \\ \langle E, B \rangle & = & \sum_{k=0}^{d} b_k \cdot k^2. \end{array}$$

The same methodology can be applied to construct dual solutions with very simple (low-dimensional) structure, i.e.

$$Y = \alpha_Y I + \beta_Y E \quad \text{and} \quad Z = \alpha_Z I + \beta_Z E,$$

where I is the identity matrix, which reduces the number of constraints of

(DB) to $\mathcal{O}(d)$:

$$\begin{aligned} \min_{\alpha_Y,\beta_Y,\alpha_Z,\beta_Z,\gamma\in\mathbb{R}} \quad & (\alpha_Z - \alpha_Y)\,\mathrm{trace}(B) + (\beta_Z - \beta_Y)\langle B, E\rangle + \gamma \\ \text{s.t.} \quad & \beta_Y \geq 0, \quad \beta_Z \geq 0, \\ & \alpha_Y + \beta_Y \geq 0, \quad \alpha_Z + \beta_Z \geq 0, \\ & d(\alpha_Y + \beta_Y + \alpha_Z + \beta_Z) + (d^2 - d)(\beta_Y + \beta_Z) = 1, \\ & k(\alpha_Z - \alpha_Y) + k^2(\beta_Z - \beta_Y) + \gamma \leq 0, \quad k = 0, \ldots, d. \end{aligned} \tag{DB$_l$}$$

These findings are summarized in the following proposition:

Proposition 6.3.3 (Necessary condition via dual approximation)
Let $B \in \mathbb{R}^{d\times d}$ satisfy all necessary conditions from Proposition 6.3.2. Then it holds:

(i) Any feasible solution $(\alpha_Y, \alpha_Z, \beta_Y, \beta_Z, \gamma)$ of (DB$_l$) yields a feasible solution (Y, Z, γ) for (DB) with $Y = \alpha_Y I + \beta_Y E$ and $Z = \alpha_Z I + \beta_Z$.

(ii) A strictly positive optimal value of (DB$_l$) implies that B is not Bernoulli-compatible.

Proof. See Krause et al. (2018). □

This allows us to efficiently obtain feasible solutions to (DB). Moreover, as shown in our numerical investigations, slightly shifting these solutions yields effective Slater points for finding Bernoulli-incompatible matrices in practical settings via Proposition 6.3.1. Hence, we incorporate this methodology in our numerical approach in Section 6.3.4.

We conclude with a necessary and sufficient condition, which is used in the construction of starting points as shown in Section 6.3.3.

Proposition 6.3.4 (Necessary and sufficient conditions)
Let $B \in \mathbb{R}^{d\times d}$. Then B is Bernoulli-compatible $\iff$ each principal sub-matrix of B is Bernoulli-compatible.

Proof. See Krause et al. (2018). □

Initial solutions

In order to find I_0 such that the restricted master problem is feasible and the corresponding optimal solution $x^*_{I_0}$ has a reasonable small objective value, we apply two different methodologies motivated by our numerical analysis. To foster easy solution of the initial restricted master problem in the Column Generation method, we keep the size of the two corresponding sets $I_{blk} \subset J$ and $I_{ind} \subset J$ in the order of $\mathcal{O}(d^2)$. The final set I_0 is then given by the union $I_0 := I_{blk} \cup I_{ind}$.

Block heuristic The first method, called the *block heuristic*, yields the set I_{blk}, and is independent of the inspected matrix $B \in \mathbb{R}^{d\times d}$, as it employs a straightforward selection of binary vectors where all entries equal to 1 occur in blocks, i.e.

$$I_{blk} = \{j \mid \exists 1 \leq l \leq u \leq d : \mathbf{p}(j)_i = \begin{cases} 1 \text{ for } l \leq i \leq u \\ 0 \text{ otherwise} \end{cases} \} \cup \{0\}.$$

Observe that therefore $I_{blk} \neq \emptyset$, which guarantees the feasibility of the restricted master problem. An illustration for $d = 4$ is given in Section 6.3.7.

Inductive heuristic The second method, called the *inductive heuristic*, constructs a set I_{ind} by an iterative approach based on Proposition 6.3.4. In the case that $d > 14$, we solve (PB) for A^k, the first principal sub-matrix of dimension $k := 14$ of B. For this dimension, the primal LP can be solved very efficiently. If the corresponding solution implies that A^k is not Bernoulli-compatible, the same holds for B due to Proposition 6.3.4 and we stop our investigations, i.e. we implicitly test a necessary condition. In the opposite case, this "local" solution is extended to construct a feasible starting point for the original LP: Initially, since A^k is Bernoulli-compatible, the positive coefficients in the optimal solution yield a convex representation

$$A^k = \sum_{j=0}^{2^k-1} \lambda_j \mathbf{B}^k_j, \quad \lambda \in \Lambda_{2^d},$$

where the $\mathbf{B}_j^k \in \mathbb{R}^{k\times k}$ define the vertices of the Bernoulli polytope for dimension $d = k$, see Corollary 6.1.5. Thus, we collect the indices of all vertices with strictly positive coefficients in the auxiliary set $S^k := \{0 \leq j \leq 2^{d-1} \mid \lambda_j > 0\}$ and compute I_{ind} by the following iterative process:

1. If $k = d$ stop; otherwise set $I_{ind} := \emptyset$.
2. For each $j \in S^k$: Add j and $j + 2^k$ to I_{ind}.
3. For $I_{ind} = \{j_1, \ldots, j_l\}$; check whether I_{ind} yields a convex representation of A^{k+1}, i.e. check if $\exists \lambda \in \Lambda_l$ such that $A^{k+1} = \sum_{i=1}^{l} \lambda_i \mathbf{B}_{j_i}^{k+1}$.
   a) In the positive case, set $S^{k+1} := \{j_i \in I_{ind} \mid \lambda_i > 0\}$, $k := k+1$ and goto 1.
   b) In the negative case, stop.

A few explanations: Observe that adding j and $j + 2^k$ in Step 2 is equivalent to adding the indices of two extended vectors of $\mathbf{p}_j$ since

$$j = \mathbf{i}\left((\mathbf{p}_j, 0)^\top\right) \quad \text{and} \quad j + 2^k = \mathbf{i}\left((\mathbf{p}_j, 1)^\top\right)$$

by means of the inverse $\mathbf{i}$ of the natural bijection of binary vectors and integers given in Section 6.1. Moreover, we compute a minimal convex representation in Step 3a), which ensures that $|I_{ind}| \in \mathcal{O}(d^2)$.

6.3.4 Stopping criteria

Finally, we extend the Column Generation by introducing specific stopping criteria for our application on the basis of Theorem 6.1.2. On the one hand, as we know that the optimal value of the primal problem is always non-negative, we can stop the Column Generation as soon as an objective value of 0 is obtained, in other words: $v(I_k) = 0$ for some iteration k implies that $\mathbf{x}^*_{I_k}$ solves (P_J). In this case the given matrix B is Bernoulli-compatible.

Moreover, we introduce a further stopping criterion which provides a negative certificate. On the basis of Proposition 6.3.3 we compute a Slater point y_s which

is used to obtain feasible dual iterates $\bar{\mathbf{y}}_{I_k}$ via Proposition 6.3.1. Whenever we have $\mathbf{b}^\top \bar{\mathbf{y}}_{I_k} > 0$, for some iteration k, this implies B is not Bernoulli-compatible by Remark 6.1.3.

