Integration der Fahrplanoptimierung und Umlaufplanung

Bachelorarbeit in Mathematik
eingereicht an der Fakultät für Mathematik und Informatik
der Georg-August-Universität Göttingen
am 30. August 2013

von
Sven J. Jäger

Erstgutachterin:
Prof. Dr. Anita Schöbel

Zweitgutachterin:
Dr. Marie E. Schmidt

Inhaltsverzeichnis

1 Einleitung

Im Zuge knapper werdender Ressourcen und des globalen Anstiegs der Konzentration von Treibhausgasen in der Atmosphäre sind Maßnahmen zur Verringerung des Ressourcenverbrauchs und der Emission von Treibhausgasen gefragt. Der öffentliche Personenverkehr (Zug, Straßenbahn, Bus) kann einen Beitrag dazu leisten. In den Jahren von 2000 bis 2010 allerdings ist der Anteil der mit diesen umweltfreundlichen Verkehrsmitteln in der EU zurückgelegten Personenkilometern am gesamten motorisierten Verkehr leicht von 16,6% auf 15,7% zurückgegangen [UBA12]. Um den Anteil des öffentlichen Personenverkehrs zu erhöhen, ist es nötig, das Angebot der Verkehrsunternehmen so attraktiv wie möglich zu gestalten. Dies kann einerseits durch eine Reduzierung der Fahrpreise, andererseits durch eine Verbesserung des Angebots geschehen. Insbesondere die Fahr- und Umsteigezeiten sind ein wichtiger Faktor bei der Wahl des Verkehrsmittels. Diese so kurz wie möglich zu halten, ist das Ziel der Fahrplanoptimierung. Ressourcen können allerdings nicht nur dadurch gespart werden, dass der Anteil des öffentlichen Verkehrs gegenüber dem Individualverkehr gesteigert wird, sondern auch durch ihre effiziente Verwendung im öffentlichen Verkehr. Lange Leerfahrten sind allerdings nicht nur aus ökologischer Sicht, sondern auch aus wirtschaftlicher Sicht ungünstig. Die Umlaufplanung zielt darauf ab, die Fahrzeuge so effizient wie möglich einzusetzen.

Fahrplanoptimierung und Umlaufplanung sind Bestandteile der Planung eines Verkehrssystems. Weitere Schritte sind Netzwerk-Design, Linienplanung und Dienstplanung. Das Problem, ein optimales Verkehrssystem unter Einbeziehung all dieser Planungsschritte aufzustellen, ist so komplex, dass es nicht ganzheitlich gelöst werden kann. Stattdessen werden im klassischen Planungsprozess die Teilprobleme einzeln in der folgenden Reihenfolge gelöst:

1. Netzwerk-Design
2. Linienplanung
3. Fahrplanoptimierung
4. Umlaufplanung

Bedingt durch die Entwicklung leistungsfähiger Computer ist es heutzutage möglich, die oben beschriebenen Planungsprobleme rechnergestützt lösen zu lassen. Besonders hilfreich ist dafür die Formulierung der Probleme als ganzzahlige lineare Programme, da diese mithilfe existierender Software gelöst werden können. Klassischerweise wird das Ergebnis jedes Schrittes als Eingabe für den nächsten Schritt verwendet und kann dort nicht mehr

verändert werden. Die ganzzahligen linearen Programme der einzelnen Planungsschritte besitzen unterschiedliche Zielfunktionen. Zum Beispiel werden bei der Fahrplanoptimierung die Fahrzeiten der Kunden, bei der Umlaufplanung die Ausgaben des Unternehmens minimiert.

Ist f_1 die Zielfunktion eines Planungsschrittes und f_2 die im folgenden Schritt verwendete, so folgt aus $f_1(x_1) \leq f_1(x_2)$ für zwei zulässige Lösungen x_1, x_2 des ersten Planungsschrittes nicht, dass $f_2(g(x_1)) \leq f_2(g(x_2))$, wobei g jeder Eingabe des zweiten Planungsschrittes die zugehörige optimale Lösung zuordnet. Es ist also möglich, dass eine schlechtere Lösung des ersten Planungsschrittes zu einer besseren Lösung des folgenden Planungsschrittes führt.

Bei den in dieser Arbeit untersuchten Planungsschritten Fahrplanoptimierung und Umlaufplanung impliziert die Festlegung einer Reihenfolge eine Priorisierung der Ziele. In der oben genannten Reihenfolge haben beispielsweise kurze Reisezeiten erste Priorität: Zunächst wird ein aus Kundensicht optimaler Fahrplan berechnet und erst an zweiter Stelle dann ein Umlaufplan gesucht, der diesen mit möglichst wenigen Fahrzeugen oder möglichst geringen Betriebskosten umsetzt. Häufig ist eine solch strikte Priorisierung unerwünscht. Um diese zu umgehen, müssen die beiden Planungsprobleme in einem Schritt gelöst werden – man spricht dann von der *Integration* der beiden Planungsschritte. In dieser Arbeit wird das bikriterielle integrierte Fahrplan-Umlaufplan-Problem untersucht, bei welchem Fahr- und Umlaufplan gleichzeitig bestimmt und beide Zielfunktionen gleichwertig nebeneinanderstehen. Die allgemeine Formulierung als bikriterielles Programm kann je nach Bedarf wie in [Sch07] beschrieben auf unterschiedliche Art und Weise gelöst werden. Beispielsweise kann die gewichtete Summe der beiden Zielfunktionen minimiert werden, oder es wird ein Fahrplan und ein Umlaufplan bestimmt, so dass die Fahrzeiten der Kunden möglichst gering sind, wenn die Betriebskosten ein vorgegebenes Budget nicht überschreiten dürfen. Es stellt sich heraus, dass das integrierte Fahrplan-Umlaufplan NP-schwer ist. Folglich sind auch die Varianten, bei denen eine Zielfunktion beschränkt und die andere minimiert werden soll, NP-schwer.

Um eine Auswahl unterschiedlicher Paare von Fahrplänen und dazu passenden Umlaufplänen zu erhalten, wird in dieser Arbeit eine Heuristik entwickelt, mit der sich Lösungen berechnen lassen, die die Pareto-Optima approximieren. Diese Heuristik wird im Rahmen dieser Arbeit implementiert und in das Projekt LinTim integriert. Tests zeigen, dass die Heuristik zu Ergebnissen kommt, welche, wie erhofft, mit geringeren Betriebskosten als die im klassischen Planungsprozess erhaltenen auskommen, dafür aber längere Fahrzeiten für die Fahrgäste bedingen.

2 Grundlagen

2.1 Komplexitätstheorie

In dieser Arbeit soll unter anderem die Komplexität des integrierten Fahrplan-Umlaufplan-Problems untersucht werden. Besonders wichtig ist dabei die Frage, ob das Problem polynomiell lösbar ist. Dazu werden einige grundlegende Begriffe der Komplexitätstheorie eingeführt. Eine ausführliche Einführung in das Thema findet man in [CLRS10]. Den ganzen Abschnitt über seien Σ und Σ' zwei Alphabete und $\Sigma^*, (\Sigma')^*$ die zugehörigen Mengen an Wörtern.

Definition 2.1 (Polynomielle Laufzeit)**.** Ein Algorithmus $\mathcal{A}$ hat *polynomielle Laufzeit*, falls es $N, c, k \in \mathbb{N}$ gibt, so dass $\mathcal{A}$ für jede Eingabe der Länge $n \geq N$ höchstens $c \cdot n^k$ Zeiteinheiten benötigt.

Definition 2.2 (berechnen)**.** Ein Algorithmus $\mathcal{A}$ *berechnet* eine Funktion $f\colon \Sigma^* \to (\Sigma')^*$, falls er, angewendet auf ein beliebiges Wort $w \in \Sigma^*$, den Funktionswert $f(w)$ erzeugt. Hat $\mathcal{A}$ polynomielle Laufzeit, so sagt man, $\mathcal{A}$ berechnet f *in polynomieller Zeit.*

Definition 2.3 (in polynomieller Zeit berechenbar)**.** Eine Funktion $f\colon \Sigma^* \to (\Sigma')^*$ heißt *in polynomieller Zeit berechenbar*, falls es einen Algorithmus $\mathcal{A}$ gibt, der f in polynomieller Zeit berechnet.

Definition 2.4 (Entscheidungsproblem)**.** Ein Entscheidungsproblem (P) auf Σ^* ist gegeben durch eine Funktion

$$\chi_{(P)}\colon \Sigma^* \to \{0, 1\}.$$

Ein Wort $w \in \Sigma^*$ heißt in diesem Zusammenhang *Probleminstanz.*

Definition 2.5 (Entscheiden/Lösen eines Entscheidungsproblems)**.** Ein Algorithmus $\mathcal{A}$ *entscheidet* oder *löst* ein Entscheidungsproblem (P) (*in polynomieller Zeit*), falls er $\chi_{(P)}$ (in polynomieller Zeit) berechnet.

Definition 2.6 (in polynomieller Zeit lösbares Entscheidungsproblem)**.** Ein Entscheidungsproblem (P) heißt *in polynomieller Zeit lösbar*, falls es einen Algorithmus $\mathcal{A}$ gibt, der (P) in polynomieller Zeit löst. Die Menge aller in polynomieller Zeit lösbaren Entscheidungsprobleme wird mit $\mathbf{P}$ bezeichnet.

Definition 2.7 (Verifikationsalgorithmus)**.** Ein *Verifikationsalgorithmus* $\mathcal{A}$ erhält neben der Probleminstanz $w \in \Sigma^*$ eine weitere als *Zertifikat* bezeichnete Eingabe $z \in (\Sigma')^*$. $\mathcal{A}$ *verifiziert* eine Probleminstanz w, falls es ein Zertifikat $z \in (\Sigma')^*$ gibt, so dass $\mathcal{A}$ angewendet auf w und z das Ergebnis 1 ausgibt.

Definition 2.8 (verifizieren)**.** Ein Verifikationsalgorithmus $\mathcal{A}$ *verifiziert* ein Entscheidungsproblem (P), falls für $w \in \Sigma^*$ gilt:

$$\mathcal{A} \text{ verifiziert } w \iff \chi_{(P)}(w) = 1.$$

Hat $\mathcal{A}$ polynomielle Laufzeit und gibt es $N, c, k \in \mathbb{N}$, so dass es für jede verifizierte Probleminstanz der Länge $n \geq N$ ein Zertifikat z mit $|z| \leq c \cdot n^k$ gibt, welches zum Ergebnis 1 führt, so verifiziert $\mathcal{A}$ das Problem (P) *in polynomieller Zeit.*

Definition 2.9 (in polynomieller Zeit verifizierbar)**.** Ein Entscheidungsproblem (P) heißt *in polynomieller Zeit verifizierbar*, falls es einen Verifikationsalgorithmus $\mathcal{A}$ gibt, der (P) in polynomieller Zeit verifiziert. Die Menge aller in polynomieller Zeit verifizierbaren Entscheidungsprobleme wird mit **NP** bezeichnet.

Beispiel. Das Partitionsproblem besteht darin, für eine gegebene Familie $(n_i)_{i \in I}$ natürlicher Zahlen zu entscheiden, ob es eine Teilfamilie I' gibt, so dass

$$\sum_{i \in I'} n_i = \sum_{i \in I \setminus I'} n_i.$$

Als Zertifikat dient hier die Angabe von I': Es gilt $|I'| \leq |I|$ und es kann in polynomieller Zeit überprüft werden, ob I' obige Gleichung erfüllt. Also ist das Partitionsproblem in polynomieller Zeit verifizierbar.

Offenbar gilt $\mathbf{P} \subseteq \mathbf{NP}$, d. h. jedes in polynomieller Zeit lösbare Problem ist auch in polynomieller Zeit verifizierbar. Bis heute ist ungeklärt, ob die Teilmengenbeziehung echt ist. Dieses Problem wird als *P-NP-Problem* bezeichnet. Einen Hinweis darauf geben die NP-vollständigen Probleme, die im Folgenden formal definiert werden.

Definition 2.10 (Polynomialzeitreduktion)**.** Ein Entscheidungsproblem (P) auf Σ^* ist *polynomiell reduzierbar* auf ein Entscheidungsproblem (P') auf $(\Sigma')^*$, wenn es eine in polynomieller Zeit berechenbare Funktion $f \colon \Sigma^* \to (\Sigma')^*$ gibt, so dass für alle $w \in \Sigma^*$ gilt:

$$\chi_{(P)}(w) = \chi_{(P')}(f(w)).$$

In diesem Fall schreiben wir $(P) \leq_{\mathrm{P}} (P')$.

Bemerkung 2.11. $\leq_{\mathrm{P}}$ ist eine Quasiordnung.

Lemma 2.12 ([CLRS10, Lemma 34.3])**.** *Seien* $(P), (P')$ *Entscheidungsprobleme mit* $(P) \leq_{\mathrm{P}} (P')$*. Dann folgt aus* $(P') \in \mathbf{P}$*, dass auch* $(P) \in \mathbf{P}$*.*

Definition 2.13 (NP-schwer, NP-vollständig)**.** Ein Entscheidungsproblem (P) heißt *NP-schwer*, wenn $(P') \leq_{\mathrm{P}} (P)$ für alle $(P') \in \mathbf{NP}$. Gilt außerdem $(P) \in \mathbf{NP}$, so heißt (P) *NP-vollständig*. Die Menge aller NP-schweren Entscheidungsprobleme wird mit **NPH** und die Menge aller NP-vollständigen Entscheidungsprobleme mit **NPC** bezeichnet.

Die NP-vollständigen Probleme sind also die schwierigsten Probleme innerhalb der Klasse **NP**. Dies hat folgende Konsequenz:

Satz 2.14 ([CLRS10, Theorem 34.4])**.** *Wenn ein NP-vollständiges Problem (P) in polynomieller Zeit gelöst werden kann, dann gilt:* $\mathbf{P} = \mathbf{NP}$.