6.3.5 Algorithms

We summarize our Column Generation approach in Algorithms 5 and 6. In order to thoroughly evaluate the performance of the pure Column Generation approach, we incorporate Algorithm 5 in our numeric investigations, where we keep the extensions not related to Column Generation to a minimum. For this purpose, the only adjustments compared to the generic Column Generation in Algorithm 4 are the solution of the subproblem as a (BQP) and the cpu of the initial set I_0 by the block heuristic of Section 6.3.3. Observe that $I_{blk} \neq \emptyset$

Algorithm 5 Pure Column Generation for the Bernoulli-Problem

1. Compute the initial set $I_0 = I_{blk}$ and set $k := 0$.
2. Solve the restricted master problem (P_{I_k}) to obtain $\mathbf{x}^*_{I_k}$ together with dual multipliers $\mathbf{y}^*_{I_k}$.
3. Solve the subproblem (SP_BQP_k) to obtain $h^*_{I_k}$ and a corresponding maximizer $j^*_{I_k}$.
4. If $h^*_{I_k} \leq 0$: $\mathbf{x}^*_{I_k}$ solves (P_J), stop.

   Else, set $I_{k+1} := I_k \cup \{j^*_{I_k}\}$, $k := k+1$ and go to 2.

in Step 1 of the algorithm ensures that (P_{I_0}) is feasible.

In contrast, we include all problem-specific extensions presented above in Algorithm 6. More precisely, in comparison to the basic Column Generation approach of Algorithm 5, we test the necessary conditions of Proposition 6.3.2 before invoking the Column Generation. Furthermore, we modified Step 2 by applying both the block and the inductive heuristic introduced in Section 6.3.3 to compute a more sophisticated initial set I_0 incorporating information of the

matrix B. In addition, we use Proposition 6.3.3 (and a slight subsequent shift) to obtain a Slater point for (DB). This in turn is beneficial for determining separating hyperplanes for Bernoulli-incompatible matrices in Step 4, see Section 6.3.2, which is followed by the newly introduced stopping criteria in Steps 5 to 6 providing possible early certificates for Bernoulli-(in)compatible matrices as shown in Section 6.3.4. Observe that Step 2 implicitly includes the

Algorithm 6 Enhanced Column Generation for the Bernoulli-Problem

1. Check the necessary conditions of Propositions 6.3.2.
2. Compute the initial set $I_0 = I_{blk} \cup I_{ind}$ and a Slater point y_s and set $k := 0$.
3. Solve the restricted master problem (P_{I_k}) to obtain $\mathbf{x}^*_{I_k}$ together with dual multipliers $\mathbf{y}^*_{I_k}$.
4. Shift $\mathbf{y}^*_{I_k}$ to $\bar{\mathbf{y}}_{I_k}$.
5. If $v(I_k) = 0$, B is Bernoulli-compatible, stop.
6. If $\mathbf{b}^\top \bar{\mathbf{y}}_{I_k} > 0$, B is not Bernoulli-compatible stop.
7. Solve the subproblem (SP_BQP_k) to obtain $h^*_{I_k}$ and a corresponding maximizer $j^*_{I_k}$.
8. If $h^*_{I_k} \leq 0$: $\mathbf{x}^*_{I_k}$ solves (P_J), stop.

   Else, set $I_{k+1} := I_k \cup \{j^*_{I_k}\}$, $k := k + 1$ and go to 3.

verification of the necessary conditions given by Propositions 6.3.4 and 6.3.3. Moreover, if an optimal solution is already contained in I_0, i.e. we have a positive certificate for Bernoulli-compatibility after Step 2, we stop the algorithm.

6.3.6 Implementation details

Finally, we list some implementation details of the Column Generation approach.

Hardware and software choice

As in the former chapters of this thesis, all tests were performed on a standard personal computer (processor: Intel Core i5-4090, 3.30 GHz, RAM: 16GB). The implementation of the Column Generation method was carried out in MATLAB 2015B and we have used IBM's ILOG CPLEX 12.6.2 for MATLAB to solve the arising LPs and BQPs. All MATLAB codes are available by email request to the author.

Cycling of Column Generation

One typical numerical issue in Column Generation is that due to numerical problems, some dual constraint corresponding to an index i of the inner set I_k might become slightly infeasible in some iteration. Especially in the proximity of the optimal solution, this can lead to the repeated introduction of this index i to the inner set and thus cycling. Using the feasibility-tolerance parameter of CPLEX, we avoided this difficulty by forcing the LP solver to produce a "feasible enough" solution in each iteration.

Computational accuracy

In the context of our numerical investigations a sufficiently accurate relative accuracy was given by classifying a matrix B as Bernoulli-compatible whenever $vp(B) < 10^{-6}$. However, in the case that $\mathrm{trace}(B)$ is rather small, an accuracy of 10^{-6} might not be adequate and we propose to test the rescaled matrix $\frac{1}{\mathrm{trace}(B)}B$ instead of B, which is equivalent according to the scaling properties discussed in Krause et al. (2018).

Numerical solution of (SP-BQP)

We have both solved (SP-BQP) directly, using the MIQP solver of CPLEX, as well as the linearization of (BQP) (cf. Padberg (1989)) via the MILP solver of CPLEX. Since both approaches produced very similar results, we have chosen the MIQP solver for the subsequent numerical studies.

One result of our numerical tests was that, instead of solving (SP-BQP) exactly, it is more efficient to reasonably approximate the solution and only compute exact solutions if no more violated constraints can be found. The approximation was controlled by modifying the MIP tolerances of the CPLEX solver. This observation is supported by recent literature on BQPs, see for example Kochenberger et al. (2014).

6.3.7 Illustration of the Column Generation

In this section, we illustrate the essential components of the Column Generation algorithm (Algorithm 4). To begin with, let us illustrate the (ordered) representation of the $2^d - 1$ vertices of the Bernoulli polytope introduced in Section 6.1.1. Recall that the j-th vertex is represented by

$$\mathbf{B}_j := \mathbf{p}(j)\mathbf{p}(j)^\top, \quad j = 0, \dots, 2^d - 1,$$

where the binary vectors $\mathbf{p}(j)$ are given by the natural bijection $\mathbf{p}$ of Section 6.1.1, i.e.

$$\mathbf{p}(0) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \mathbf{p}(1) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \mathbf{p}(2) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \dots, \mathbf{p}(2^d - 1) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}.$$

Furthermore, we associate each matrix $\mathbf{B}_j$ with some non-negative variable a_j. These variables serve – if a representation is found – as weights in the convex representation of the Bernoulli matrix B.

Initialization: The Column Generation starts by solving the restricted master problem on some subset of variables $I_0 := I_{blk} \cup I_{ind}$. As the inductive heuristic is only applied for $d > 14$ it is not suitable for illustrations on paper, hence, we have $I_0 = I_{blk}$ in the following. Taking the case $d = 4$ as an example, the variables with indices

$$I_{blk} = \{0, 1, 2, 4, 8, 3, 6, 12, 7, 14, 15\}$$

are selected in the block heuristic. These variables correspond to the following set of binary vectors (the logic behind is the selection of "blocks of ones" with increasing size)

$$\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \right\}.$$

Iteration: In each iteration step k, a new variable is introduced to the inner set I_k and the problem is solved again. Based on the initial set I_0 above, in Table 6.1, the first variable to enter the inner set has index 9 and corresponds to the binary vector

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

This process is explicitly depicted for four problem instances in Tables 6.1, 6.2, 6.3, and 6.4. The state of all variables throughout the iterations of the Column Generation is given.

Examples: In the remainder of this section, we illustrate the Column Generation on four problem instances. First, we present two examples from problem class 1 (see Section 6.4) with $d = 4$ and $2^d - 1 = 15$ variables, where the parameters are chosen such that only the first matrix is Bernoulli-compatible.