Beweis. Sei $(P') \in \mathbf{NP}$. Dann gilt nach Definition von NP-vollständig, dass $(P') \leq_{\mathrm{P}} (P)$. Also folgt aus Lemma 2.12 $(P') \in \mathbf{P}$. Somit erhalten wir die fehlende Inklusion $\mathbf{NP} \subseteq \mathbf{P}$. □

Eigentlich soll für die in dieser Arbeit untersuchten Probleme entschieden werden, ob diese polynomiell lösbar sind oder nicht. Einige der Probleme sind allerdings NP-vollständig. Für diese kann die Frage nicht beantwortet werden, ohne das P-NP-Problem zu lösen. Da bisher viele NP-vollständige Probleme untersucht wurden und für keines von ihnen ein Polynomialzeitalgorithmus gefunden wurde, geht ein Großteil der Fachwelt davon aus, dass $\mathbf{P} \neq \mathbf{NP}$. In diesem Fall würde nach obigem Satz der Beweis, dass ein Problem NP-vollständig ist, genügen, um zu zeigen, dass man es nicht in polynomieller Zeit lösen kann. Aus diesem Grund werden wir für jedes im Laufe dieser Arbeit betrachtete Problem zeigen, dass es polynomiell lösbar oder NP-vollständig ist.

Um zu beweisen, dass ein Problem (P) NP-vollständig ist, müssen wir zeigen, dass es in polynomieller Zeit verifizierbar und NP-schwer ist. Letzteres tun wir, indem wir ein Problem, von dem bekannt ist, dass es NP-schwer ist, auf (P) reduzieren. Dies reicht nach dem folgenden Lemma aus.

Lemma 2.15 ([CLRS10, Lemma 34.8])**.** *Seien $(P), (P')$ Entscheidungsprobleme mit $(P') \leq_{\mathrm{P}} (P)$. Dann folgt aus $(P') \in \mathbf{NPH}$, dass auch $(P) \in \mathbf{NPH}$.*

Beweis. Sei $(P'') \in \mathbf{NP}$. Dann gilt nach Definition von NP-schwer, dass $(P'') \leq_{\mathrm{P}} (P')$. Da $\leq_{\mathrm{P}}$ transitiv ist, gilt also auch $(P'') \leq_{\mathrm{P}} (P)$. □

Nachdem Stephen A. Cook 1971 bewies, dass das Erfüllbarkeitsproblem der Aussagenlogik NP-vollständig ist [Coo71], verwendete Richard Karp 1972 die oben beschriebene Beweistechnik, um für 21 äußerlich sehr unterschiedlich aussehende Probleme zu zeigen, dass diese ebenfalls NP-vollständig sind [Kar72]. Er regte damit die weitere Erforschung der Klasse **NP** an und ermöglichte Reduktionsbeweise ausgehend von irgendeinem dieser 21 Probleme. Heute ist für sehr viele Probleme bekannt, dass sie NP-vollständig sind; das Werk [GJ79] enthält ein umfangreiches Kompendium.

Definition 2.16 (Optimierungsproblem)**.** Ein *Optimierungsproblem* (P) besteht aus

- der Information, ob die Zielfunktion maximiert oder minimiert werden soll,
- einer Menge $\mathcal{I}$ zulässiger Eingaben, den so genannten *Instanzen*,

sowie zu jeder Instanz $P \in \mathcal{I}$ aus

- einer Menge $\mathcal{F}_P$ von *zulässigen Lösungen*,
- einer so genannten *Zielfunktion* $f_P \colon \mathcal{F}_P \to \mathbb{R}$.

Ziel ist es, eine zulässige Lösung $x \in \mathcal{F}_P$ zu finden mit maximalem bzw. minimalem Zielfunktionswert $f_P(x)$.

Definition 2.17 (Lösen eines Optimierungsproblems)**.** Ein Algorithmus $\mathcal{A}$ *löst* ein Minimierungsproblem (P), falls er für eine beliebige Instanz P ein $x \in \mathcal{F}_P$ erzeugt, so dass $f_P(x) \leq f_P(y)$ für alle $y \in \mathcal{F}_P$. Für Maximierungsprobleme wird stattdessen $f_P(x) \geq f_P(y)$ für alle $y \in \mathcal{F}_P$ gefordert. Hat $\mathcal{A}$ polynomielle Laufzeit, so sagt man, $\mathcal{A}$ löst (P) *in polynomieller Zeit*.

Definition 2.18 (in polynomieller Zeit lösbares Optimierungsproblem)**.** Ein Optimierungsproblem (P) heißt *in polynomieller Zeit lösbar*, falls es einen Algorithmus $\mathcal{A}$ gibt, der (P) in polynomieller Zeit löst.

Definition 2.19 (Entscheidungsversion)**.** Sei (P) ein Minimierungsproblem. Dann heißt das Entscheidungsproblem $(P_{\leq})$, dessen Instanzen Paare (P, k) sind, wobei P eine Instanz von (P) und $k \in \mathbb{R}$ ist, mit

$$\chi_{(P_{\leq})}(P, k) = \begin{cases} 1, & \text{falls } \exists\, x \in \mathcal{F}_P : f_P(x) \leq k, \\ 0, & \text{sonst.} \end{cases}$$

die *Entscheidungsversion* von (P). Entsprechend ist die Entscheidungsversion $(P_{\geq})$ eines Maximierungsproblems definiert.

In obiger Definition ist $(P_{\leq})$ „nicht schwerer“ als (P), denn man kann $(P_{\leq})$ lösen, indem man (P) löst und den Wert des Minimums mit k vergleicht. Selbst wenn $(P_{\leq})$ polynomiell verifizierbar ist, gilt Entsprechendes nicht zwangsläufig für (P). Dies erklärt folgende Definition.

Definition 2.20 (NP-schweres Optimierungsproblem, [NSW11])**.** Ein Optimierungsproblem heißt *NP-schwer*, wenn das zugehörige Entscheidungsproblem NP-vollständig ist.

2.2 Graphen und Netzwerke

Graphen wurden von Leonhard Euler im Rahmen der Formulierung des berühmten Königsberger Brückenproblem zur Abstraktion von geographischen Gegebenheiten eingeführt. Allerdings lassen sich mit Graphen viel allgemeiner beliebige binäre Relationen darstellen. Im Laufe des letzten Jahrhunderts haben Graphen dank ihrer Vielseitigkeit und anschaulichen Darstellung Einzug in zahlreiche Bereiche – von der Wirtschaftswissenschaft bis hin zur theoretischen Informatik – gehalten. Auch in der Verkehrsoptimierung lassen sich die meisten Probleme mithilfe von Graphen modellieren. Dabei ist der Zusammenhang zwischen Graphen und geographischen Gegebenheiten in den unterschiedlichen Modellen noch mehr oder weniger vertreten. Da die Terminologie in der Graphentheorie nicht ganz einheitlich verwendet wird, werden hier einige Definitionen von Begriffen zusammengestellt, so wie sie in dieser Arbeit verwendet werden.

Definition 2.21 (Graph)**.** Ein Graph $G = (V, E)$ besteht aus einer nicht leeren, endlichen Menge V von *Knoten* und einer Menge E von *Kanten*, wobei

- $E \subseteq \{\{i, j\} \mid i, j \in V,\ i \neq j\}$ für einen *ungerichteten Graphen* G, bzw.
- $E \subseteq V^2$ für einen *gerichteten Graphen* G.