Second, we illustrate the Column Generation on two examples also considered in Embrechts et al. (2016) who investigated the following research question: For which $\beta \in [0,1]$ is the matrix

$$\Gamma_d(\beta) = \begin{pmatrix} 1 & 0 & \cdots & 0 & \beta \\ 0 & 1 & \cdots & 0 & \beta \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & \beta \\ \beta & \beta & \cdots & \beta & 1 \end{pmatrix}$$

a matrix of pairwise (either lower or upper) tail-dependence coefficients? This is equivalent to checking whether $\Gamma_d(\beta)/d$ is Bernoulli-compatible, cf. Krause et al. (2018). Thus, we incorporate two examples in our illustration where we keep $d = 4$: In Tables 6.3 and 6.4 we set $\beta = \frac{1}{8}$ and $\beta = \frac{1}{2}$, respectively. As expected, the former matrix turns out to be Bernoulli-compatible (note that the final α in Table 6.3 is zero), while the latter is not. Interestingly, for $\beta = \frac{1}{2}$ the matrix is still positive semi-definite.

We further observe that for all problem classes the primal solution does not change for two (or more) successive iterations. In such a case, adding the index corresponding to the maximally violated dual constraint has no effect on the primal solution. It does, however, add one more dual constraint to the inner problem and thereby different dual multipliers are generated from the primal solution.

In the i-th row and j-th column of the first $2^d - 1 = 15$ columns, the value of a_j of the vertex $\mathbf{B}_j$ of the Bernoulli polytope in the i-th iteration is given. Here, empty cells indicate that the corresponding variable is not yet added to the restricted master problem. Non-empty cells contain the value of the corresponding variable in the optimal solution of the restricted problem at the current iteration. Furthermore, the last two columns show the respective objective value α and the maximal violation of a dual constraint in each iteration.

Its\B_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	α	Vio
0	0	.182	.089	.093	.182		0	0	.182				0		.093	.179	.046	.143
1	0	.098	.157	.059	.157		0	.059	.098	.118			.059		.059	.136	.030	.182
2	0	0	.128	.128	.128	.108	0	.020	.108	.148			.020		.128	.088	.010	.200
3	0	0	.113	.113	.113	.113	0	.050	.062	.163	.050		.050		.113	.063	0	0

Table 6.1: Depicted is the Column Generation method for some Bernoulli-compatible matrix B of problem class 1, where the parameters are $\eta = 0.5$ and $\kappa = 0.9$. Column j, $0 \leq j \leq 15$, of this (and the following) table(s) represents one vertex B_j of the Bernoulli polytope and the corresponding coefficients a_j change over the iterations (rows). An empty cell indicates that the respective variable has not yet been added to the restricted master problem. Non-empty cells contain the value of the variable in the optimal solution of the restricted problem at the current iteration. The last two columns show the objective value α and the maximal violation of a dual constraint for each iteration. Row 0 represents the solution on the starting set I_0.

Its\B_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	α	Vio
0	0	.224	.120	.104	.224		0	0	.224				0		.104	0	.072	.143
1	0	.159	.159	.091	.159		.091	0	.250				0	.091	0	0	.059	.125
2	0	.159	.159	.091	.159	0	.091	0	.250				0	.091	0	0	.059	.125
3	0	.189	.189	0	.108	.081	.081	0	.189			.081	.081	0	0	0	.049	.111
4	0	.189	.189	0	.108	.081	.081	0	.189		0	.081	.081	0	0	0	.049	.111
5	0	.141	.141	.073	.141	.073	.073	0	.141	.073	.073	0	.073	0	0	0	.041	0

Table 6.2: Visualization of the Column Generation method for some non-Bernoulli-compatible matrix B of class 1, where $\eta = 0.4$ and $\kappa = 0.2$. For a full description of this table, see the caption to Table 6.1. Note that the final α is positive in this example.

Its\B_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	α	Vio
0	.109	.219	.219	0	.219		0	0	.219				0		0	.016	.016	.500
1	.109	.219	.219	0	.219		0	0	.219				0	0	0	.016	.016	.500
2	.109	.219	.219	0	.219		0	0	.219	0			0	0	0	.016	.016	.500
3	.094	.219	.219	0	.219		0	0	.156	.031	.031		.031	0	0	0	0	0

Table 6.3: $\Gamma_4(\frac{1}{8})/4$ is Bernoulli-compatible. For a full description of this table, see Table 6.1.

Its\B_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	α	Vio
0	.438	.125	.125	0	.125		0	0	.125				0		0	.063	.063	.500
1	.438	.125	.125	0	.125		0	0	.125				0	0	0	.063	.063	.500
2	.438	.125	.125	0	.125		0	0	.125	0			0	0	0	.063	.063	.500
3	.354	.125	.125	0	.125		0	0	0	.083	.083		.083	0	0	.021	.021	.167
4	.357	.125	.125	0	.125		0	0	0	.071	.071	.018	.071	.018	.018	0	.018	0

Table 6.4: $\Gamma_4(\frac{1}{2})/4$ is not Bernoulli-compatible (but positive semi-definite). For a full description of this table, see Table 6.1.

6.4 Numerical analysis

In this section, we report in detail the setup of our case study based on selected test instances, before we discuss the main numerical findings. In summary, our numerical analysis shows that the pure Column Generation method (Algorithm 5) is quite efficient up to $d = 30$. Furthermore, we can efficiently test for Bernoulli-compatibility up to the dimension $d = 40$ using several problem-specific enhancements in Algorithm 6.

6.4.1 Test problems

Unfortunately, for testing Bernoulli-compatibility, there is no common test library available. Therefore, we have come up with five different families of test problems for the numerical tests. The first two represent specifically selected parametrized problem classes, whereas the last three are based on random combinations of vertices of the Bernoulli polytope $\mathcal{B}_d$. All test cases satisfy the necessary conditions from Proposition 6.3.2, besides a few exceptions for $d < 6$, as well as a significant number of instances in class 1 which violate the Fréchet–Hoeffding bounds, see Figures 6.5 and 6.9.

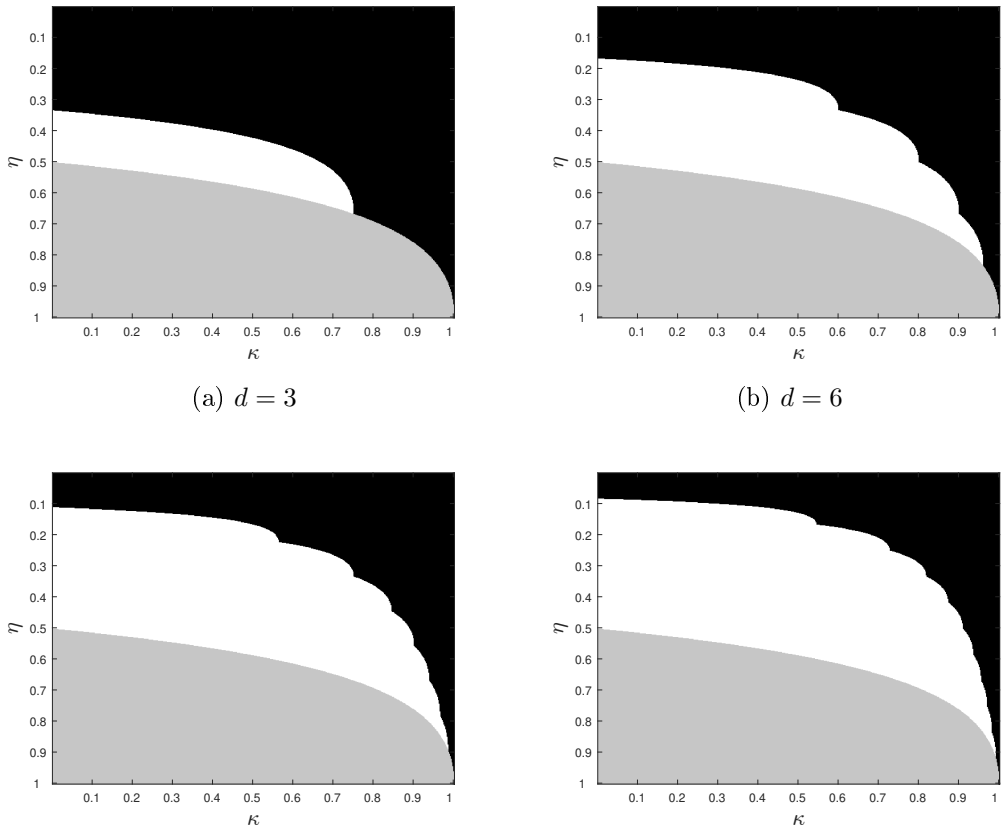
Problem class 1: The matrices B of the first class are given by