Diese Definition schließt aus, dass es mehrere Kanten zwischen zwei Knoten gibt, die in die gleiche Richtung zeigen.

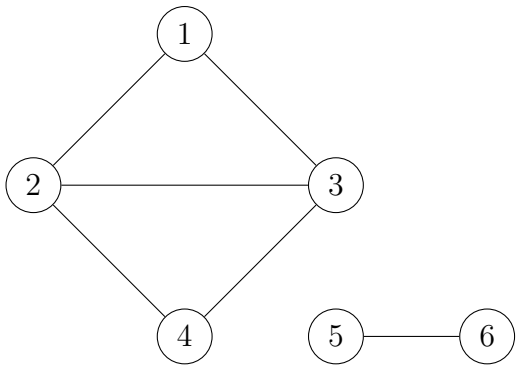

Abb. 2.1: Beispiel für einen ungerichteten Graphen

Definition 2.22 (Teilgraph)**.** Ein *Teilgraph* $G' = (V', E')$ eines Graphen $G = (V, E)$ ist ein Graph mit $V' \subseteq V$ und $E' \subseteq E$. Gilt $V' = V$, so heißt G' ein *spannender Teilgraph* von G.

Da für alle in dieser Arbeit betrachteten Probleme jeder ungerichtete Graph durch einen gerichteten Graphen ersetzt werden kann, der für jede Kante $\{i, j\}$ des ungerichteten Graphen die beiden Kanten (i, j) und (j, i) enthält, betrachten wir im Folgenden nur noch gerichtete Graphen.

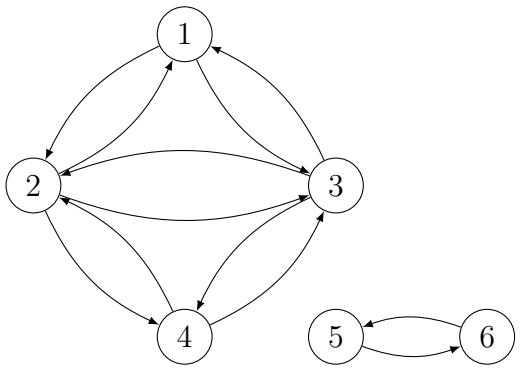

Abb. 2.2: Gerichtete Version des obigen ungerichteten Graphen

Definition 2.23 (Weg)**.** Ein *Weg* in einem Graphen $G = (V, E)$ ist eine Folge $W = (v_0, e_1, v_1, \ldots, e_l, v_l)$ mit $v_i \in V,\ i = 0, \ldots, l$ und $e_i \in E,\ i = 1, \ldots, l$, so dass

$$e_i \in \{(v_{i-1}, v_i), (v_i, v_{i-1})\}\ \forall\, i \in \{1, \ldots, l\}.$$

v_0 heißt *Start-* und v_l *Endknoten* von W. Die Menge aller Kanten im Weg W wird mit $E(W)$, die Menge aller Knoten mit $V(W)$ bezeichnet. Weiter werden folgende Mengen definiert:

$$W^+ := \{e_i \in E(W) \mid e_i = (v_{i-1}, v_i)\},$$
$$W^- := \{e_i \in E(W) \mid e_i = (v_i, v_{i-1})\}.$$

- Gilt $W^- = \emptyset$, so heißt W *gerichteter Weg.*
- Gilt $v_0 = v_l$, so heißt W *Zyklus.*
- Gilt $v_i \neq v_j$ für alle $i \neq j$, so heißt W *einfacher Weg* oder *Pfad.*
- Gilt $v_0 = v_l$ und $v_i \neq v_j$ für alle anderen Paare $i \neq j$, so heißt W *einfacher Zyklus* oder *Kreis.*

Definition 2.24 (zusammenhängend)**.** Ein Graph $G = (V, E)$ heißt *zusammenhängend*, wenn es zwischen jedem Paar von Knoten $u, v \in V$ einen Weg W mit Startknoten u und Endknoten v gibt.

Definition 2.25 (Zusammenhangskomponente)**.** Ein Teilgraph $G' = (V', E')$ eines Graphen $G = (V, E)$ heißt *Zusammenhangskomponente* von G, falls folgende Bedingungen erfüllt sind:

(i) G' ist zusammenhängend.

(ii) G' ist maximal mit dieser Eigenschaft, d. h. es gibt keinen zusammenhängenden Teilgraphen $G'' \neq G'$ von G, der G' als Teilgraph enthält.

Definition 2.26 (Knoten-Kanten-Inzidenzmatrix)**.** Sei $G = (V, E)$ ein gerichteter Graph mit $V = \{v_1, \dots, v_n\}$ und $E = \{e_1, \dots, e_m\}$. Dann heißt die Matrix $A \in \{-1, 0, 1\}^{n \times m}$ mit

$$a_{ij} = \begin{cases} 1, & \text{falls } e_j = (v_i, v_k) \text{ für ein } v_k \in V \\ -1, & \text{falls } e_j = (v_k, v_i) \text{ für ein } v_k \in V \\ 0, & \text{sonst} \end{cases}$$

die *Knoten-Kanten-Inzidenzmatrix* von G.

Die Einträge der Knoten-Kanten-Inzidenzmatrix werden häufig mit den Knoten und den Kanten selbst indiziert, was unübersichtliche Indizes erspart. Diese Schreibweise wird im Folgenden häufiger verwendet, weshalb sie an dieser Stelle formal eingeführt wird.

Notation 2.27 (Familie)**.** Seien B, I zwei Mengen. Eine *Familie* $(b_i)_{i \in I}$ von Elementen aus B mit Indexmenge I ist eine Abbildung $f \colon I \to A$, wobei $f(i) =: b_i$ geschrieben wird. Die Menge aller mit I indizierten Familien von Elementen aus B wird mit B^I bezeichnet.

Die Knoten-Kanten-Inzidenzmatrix lässt sich damit als Familie $A \in \{-1, 0, 1\}^{V \times E}$ auffassen. Da die in dieser Arbeit vorkommenden Indexmengen allesamt endlich sind, können wir stets annehmen, dass die Elemente durchnummeriert werden können, was uns ermöglicht, die zugehörigen Familien als gewöhnliche Vektoren und Matrizen aufzufassen und die dafür üblichen Schreibweisen, wie z. B. die Matrix-Vektor-Multiplikation, zu verwenden.

Definition 2.28 (Länge)**.** Sei $G = (V, E)$ ein gerichteter Graph, $x \in \mathbb{R}^E$ und $W = (v_0, e_1, \dots, e_l, v_l)$ ein Weg in G. Die *Länge von W bezüglich x* ist

$$\text{length}_x(W) := \sum_{e \in E(W)} x_e.$$

Definition 2.29 (Kürzester Weg)**.** Sei $G = (V, E)$ ein gerichteter Graph und $x \in \mathbb{R}^E$. Ein Weg $W = (v_0, e_1, \dots, e_l, v_l)$ heißt *kürzester (gerichteter) Weg* von v_0 nach v_l (bzgl. x), falls er unter allen (gerichteten) Wegen von v_0 nach v_l minimale Länge bzgl. x hat.

Bemerkung 2.30. Ist G zusammenhängend und gibt es in G keine (gerichteten) Kreise negativer Länge, so gibt es für jedes Paar u, v von Knoten einen kürzesten (gerichteten) Weg von u nach v.

Das Optimierungsproblem, zu einem gegebenen Graphen G ohne Kreise negativer Länge, einer Kantengewichtung x sowie zwei Knoten u und v den kürzesten (gerichteten) Weg von u nach v zu finden, kann mit dem Bellman-Ford-Algorithmus in Zeit $\mathcal{O}(|V| \cdot |E|)$ gelöst werden [CLRS10]. Der Bellman-Ford-Algorithmus berechnet sogar die kürzesten Wege von u zu jedem anderen Knoten in G. Will man kürzeste Wege für jedes Paar von Knoten berechnen, so kann man den Bellman-Ford-Algorithmus für jeden Knoten durchführen,

was zu einer Gesamtlaufzeit von $\mathcal{O}(|V|^2 \cdot |E|)$ führt. Effizienter lässt sich letzteres Problem mit dem Floyd-Warshall-Algorithmus lösen, der in Zeit $\mathcal{O}(|V|^3)$ läuft [CLRS10].

Definition 2.31 (Fluss, Zirkulation)**.** Sei $G = (V, E)$ ein gerichteter Graph und sei $b \in \mathbb{R}^V$. Eine Kantengewichtung $x \in \mathbb{R}^E$ heißt *Fluss* zu b, falls

$$\sum_{j:(i,j)\in E} x_{(i,j)} - \sum_{j:(j,i)\in E} x_{(j,i)} = b_i \quad \forall\, i \in V.$$

Im Falle $b_v = 0$ für alle $v \in V$ heißt x *Zirkulation.*

Das Netzwerkflussproblem (NFP) besteht darin, zu einem gegebenen Graphen, einem so genannten Bedarfsvektor, unteren und oberen Schranken an den Fluss über jede Kante und Kosten für jede Kante, einen zulässigen Fluss mit minimalen Kosten zu bestimmen. Formal ist das Problem folgendermaßen definiert:

Gegeben sind ein gerichteter Graph $G = (V, E)$, $b \in \mathbb{R}^V$, $l, u \in \mathbb{R}^E$ und $c \in \mathbb{R}^E$.

Gesucht ist ein Fluss $x \in \mathbb{R}^E$ zu b, so dass $l_{(i,j)} \leq x_{(i,j)} \leq u_{(i,j)}$ für alle $(i, j) \in E$ und so dass $\sum_{(i,j)\in E} c_{(i,j)} x_{(i,j)}$ minimal ist.

Dieses Problem wird im folgenden Abschnitt weiter untersucht.

2.3 Grundlagen der Optimierung

Die in dieser Arbeit untersuchten Probleme lassen sich als ganzzahlige lineare Programme darstellen, deren Grundlage lineare Programme sind. In diesem Abschnitt werden beide eingeführt sowie allgemeine Aussagen zu ihrer Komplexität getroffen. Der Abschnitt orientiert sich an dem Skript [Sch12] und dem Lehrbuch [NSW11]. Da das betrachtete integrierte Fahrplan-Umlaufplan-Problem zwei unterschiedliche Zielfunktionen hat, wird am Ende dieses Abschnitts kurz auf bikriterielle Optimierung eingegangen.

2.3.1 Lineare Optimierung

Definition 2.32 (Lineares Programm)**.** Ein *lineares Programm* mit Variablen $x_1, \ldots, x_n$ ist gegeben durch

- eine lineare *Zielfunktion* $f\colon \mathbb{R}^n \to \mathbb{R}$,
- eine endliche Menge von linearen Nebenbedingungen der Form $g(x_1, \ldots, x_n) = b$, $g(x_1, \ldots, x_n) \leq b$ oder $g(x_1, \ldots, x_n) \geq b$, wobei g eine lineare Funktion und $b \in \mathbb{R}$ ist,
- sowie die Information, ob f zu minimieren oder zu maximieren ist.

Definition 2.33 (zulässig, optimal)**.** Der *zulässige Bereich* $\mathcal{F}$ ist die Menge aller Punkte $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$, welche alle Nebenbedingungen erfüllen. Jeder Punkt a im zulässigen Bereich heißt *zulässige Lösung*. Gilt außerdem $f(a) \leq f(b)$ für alle zulässigen Lösungen b im Fall eines Minimierungsproblems, bzw. $f(a) \geq f(b)$ im Fall eines Maximierungsproblems, so heißt a *Optimallösung*.

Definition 2.34 ($\leq$-Form, Standardform)**.** Ein lineares Programm heißt

- *in $\leq$-Form*, falls es sich um ein Minimierungsproblem handelt und alle Nebenbedingungen die Form $g(x_1, \ldots, x_n) \leq b$ haben.
- *in Standardform*, falls es sich um ein Minimierungsproblem handelt, welches für alle $i \in \{1, \ldots, n\}$ die Bedingung $x_i \geq 0$ sowie weitere Bedingungen der Form $g(x_1, \ldots, x_n) = b$ enthält.

Mit der Matrix-Vektor-Schreibweise lässt sich ein lineares Programm in $\leq$-Form schreiben als

$$\begin{aligned} &\min c^T x \\ &\text{so dass } Ax \leq b \end{aligned}$$

und ein lineares Programm in Standardform als

$$\begin{aligned} &\min c^T x \\ &\text{so dass } Ax = b \\ &\qquad\quad x \geq 0, \end{aligned} \tag{2.1}$$

wobei $c \in \mathbb{R}^n$ der Zielfunktionsvektor, $A \in \mathbb{R}^{m \times n}$ die Koeffizientenmatrix und $b \in \mathbb{R}^m$ ist.

Jedes lineare Programm lässt sich durch elementare Umformungen in ein äquivalentes lineares Programm in $\leq$-Form bzw. in Standardform überführen.

Definition 2.35 (Basislösung)**.** Sei $A \in \mathbb{R}^{m \times n},\ b \in \mathbb{R}^m$ mit $n \geq m = \operatorname{rg}(A)$. Ein Punkt $x \in \mathbb{R}^n$ heißt *Basislösung* des Systems $Ax = b$, wenn es eine Partition $B \dot\cup N = \{1, \ldots, n\}$ der Spaltenindizes gibt, so dass

(i) die Spalten $A_j,\ j \in B$ eine Basis des $\mathbb{R}^m$ bilden,

(ii) $x_N = 0$,

(iii) $x_B = A_B^{-1} b$.

Satz 2.36 (Hauptsatz der linearen Programmierung)**.** *In der Situation des obigen linearen Programms (2.1) sei $n \geq m$ und $\operatorname{rg}(A) = m$. Gibt es eine Optimallösung von (2.1), dann gibt es auch eine Optimallösung, welche eine Basislösung von $Ax = b$ ist. Diese heißt* optimale Basislösung *von (2.1).*

Satz 2.37 (Khachiyan, [Sch98, Theorem 13.4])**.** *Lineare Programme (mit rationalen Koeffizienten) können in polynomieller Zeit gelöst werden.*

Bemerkung 2.38. Mit dem Innere-Punkte-Verfahren kann sogar in polynomieller Zeit eine optimale Basislösung gefunden werden.