$$B := (\eta - \eta^2\kappa)I + \kappa\eta^2 E,$$

for some $0 \leq \eta \leq 1$ and $0 \leq \kappa \leq 1$, where I denotes the identity matrix. Instances of this problem class can be either Bernoulli-compatible or not, depending on the parameters, see Figure 6.5.

Problem class 2: Matrices B of the second class are given by

$$\begin{aligned} B_{ii} &= \frac{p}{p+q}, && i = 1, \ldots, d, \\ B_{ij} &= \frac{p}{p+q}\frac{p+1}{p+q+1}, && 1 \leq i \neq j \leq d, \end{aligned}$$



(a) $d = 3$ (b) $d = 6$

(c) $d = 9$ (d) $d = 12$

Figure 6.5: Bernoulli-compatibility of matrices in problem class 1, depending on η and κ for $d = 3, 6, 9$ and 12. Black areas indicate Bernoulli-compatible matrices; gray areas indicate Bernoulli-incompatible instances that violate the Fréchet–Hoeffding bounds and white areas indicate Bernoulli-incompatible matrices that satisfy the Fréchet–Hoeffding bounds.

for some $0 < p < 1$ and $0 < q < 1$. Instances of this problem class are always Bernoulli-compatible: Draw $(U_1, \ldots, U_d)$ from a copula that is defined as the convex combination of $1/(p+q+1)$ times the comonotonicity copula and $(p+q)/(p+q+1)$ times the independence copula. Further, let $X_i := \mathbf{1}_{U_i \leq p/(p+q)}$ for $i = 1, \ldots, d$. It is then easily verified (by conditioning) that $\mathbb{E}[X_i X_j] = \frac{p(1+p)}{(1+p+q)(p+q)}$ for $i \neq j$. Moreover, $\mathbb{E}[X_i^2] = \frac{p}{p+q}$ by the uniform margins property of a copula.

Problem class 3: The third class constitutes randomly generated matrices

$$B := \sum_{k=1}^{n} \lambda_{i_k} \cdot \mathbf{B}_{i_k}.$$

The number of terms n is uniformly distributed in the interval $[d^2, d^4]$ and the vertices B_{i_k} are uniformly distributed over all vertices of $\mathcal{B}_d$. Finally, the non-zero coefficients $\lambda_{i_1}, \ldots, \lambda_{i_n}$ of the convex combination are uniformly distributed on the standard (n)-simplex, i.e. sampled from a Dirichlet distribution. As a convex combination of extremal points of $\mathcal{B}_d$, B is always Bernoulli-compatible.

Problem class 4: Based on class 3, the matrices B of problem class 4 are given by

$$B := A + \frac{1}{10}\mathbf{B}_j,$$

where the matrix A is generated as in problem class 3. One specific index j with $\lambda_j > 0$ is randomly chosen and increased by 0.1. In practice, this usually leads to Bernoulli-incompatible matrices for $d > 14$.

Problem class 5: Finally, we also consider a problem class which is supposed to produce "hard" problem instances, by setting

$$B := A + \frac{1}{d}\mathbf{B}_j.$$

Now, the matrix B is derived in a similar fashion as in class 4, however, the shift decreases with increasing dimension. This is supposed to produce both

Bernoulli-compatible and Bernoulli-incompatible matrices which are close to the Bernoulli polytope's boundary[5].

In Figure 6.6, we have illustrated the distance of randomly generated instances from problem classes 4 and 5.

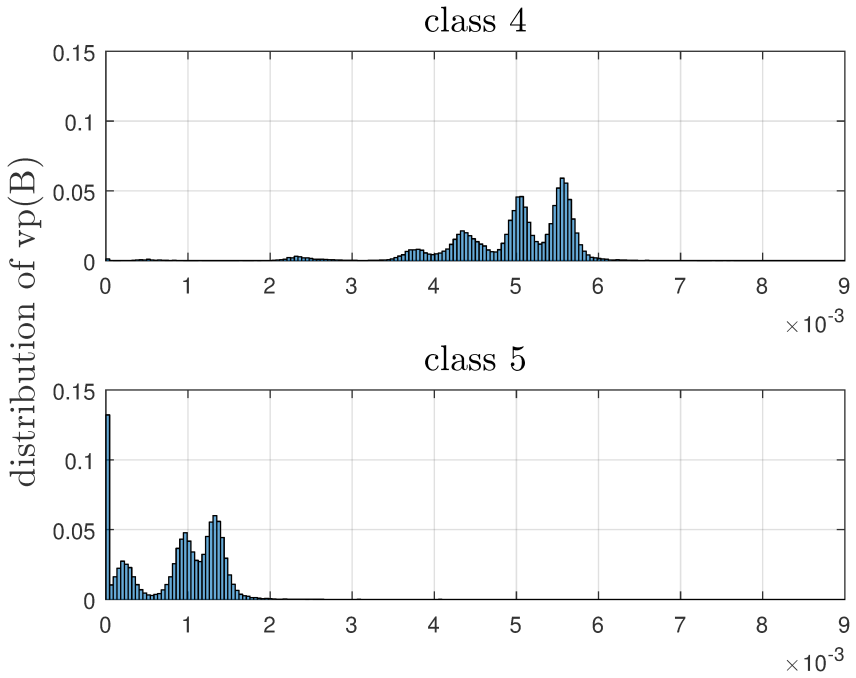


Figure 6.6: Distribution of the distance $vp(B)$ to the Bernoulli polytope for matrices B from problem classes 4 and 5 in the case of $d = 14$. Instances of class 4 are clustered around $5 \cdot 10^{-3}$ which is significantly larger than our threshold for Bernoulli-compatible matrices of 10^{-6}, see also Section 6.3.6. For class 5, $vp(B)$ is significantly lower for most instances, whereof approximately ten percent satisfy $vp(B) < 10^{-6}$.

It can be observed that class 4 usually only contains Bernoulli-incompatible matrices, whereas the situation for class 5 is mixed, with on average much smaller distance than for class 4.

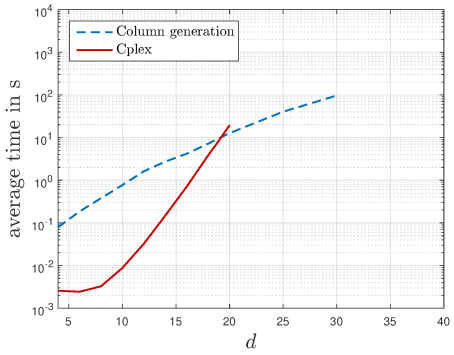
[5]This assumption is supported by our numerical findings in this section.

6.4.2 Results

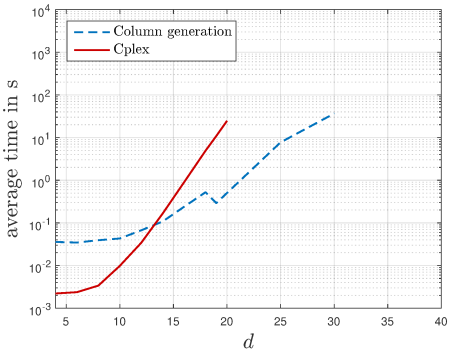
In this section, we analyze the performance of the Column Generation method in two steps. In order to evaluate the efficiency of the pure Column Generation approach, we first compare Algorithm 5 of Section 6.3.5 to standard LP solvers. In a second step, we analyze the added value of our problem-specific extensions which are not directly linked to Column Generation in Algorithm 6. As mentioned before, any problem formulation of (PB) and (DB) can be solved directly by an arbitrary LP solver. Therefore, we tested different combinations of problem formulations and LP solvers of CPLEX. As expected, we found that – if one is just interested in any solution – the most efficient approach was to solve the feasibility problem of (PB) with the primal Simplex of CPLEX. None of the direct solvers could solve the problem for $d > 20$ due to memory issues. Therefore, direct methods are omitted for $d > 20$ in the following.

Let us start with the Column Generation approach in its pure form represented by Algorithm 5. The corresponding results are presented in Figure 6.7 where it is observed that the pure Column Generation is able to solve instances up to $d = 30$, whereas all direct approaches fail to solve instances where $d > 20$. The cpu time can be further significantly reduced when we include the problem-specific extensions of Algorithm 6. The performance of this enhanced method is illustrated in Figure 6.8. For problem class 4, we observe that for $d > 10$ only Bernoulli-incompatible instances are produced and all these instances can be solved via dual approximation (see Figure 6.9). Again, very similar results as for the pure Column Generation method can be observed. For $d \geq 15$, the enhanced Column Generation method starts to outperform the direct solvers, as for a significant number of instances, the solution of large LPs can be avoided. This is also the main reason why the enhanced method is clearly faster than the pure Column Generation method as shown in Figure 6.8. This is further illustrated in Figure 6.9 in terms of statistical information on how problem instances were solved. The following categories are represented:

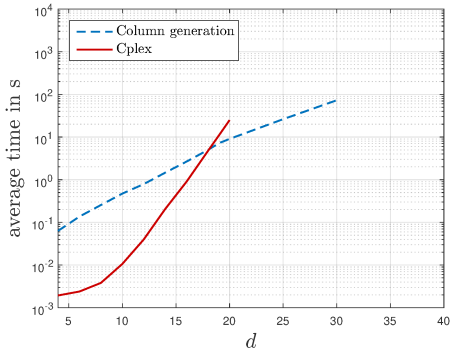
- "Necessary conditions" implies that one of the necessary conditions of Proposition 6.3.2 is not satisfied.