2.3.2 Ganzzahlige lineare Optimierung

Definition 2.39 (Ganzzahliges lineares Programm)**.** Ein *ganzzahliges lineares Programm* oder *IP* mit den Variablen $x_1, \ldots, x_n$ ist ein lineares Programm mit diesen Variablen mit der zusätzlichen Nebenbedingung $(x_1, \ldots, x_n) \in \mathbb{Z}^n$.

Ganzzahlige lineare Programme lassen sich als Instanzen eines Optimierungsproblems im Sinne von Definition 2.16 auffassen. Dieses bezeichnen wir mit (INTEGER PROGRAMMING). Eine Instanz IP besteht aus $c \in \mathbb{R}^n$, $A_1 \in \mathbb{R}^{m_1 \times n}$, $A_2 \in \mathbb{R}^{m_2 \times n}$, $A_3 \in \mathbb{R}^{m_3 \times n}$, $b_1 \in \mathbb{R}^{m_1}$, $b_2 \in \mathbb{R}^{m^2}$ und $b \in \mathbb{R}^{m_3}$ für $n, m_1, m_2, m_3 \in \mathbb{N}$. Der zugehörige zulässige Bereich ist $\mathcal{F}_{IP} := \{x \in \mathbb{Z}^n \mid A_1 x = b_1,\ A_2 x \leq b_2,\ A_3 x \geq b_3\}$ und die zugehörige Zielfunktion ist gegeben durch $f_{IP}(x) := c^T x$.

Satz 2.40. *Das Optimierungsproblem* (INTEGER PROGRAMMING) *ist NP-schwer.*

Definition 2.41 (Relaxation)**.** Sei P eine Instanz eines Optimierungsproblems mit zulässigem Bereich $\mathcal{F}_P$ und Zielfunktion f_P. Ein Programm P' mit zulässigem Bereich $\mathcal{F}_{P'}$ und Zielfunktion $f_{P'}$ heißt *Relaxation* von P, falls $\mathcal{F}_P \subseteq \mathcal{F}_{P'}$ und $f_{P'}(x) \leq f_P(x)$ für alle $x \in \mathcal{F}_P$.

Lemma 2.42. *Sei P ein Optimierungsproblem und P' eine Relaxation davon. Sei x^* eine Optimallösung von P und x' eine Optimallösung von P'.*

1. *Dann gilt $f_{P'}(x') \leq f_P(x^*)$.*
2. *Ist $x' \in \mathcal{F}_P$, so ist x' optimal für P.*

Definition 2.43 (LP-Relaxation)**.** Ist IP ein ganzzahliges lineares Programm, so heißt das lineare Programm, welches entsteht, wenn man ausschließlich die Ganzzahligkeitsbedingung weglässt, die *LP-Relaxation* von IP.

Definition 2.44 (total unimodular)**.** Eine Matrix $A \in \mathbb{R}^{m \times n}$ heißt *total unimodular*, falls für jede Submatrix B von A gilt:

$$\det(B) \in \{-1, 0, 1\}.$$

Satz 2.45. *Sei $A \in \mathbb{R}^{m\times n}$ total unimodular und $b \in \mathbb{Z}^m$. Dann ist jede Basislösung von $Ax = b$ ganzzahlig.*

Lemma 2.46. *Ist $A \in \mathbb{R}^{m\times n}$ total unimodular, so sind auch die Matrizen $A^T, (A|e_i)$, $(A| - e_i)$ für $i \in \{1, \dots, m\}$ und $(A| - A_j)$ für $j \in \{1, \dots, n\}$ total unimodular.*

Beweis. Dass das Hinzufügen von Spalten der Form e_i die totale Unimodularität nicht ändert, wird in [Sch12] gezeigt. Das gleiche gilt für Spalten der Form $-e_i$, da $\det(A| - e_i) = -\det(A|e_i)$. Sei nun B eine Submatrix von $(A| - A_j)$. Ist B eine Submatrix von A, so gilt $\det(B) \in \{-1, 0, 1\}$. Anderenfalls enthält B die Spalte $-A_j$. Enthält B außerdem die Spalte A_j, so ist $\det(B) = 0$. Ansonsten ist $\det(B) = \pm\det(B')$, wobei B' die Matrix ist, die entsteht, wenn $-A_j$ aus B entfernt und A_j hinzufügt wird. □

Wir betrachten nun den Spezialfall (TU-INTEGER PROGRAMMING) von (INTEGER PROGRAMMING), in dem die Koeffizientenmatrix A total unimodular und b ganzzahlig ist.

Korollar 2.47. *Das Optimierungsproblem* (TU-INTEGER PROGRAMMING) *ist polynomiell lösbar.*

Beweis. Sei IP die Probleminstanz mit total unimodularer Matrix $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{R}^n$. A lässt sich durch Hinzufügen von Spalten der Form $e_i, -e_i,\ i \in \{1, \dots, m\}$,
$-A_j,\ j \in \{1, \dots, n\}$ in die Koeffizientenmatrix A' der zugehörigen Standardform umwandeln. Somit ist A' nach Lemma 2.46 ebenfalls total unimodular. Nach Satz 2.45 sind alle Basislösungen von $A'x = b$ ganzzahlig, also gibt es nach Satz 2.36 eine ganzzahlige optimale Lösung der LP-Relaxation, welche nach Bemerkung 2.38 in polynomieller Zeit gefunden werden kann. Diese ist nach Lemma 2.42 auch optimal für IP. □

Es folgen einige Bedingungen, die die totale Unimodularität einer Matrix gewährleisten.

Satz 2.48. *Sei $A \in \mathbb{R}^{m\times n}$. A ist total unimodular, falls die folgenden drei Bedingungen erfüllt sind:*

(i) $a_{ij} \in \{-1, 0, 1\}$ *für alle* $i \in \{1, \dots, m\},\ j \in \{1, \dots, n\}$.

(ii) Jede Spalte von A enthält höchstens zwei von 0 verschiedene Elemente.

(iii) Es gibt eine Partition $I_1 \dot\cup I_2 = \{1, \dots, m\}$ der Menge der Zeilen, so dass für jede Spalte j mit zwei von 0 verschiedenen Einträgen gilt:

$$\sum_{i\in I_1} a_{ij} = \sum_{i\in I_2} a_{ij}.$$

Korollar 2.49. *Knoten-Kanten-Inzidenzmatrizen von gerichteten Graphen sind total unimodular.*

Beweis. Die ersten beiden Bedingungen des obigen Satzes sind offensichtlich erfüllt. Für die dritte Bedingung ist $I_1 := \{1, \ldots, |V|\}$ und $I_2 := \emptyset$ zu wählen. □

Korollar 2.50. *Das in Abschnitt 2.2 definierte Netzwerkflussproblem kann in polynomieller Zeit gelöst werden und hat für ganzzahlige Parameter b, l, u stets eine ganzzahlige Optimallösung.*

Beweis. Eine Probleminstanz (V, E, b, l, u, c) des Netzwerkflussproblems lässt sich folgendermaßen als ganzzahliges lineares Programm formulieren, wobei A die Knoten-Kanten-Inzidenzmatrix von (V, E) ist

$$\begin{aligned} &\min\ c^T x \\ &\text{s. d. } Ax = b \\ &\qquad l \leq x \leq u \end{aligned}$$