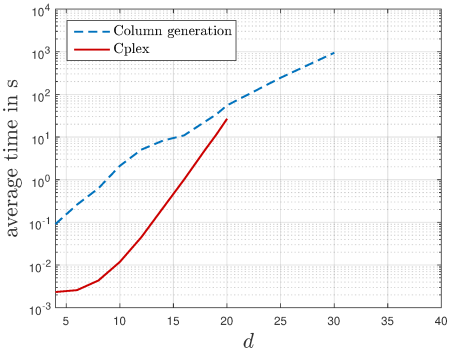
(a) Problem class 1



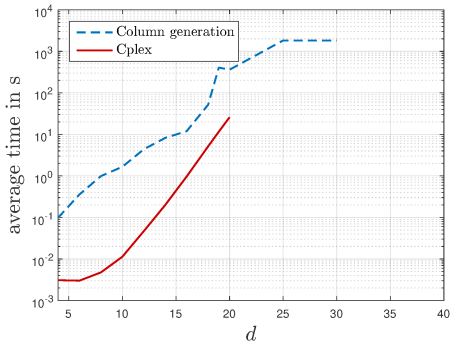
(b) Problem class 2



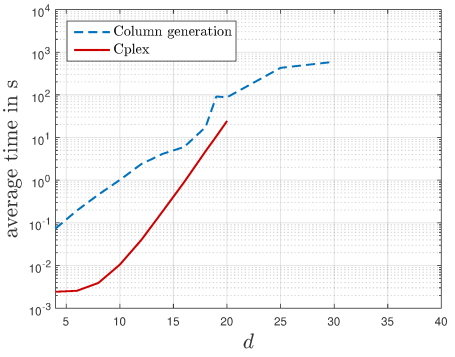
(c) Problem class 3



(d) Problem class 4

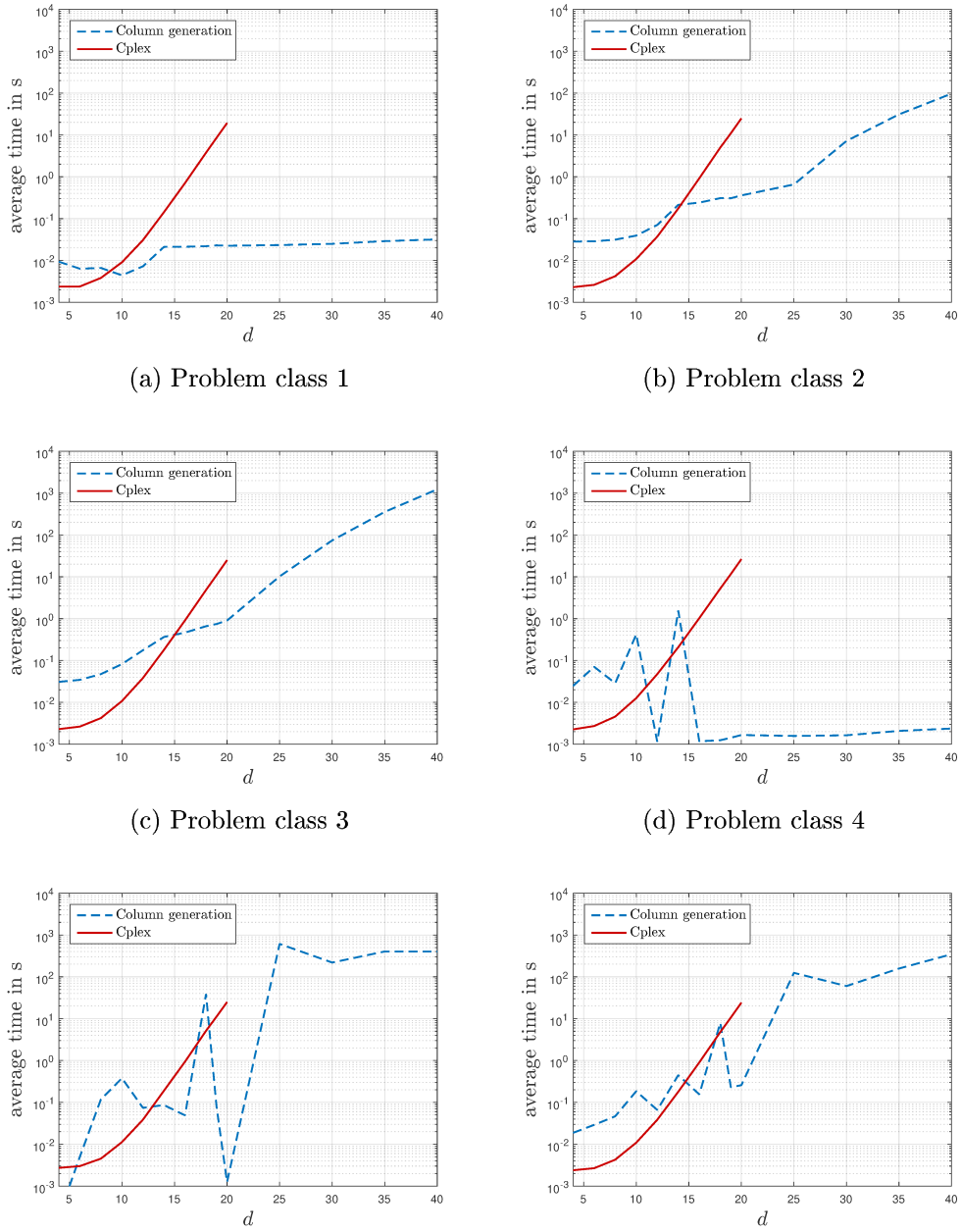


(e) Problem class 5



(f) Average for classes 1 to 5

Figure 6.7: In each subplot, the averaged cpu time is illustrated for the pure Column Generation method. The average is taken over 100 instances for each d. The computation was aborted, whenever the time limit of 30 minutes was reached. This was the case for all instances of class 5 when $d \geq 25$.



(a) Problem class 1

(b) Problem class 2

(c) Problem class 3

(d) Problem class 4

(e) Problem class 5

(f) Average for classes 1 to 5

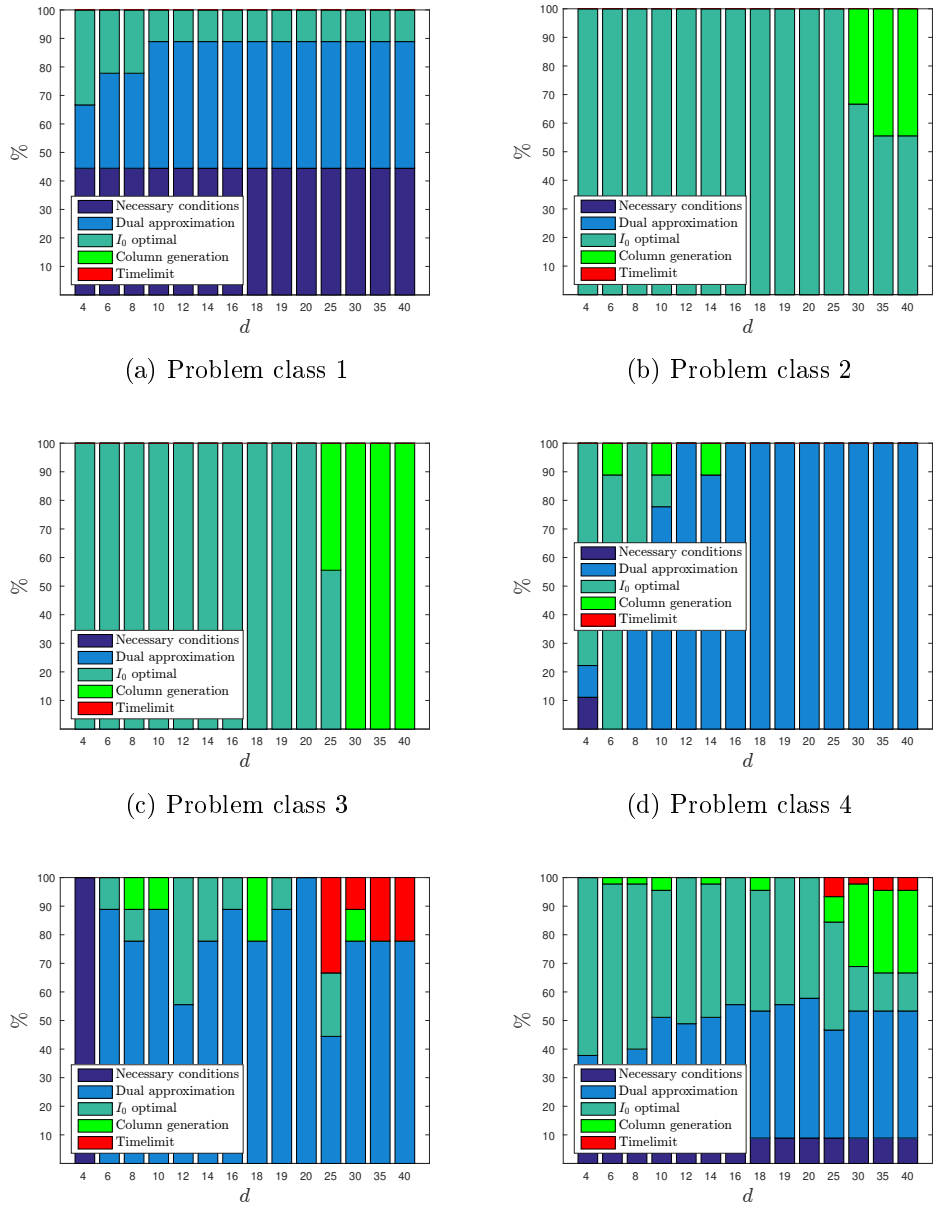
Figure 6.8: In each subplot, the averaged cpu time is illustrated for the enhanced Column Generation method. The results were obtained as described in Figure 6.7, i.e. averaged over 100 instances for each d with a time limit of 30 minutes.

- "Dual approximation" implies that a separating hyperplane was found, i.e. the stopping criterion in Step 6 of Algorithm 6 was satisfied.
- "I_0 optimal" indicates that the inner set $I_0 := I_{blk} \cup I_{ind}$ yields an optimal solution.
- "Column generation" implies that a positive certificate was found by running the Column Generation.
- "Time limit" implies that the time limit of 30 minutes for the Column Generation was reached.

Figure 6.9 shows that most problem instances can be successfully solved by primal or dual heuristics as long as d is small to medium sized. In the case of large d and Bernoulli-compatible matrices all problem instances are ultimately solved by Column Generation, whereas most Bernoulli-incompatible matrices could be identified via dual approximation.

6.5 Conclusion and outlook

The main objective of this exposition was to present an approach to test whether a matrix $B \in \mathbb{R}^{d \times d}$ is a *Bernoulli-compatible matrix* or not. Technically, we aim at finding a mixture model representation by solving a linear program, as pioneered by Lee (1993). To deal with the problem of exponentially many variables in the primal LP, we proposed to solve larger LPs with a Column Generation method. For an efficient implementation of the Column Generation method, it was crucial to replace the full enumeration for identifying the most violating constraint. Due to the specific structure of the problem, this can be achieved by solving a binary quadratic program. Although the membership testing problem is known to be NP-complete, we observe very promising performance of such a pure Column Generation method up to dimension $d = 30$ on a variety of test problems. To improve the performance of the pure Column Generation method, we have enhanced the method by a novel dual bound for early termination, which has shown to be quite effective. In addition, primal and dual heuristics, which can be efficiently tested before



(a) Problem class 1

(b) Problem class 2

(c) Problem class 3

(d) Problem class 4

(e) Problem class 5

(f) Average

Figure 6.9: In each subplot, the average percentage shares of different solution types are illustrated for the enhanced Column Generation method. The average is taken over 100 instances for each d.

applying Column Generation, further significantly increase the effectiveness of our approach.