Die Koeffizientenmatrix ist

$$\begin{pmatrix} A \\ I \\ -I \end{pmatrix}.$$

A ist nach Korollar 2.49 total unimodular. Durch mehrfache Anwendung von Lemma 2.46 erhält man die totale Unimodularität der Koeffizientenmatrix. □

2.3.3 Bikriterielle Optimierung

Die bikriterielle Optimierung untersucht Probleme mit zwei Zielfunktionen, die sich teilweise widersprechen können. Sie ist ein Spezialfall der multikriteriellen Optimierung

Definition 2.51 (Multikriterielles Minimierungsproblem)**.** Ein *multikriterielles Minimierungsproblem* ist definiert durch

- eine *zulässige Menge* $X \subset \mathbb{R}^n$,
- einen *Zielraum* $\mathbb{R}^p$ mit einer Ordnungsrelation $\prec$
- einen *Zielfunktionsvektor* $f = (f_1, \ldots, f_p)$, der X in den Raum $\mathbb{R}^p$ abbildet.

Wir schreiben dafür

$$\min\nolimits_{x \in X}^{\prec}(f_1(x), f_2(x)).$$

Ein bikriterielles Minimierungsproblem (BOP) ist ein multikriterielles Minimierungsproblem mit $p = 2$. Im Folgenden wird nur dieser Fall betrachtet.

Definition 2.52. Jedes Element $x \in X$ heißt *zulässige Lösung*. x heißt *optimal* (bzgl. $\prec$), wenn es kein $y \in X$ mit $y \neq x$ gibt, so dass $f(y) \prec f(x)$.

Es kommen unterschiedliche Ordnungsrelationen auf $\mathbb{R}^2$ in Frage, hinsichtlich derer die Zielfunktionen minimiert werden können:

$$\begin{aligned}
(x_1, x_2) &\leq (y_1, y_2) && :\Leftrightarrow x_1 \leq y_1 \wedge y_1 \leq y_2 \\
(x_1, x_2) &< (y_1, y_2) && :\Leftrightarrow (x_1 \leq y_1 \wedge x_2 \leq y_2) \text{ und } (x_1 < y_1 \vee x_2 < y_2) \\
(x_1, x_2) &\ll (y_1, y_2) && :\Leftrightarrow x_1 < y_1 \wedge x_2 < y_2 \\
(x_1, x_2) &\leq_{\text{lex}} (y_1, y_2) && :\Leftrightarrow x_1 < y_1 \text{ oder } (x_1 = y_1 \wedge x_2 \leq y_2)
\end{aligned}$$

Je nach verwendeter Ordnungsrelation werden unterschiedliche Bezeichnungen verwendet:

- Ist $x \in X$ optimal bzgl. $<$, so heißt x *Pareto-optimal* oder *effizient*.
- Ist $x \in X$ optimal bzgl. $\ll$, so heißt x *schwach Pareto-optimal* oder *schwach effizient*.
- Ist $x \in X$ optimal bzgl. $\leq_{\text{lex}}$, so heißt x *lexikografisch optimal*.

Abbildung 2.3 veranschaulicht die drei Begriffe. Eine Lösung ist Pareto-optimal, wenn jede Veränderung, die einen Zielfunktionswert verbessert, zu einer Verschlechterung eines anderen Zielfunktionswertes führt. Der Ökonom und Soziologe Vilfredo Pareto verwendete dieses Kriterium bezogen auf die Zufriedenheit von Individuen einer Gesellschaft. Schwache Pareto-Optimalität bedeutet, dass sich nicht beide Zielfunktionswerte verbessern lassen, im Gegensatz zur Pareto-Optimalität kann es allerdings möglich sein, einen Zielfunktionswert bei gleich bleibendem zweiten Zielfunktionswert zu verändern. Lexikografische Optimalität drückt eine Hierarchisierung der Zielfunktionen aus: Lexikografisch optimal ist eine Lösung mit optimalem ersten Zielfunktionswert und unter diesen mit bestem zweiten Zielfunktionswert.

Definition 2.53. Sei $\mathcal{I}$ eine Menge zulässiger Eingaben, welche Tupel der Form $I = (X, (f_1, f_2))$ kodieren. Wir betrachten das allgemeine bikriterielle Minimierungsproblem $(BOP_{\mathcal{I}})$ mit Eingabe $I \in \mathcal{I}$. Dann heißt das Entscheidungsproblem $(BOP_{\mathcal{I},\prec})$ für eine Ordnungsrelation $\prec$ mit

$$\chi_{(BOP_{\mathcal{I},\prec})}(X, (f_1, f_2), (k_1, k_2)) = \begin{cases} 1, & \text{falls } \exists\, x \in X : (f_1(x), f_2(x)) \prec (k_1, k_2) \\ 0, & \text{sonst,} \end{cases}$$

dessen Instanzen Tupel $(X, (f_1, f_2), (k_1, k_2))$ mit $(X, (f_1, f_2)) \in \mathcal{I}$ und $(k_1, k_2) \in \mathbb{R}^2$ sind, die *Entscheidungsversion* von $(BOP_{\mathcal{I}})$ bzgl. $\prec$.

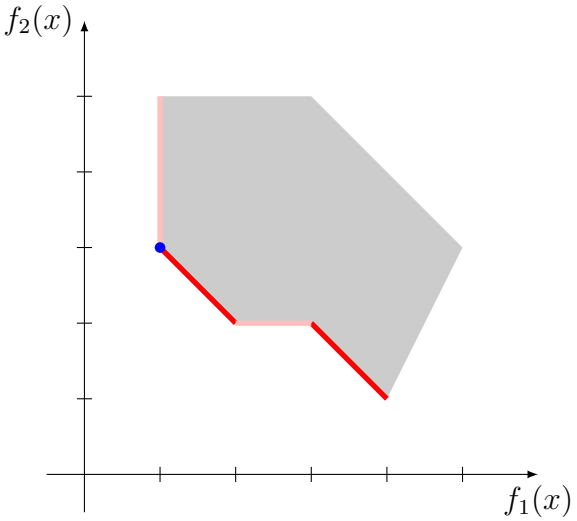

Abb. 2.3: Beispiel des Bildes $f(X)$ einer zulässigen Menge X im Zielraum $\mathbb{R}^2$ mit Kennzeichnung der lexikographisch optimalen Lösung (blau), der effizienten Lösungen (blau, rot) und der schwach effizienten Lösungen (blau, rot, rosa)

2.4 Planungsschritte vor der Fahrplanoptimierung

Die Terminologie dieses Abschnitts orientiert sich an der Vorlesung [Sch13].

Das Ergebnis des Netzwerk-Design-Schrittes ist ein Public Transportation Network (öffentliches Verkehrsnetzwerk), das im Folgenden formal definiert wird.

Definition 2.54 (Public Transportation Network)**.** Ein *Public Transportation Network* $PTN = (V, E)$ ist ein Graph mit Haltestellen/Bahnhöfen V und direkten Verbindungen E zwischen diesen. Man unterscheidet zwischen ungerichteten und gerichteten PTNs.