For practical applications, an important byproduct of the derived mixture model representation is the possibility to exploit it for the simulation of Bernoulli vectors. The fact that a solution with at most $\mathcal{O}(d^2)$ vertices is identified by our approach makes this especially convenient.

Lastly, a few words on further research: As the computation for the arising LPs and BQPs are roughly the same, a significant speed up of the iterations will only be achieved if both problems can be solved faster. However, in order to reduce the number of iterations performed, a possible enhancement could be given by replacing the binary quadratic program in certain (early) iterations with heuristic approaches that add more than one variable to the restricted master problem. As for our method to produce feasible dual iterates via shifting towards a Slater point, it would be interesting to see if this approach can be transferred to other problems where suitable Slater points are available.

Bibliography

Ahrens, J. H. and Finke, G. (1980). Primal transportation and transshipment algorithms. *Zeitschrift für Operations Research*, 24(1):1–32.

Ahuja, R., Magnanti, T., and Orlin, J. (1993). *Network Flows – Theory, Algorithms, and Applications*. Prentice Hall, New Jersey.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia. PMLR.

Arthur, J. L. and Frendewey, J. O. (1994). An algorithm for generating minimum cost network flow problems with specific structure and known optimal solutions. *Networks*, 24(8):445–454.

Balakrishnan, N. and Nevzorov, V. (2005). *Dirichlet Distribution*, pages 269–276. Wiley-Blackwell.

Bradley, G. H., Brown, G. G., and Graves, G. W. (1977). Design and implementation of large scale primal transshipment algorithms. *Management Science*, 24(1):1–34.

Brenner, U. (2008). A faster polynomial algorithm for the unbalanced Hitchcock transportation problem. *Oper. Res. Lett.*, 36(4):408–413.

Brenner, U. and Struzyna, M. (2005). Faster and better global placement by a new transportation algorithm. In *Proceedings. 42nd Design Automation Conference, 2005.*, pages 591–596.

Broder, A. (1989). Generating random spanning trees. In *30th Annual Symposium on Foundations of Computer Science*, pages 442–447.

Burkard, R. E. and Çela, E. (1999). Linear assignment problems and extensions. In Du, D.-Z. and Pardalos, P. M., editors, *Handbook of Combinatorial Optimization: Supplement Volume A*, pages 75–149. Springer US, Boston, MA.

Carrière, M., Cuturi, M., and Oudot, S. (2017). Sliced Wasserstein kernel for persistence diagrams. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 664–673, International Convention Centre, Sydney, Australia. PMLR.

Charnes, A. and Cooper, W. W. (1954). The stepping stone method of explaining linear programming calculations in transportation problems. *Management Science*, 1(1):49–69.

Charnes, A., Karney, D., Klingman, D., Stutz, J., and Glover, F. (1975). Past, present and future of large scale transshipment computer codes and applications. *Computers & Operations Research*, 2(2):71–81.

Charnes, A. and Klingman, D. (1971). The more-for-less paradox in the distribution model. *Cahiers du Centre d'Etudes de Recherche Operationelle*, 13:11–22.

Cuturi, M. (2013). Sinkhorn distances: Lightspeed computation of optimal transport. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 2292–2300, USA. Curran Associates Inc.

Dantzig, G. B. (1951). Application of the simplex method to a transportation problem. *Activity Analysis and Production and Allocation*, pages 359–373.

Dantzig, G. B. (1963). *Linear Programming and Extensions*. Princeton University Press, Princeton.

Dantzig, G. B. (1990). Origins of the simplex method. In Nash, S. G., editor, *A History of Scientific Computing*, pages 141–151. ACM, New York, NY, USA.

Daum, S. and Werner, R. (2011). A novel feasible discretization method for linear semi-infinite programming applied to basket option pricing. *Optimization*, 60(10–11):1379–1398.

Deineko, V. G., Klinz, B., and Woeginger, G. J. (2003). Which matrices are immune against the transportation paradox? *Discrete Applied Mathematics*, 130(3):495–501.

Dennis, J. B. (1958). A high-speed computer technique for the transportation problem. *J. ACM*, 5(2):132–153.

Deza, M., Grishukhin, V. P., and Laurent, M. (1993). The hypermetric cone is polyhedral. *Combinatorica*, 13(4):397–411.

Deza, M. M. and Laurent, M. (1997). *Geometry of Cuts and Metrics*. Springer Publishing Company, Incorporated, 1st edition.

Embrechts, P., Hofert, M., and Wang, R. (2016). Bernoulli and tail-dependence compatibility. *Ann. Appl. Probab.*, 26(3):1636–1658.

Fiebig, U.-R., Strokorb, K., and Schlather, M. (2017). The realization problem for tail correlation functions. *Extremes*, 20(1):121–168.

Finke, G. (1978). A unified approach to reshipment, overshipment and post-optimization problems. In Stoer, J., editor, *Optimization Techniques*, pages 201–208, Berlin, Heidelberg. Springer Berlin Heidelberg.

Ford, L. R. and Fulkerson, D. R. (1956). Solving the transportation problem. *Management Science*, 3(1):24–32.

Ford, L. R. and Fulkerson, D. R. (1957a). A primal-dual algorithm for the capacitated hitchcock problem. *Naval Research Logistics Quarterly*, 4(1):47–54.

Ford, L. R. and Fulkerson, D. R. (1957b). A simple algorithm for finding maximal network flows and an application to the hitchcock problem. *Canadian Journal of Mathematics*, 9:210–218. Supplement: Report of the Washington Meeting.

Ford, L. R. and Fulkerson, D. R. (1962). *Flows in Networks*. Princeton University Press.

Frogner, C., Zhang, C., Mobahi, H., Araya-Polo, M., and Poggio, T. (2015). Learning with a wasserstein loss. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2053–2061, Cambridge, MA, USA. MIT Press.

Gass, S. I. (1969). *Linear programming: methods and applications / Saul I. Gass*. McGraw-Hill New York, 3rd ed. edition.

Gass, S. I. (1990). On solving the transportation problem. *Journal of the Operational Research Society*, 41(4):291–297.

Glicksman, S., Johnson, L., and Eselson, L. (1960). Coding the transportation problem. *Naval Research Logistics Quarterly*, 7(2):169–183.

Glover, F., Karney, D., and Klingman, D. (1972). The augmented predecessor index method for locating stepping-stone paths and assigning dual prices in distribution problems. *Transportation Science*, 6(2):171–179.

Glover, F., Karney, D., and Klingman, D. (1974a). Implementation and computational comparisons of primal, dual and primal-dual computer codes for minimum cost network flow problems. *Networks*, 4(3):191–212.

Glover, F., Karney, D., Klingman, D., and Napier, A. (1974b). A computation study on start procedures, basis change criteria, and solution algorithms for transportation problems. *Management Science*, 20(5):793–813.

Glover, F. and Klingman, D. (1972). Double-pricing dual and feasible start algorithms for the capacitated transportation (distribution) problem. *Applied Mathematics for Management*, AMM-20.

Glover, F. and Klingman, D. (1982). Recent developments in computer implementation technology for network flow algorithms. *INFOR: Information Systems and Operational Research*, 20(4):433–452.

Glover, F., Klingman, D., and Barr, R. (1979). Enhancements of spanning tree labelling procedures for network optimization. *Management Science*, 17:16–34.

Glover, F., Klingman, D., and Stutz, J. (1974c). Augmented threaded index method for network optimization. *INFOR: Information Systems and Operational Research*, 12(3):293–298.

Gottschlich, C. and Schuhmacher, D. (2014). The shortlist method for fast computation of the earth mover's distance and finding optimal solutions to transportation problems. *PLOS ONE*, 9(10):1–10.

Grigoriadis, M. D. (1986). An efficient implementation of the network simplex method. In Gallo, G. and Sandi, C., editors, *Netflow at Pisa*, pages 83–111. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hitchcock, F. L. (1941). The distribution of a product from several sources to numerous localities. *Journal of Mathematics and Physics*, 20(1–4):224–230.

Hoffman, A. J. (1963). On simple transportation problems. *Convexity, Proceedings of Symposia in Pure Mathematics*, 7:317–327.

Houthakker, H. S. (1955). On the numerical solution of the transportation problem. *Journal of the Operations Research Society of America*, 3(2):210–214.

Kaibel, V. and Weltge, S. (2015). A short proof that the extension complexity of the correlation polytope grows exponentially. *Discrete & Computational Geometry*, 53(2):397–401.

Kantorovich, L. V. (1960). Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422.

Kantorovitch, L. (1958). On the translocation of masses. *Management Science*, 5(1):1–4.

Kleinschmidt, P. and Schannath, H. (1995). A strongly polynomial algorithm for the transportation problem. *Mathematical Programming*, 68(1):1–13.

Klingman, D., Napier, A., and Stutz, J. (1974). Netgen: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20(5):814–821.

Kochenberger, G., Hao, J.-K., Glover, F., Lewis, M., Lü, Z., Wang, H., and Wang, Y. (2014). The unconstrained binary quadratic programming problem: a survey. *Journal of Combinatorial Optimization*, 28(1):58–81.

Kolouri, S., Park, S. R., Thorpe, M., Slepcev, D., and Rohde, G. K. (2017). Optimal mass transport: Signal processing and machine-learning applications. *IEEE Signal Processing Magazine*, 34(4):43–59.

Koopmans, T. C. (1949). Optimum utilization of the transportation system. *Econometrica (pre-1986)*, 17:136.

Korte, B. and Vygen, J. (2007). *Combinatorial Optimization: Theory and Algorithms*. Springer Publishing Company, Incorporated, 4th edition.

Kovács, P. (2015). Minimum-cost flow algorithms: an experimental evaluation. *Optimization Methods and Software*, 30(1):94–127.

Krause, D., Scherer, M., Schwinn, J., and Werner, R. (2018). Membership testing for bernoulli and tail-dependence matrices. *Journal of Multivariate Analysis*, 168:240–260.

Lee, A. J. (1993). Generating random binary deviates having fixed marginal distributions and specified degrees of association. *The American Statistician*, 47(3):209–215.

Lübbecke, M. (2010). *Column Generation*, volume 8 of *Wiley Encyclopedia of Operations Research and Management Science (EORMS)*. Wiley.

Luenberger, D. G. and Ye, Y. (2015). *Linear and Nonlinear Programming*. Springer Publishing Company, Incorporated.

Monge, G. (1781). Memoire sur la theorie des deblais et des remblais. *Histoire de l'Academie Royale des Sciences de Paris*.

Montavon, G., Müller, K.-R., and Cuturi, M. (2016). Wasserstein training of restricted boltzmann machines. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 3718–3726. Curran Associates, Inc.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38.

Orlin, J. (1988). A faster strongly polynomial minimum cost flow algorithm. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 377–387, New York, NY, USA. ACM.

Padberg, M. (1989). The boolean quadric polytope: Some characteristics, facets and relatives. *Mathematical Programming*, 45(1):139–172.

Pitowsky, I. (1991). Correlation polytopes: Their geometry and complexity. *Mathematical Programming*, 50(1):395–414.

Qaqish, B. F. (2003). A family of multivariate binary distributions for simulating correlated binary variables with specified marginal means and correlations. *Biometrika*, 90(2):455–463.

Rachev, S. T. and Rüschendorf, L. (1998). *Mass transportation problems*. Probability and its Applications A Series of the Applied Probability Trust. Springer, New York.

Reinfeld, N. V. and Vogel, W. R. W. R. (1958). *Mathematical programming*. Englewood Cliffs, N. J. : Prentice-Hall.

Ross, G. T., Klingman, D., and Napier, A. (1975). A computational study of the effects of problem dimensions on solution times for transportation problems. *J. ACM*, 22(3):413–424.

Rubner, Y., Guibas, L., and Tomasi, C. (1997). The earth mover's distance, multi-dimensional scaling, and color-based image retrieval. In *in Proceedings of the ARPA Image Understanding Workshop*, pages 661–668.

Russell, E. J. (1969). Extension of dantzig's algorithm to finding an initial near-optimal basis for the transportation problem. *Operations Research*, 17(1):187–191.

Sandi, C. (1986). On a nonbasic dual method for the transportation problem. In Gallo, G. and Sandi, C., editors, *Netflow at Pisa*, pages 65–82. Springer Berlin Heidelberg, Berlin, Heidelberg.

Schmitz, M. A., Heitz, M., Bonneel, N., Ngolè Mboula, F. M., Coeurjolly, D., Cuturi, M., Peyré, G., and Starck, J.-L. (2017). Wasserstein Dictionary Learning: Optimal Transport-based unsupervised non-linear dictionary learning. *ArXiv e-prints*.

Schrenk, S., Finke, G., and Cung, V.-D. (2011). Two classical transportation problems revisited: Pure constant fixed charges and the paradox. *Mathematical and Computer Modelling*, 54(9):2306–2315.

Schrieber, C., Schuhmacher, D., and Gottschlich, C. (2017). Dotmark – 2013; a benchmark for discrete optimal transport. *IEEE Access*, 5:271–282.

Schrijver, A. (2002). On the history of the transportation and maximum flow problems. *Mathematical Programming*, 91(3):437–445.

Schwinn, J. (2015). Transportprobleme (transportation problems). Master's thesis, University of Augsburg, Universitätsstraße 2, 86159 Augsburg.

Schwinn, J. and Werner, R. (2018). On the effectiveness of primal and dual heuristics for the transportation problem. *IMA Journal of Management Mathematics*, page dpy011.

Shor, N. Z., Kiwiel, K., and Ruszcayǹski, A. (1985). *Minimization Methods for Non-differentiable Functions*. Springer-Verlag New York, Inc., New York, NY, USA.

Srinivasan, V. and Thompson, G. (1973). Benefit-cost analysis of coding techniques for the primal transportation algorithm. *Journal of the ACM*, 20.

Srinivasan, V. and Thompson, G. L. (1972). Accelerated algorithms for labeling and relabeling of trees, with applications to distribution problems. *J. ACM*, 19(4):712–726.

Srivastava, S., Cevher, V., Dinh, Q., and Dunson, D. (2015). WASP: Scalable Bayes via barycenters of subset posteriors. In Lebanon, G. and Vishwanathan, S. V. N., editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 912–920, San Diego, California, USA. PMLR.

Staib, M., Claici, S., Solomon, J. M., and Jegelka, S. (2017). Parallel streaming wasserstein barycenters. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 2647–2658. Curran Associates, Inc.

Szwarc, W. (1971). The transportation paradox. *Naval Research Logistics Quarterly*, 18(2):185–202.

Tokuyama, T. and Nakano, J. (1995). Efficient algorithms for the Hitchcock transportation problem. *SIAM J. Comput.*, 24(3):563–578.

Villani, C. (2009). *Optimal Transport*. Grundlehren der mathematischen Wissenschaften. Springer-Verlag Berlin Heidelberg, Berlin.