Mithilfe gerichteter PTNs können Einbahnstraßen modelliert werden. Außer der reinen Topologie des Netzwerks liegen in der Praxis die Längen der Kanten d_e, $e \in E$, d. h. die Abstände der beiden Stationen, in Metern vor.

In den auf das Netzwerk-Design folgenden Schritten werden Zeiten verwendet. Diese werden grundsätzlich in Minuten angegeben. Weiterhin wird ein *Planungszeitraum* D (z. B. ein Tag), der in *Planungsintervalle* der Länge T (z. B. eine Stunde) aufgeteilt wird, verwendet.

Sowohl bei der Linienplanung als auch bei der Fahrplanoptimierung werden neben den oben beschriebenen geografischen Daten häufig *Fahrgastdaten* berücksichtigt. Es werden zwei Typen unterschieden:

- Verkehrsnachfrage: Von wo nach wo möchten die Passagiere fahren?

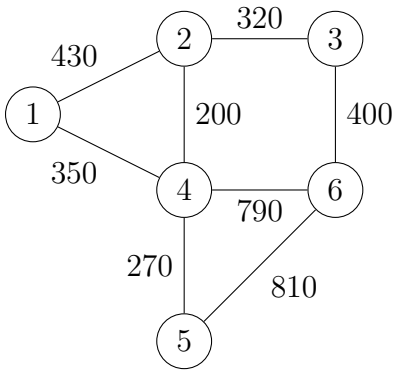

Abb. 2.4: Beispiel für ein ungerichtetes Public Transportation Network mit Kantenlängen in Metern

- Querschnittsbelastungen: Wie viele Fahrgäste nutzen eine Kante im PTN?

Definition 2.55 (OD-Matrix)**.** Seien $u, v \in V$. Dann bezeichnet C_{uv} die (Verkehrs-) Nachfrage zwischen u und v im PTN, d. h. die Anzahl der Fahrgäste, die innerhalb eines Zeitintervalls der Länge T von u nach v möchten (und den ÖV nutzen). Die Matrix $C = (C_{uv})_{u,v \in V}$ heißt *OD-Matrix*.

Im Linienplanungsschritt werden Linien im PTN festgelegt, die die Verkehrsnachfrage abdecken.

Definition 2.56 (Linie)**.** Eine *Linie* ist ein Weg im PTN. Die *Bedienhäufigkeit* f_l einer Linie l gibt an, wie häufig sie im Planungsintervall T befahren wird. Ein *Linienkonzept* $LC = (\mathcal{L}, (f_l)_{l \in \mathcal{L}})$ ist eine Menge von Linien $\mathcal{L}$ mit ihren Bedienhäufigkeiten.

Wir wollen annehmen, dass Daten darüber vorliegen, wie viele Personen an jeder Station zwischen zwei dort verkehrenden Linien umsteigen. Ist die Verkehrsnachfrage bekannt, so können diese Daten daraus berechnet werden; anderenfalls nehmen wir an, dass neben den Querschnittsbelastungen auch Daten zu allen Umstiegen bekannt sind.

2.5 Fahrplanoptimierung

2.5.1 Zulässige Fahrpläne

Bei der Fahrplanoptimierung werden Ankünften und Abfahrten von Fahrzeugen Zeiten zugeordnet, wobei zeitliche Abhängigkeiten (z. B. Fahr-, Warte und Umsteigezeiten) berücksichtigt werden müssen. Damit unterscheidet sich das Auffinden eines zulässigen Fahrplans mathematisch nicht von anderen Planungsprozessen wie z. B. dem Aufstellen eines

Projektplans: Zu einer Menge $\mathcal{E}$ von *Ereignissen* und einer Menge $\mathcal{A}$ von *Aktivitäten* mit vorgegebener Minimal- und Maximaldauer, welche zwischen den Ereignissen durchgeführt werden müssen, ist eine Zuordnung von Zeitpunkten zu den Ereignissen gesucht. Diese muss alle von notwendigen Aktivitäten herrührenden zeitlichen Beschränkungen berücksichtigen. Ereignisse und Aktivitäten lassen sich als gerichteter Graph $\mathcal{N} = (\mathcal{E}, \mathcal{A})$, das so genannte *Ereignis-Aktivitäts-Netzwerk* (EAN), darstellen, an dessen Kanten die Minimal- und Maximaldauer notiert sind.

Bei der Fahrplanoptimierung entsprechen die Ereignisse den Ankünften und Abfahrten der Fahrzeuge und die wichtigsten Aktivitäten sind Fahr-, Warte- und Umsteige-Aktivitäten. Wie die Ereignisse und Aktivitäten aus dem PTN und dem Linienkonzept bestimmt werden, wird in Abschnitt 2.5.2 beschrieben. Eine Zuordnung π von Zeitpunkten zu den Ereignissen heißt dabei *Fahrplan*. Die Zeitpunkte werden dabei in Minuten angegeben.

Wir unterscheiden zwischen *aperiodischen* und *periodischen* Fahrplänen: Bei aperiodischen Fahrplänen steht ein Wert $\pi_i \in \mathbb{Z}$ für einen einmaligen Zeitpunkt, bei periodischen Fahrplänen steht ein solcher Wert π_i für alle Zeitpunkte $\pi_i + zT$ mit $z \in \mathbb{Z}$.

Definition 2.57 (Zulässiger Fahrplan)**.** Sei $\mathcal{E}$ eine Menge von Ereignissen und $\mathcal{A}$ eine Menge von Aktivitäten mit Schranken $L_a, U_a \in \mathbb{N}_0,\ L_a \leq U_a$.

- Ein aperiodischer Fahrplan $\pi \in \mathbb{Z}^{\mathcal{E}}$ heißt *zulässig*, wenn $\pi_j - \pi_i \in [L_a, U_a]$ für alle $a = (i, j) \in \mathcal{A}$.
- Ein periodischer Fahrplan $\overline{\pi} \in \mathbb{Z}^{\mathcal{E}}$ heißt *zulässig*, wenn es für jede Aktivität $a = (i, j) \in \mathcal{A}$ ein $z_a \in \mathbb{Z}$ gibt mit $\pi_j - \pi_i + z_a \cdot T \in [L_a, U_a]$. Die z_a heißen dann *Modulo-Parameter*.

Man kann im periodischen Fall fordern, dass $\pi \in \{0, \ldots, T-1\}$, ohne Zulässigkeit und Zielfunktionswert zu verändern.

Das Problem, zu einem gegebenen EAN und Schranken einen zulässigen aperiodischen Fahrplan zu bestimmen, wird als *Feasible Differential Problem* (FDP) oder (TT0) bezeichnet, im periodischen Fall spricht man vom *Periodic Event Scheduling Problem* (PESP) oder (PTT0). Das Periodic Event Scheduling Problem wurde 1989 von Paolo Serafini und Walter Ukovich eingeführt [SU89].

Gibt es zwei Aktivitäten $a_1, a_2 \in \mathcal{A}$ mit $a_1 = (i, j)$ und $a_2 = (j, i)$, so kann es, da $L_{a_1} \geq 0$ und $L_{a_2} \geq 0$, nur dann einen zulässigen aperiodischen Fahrplan geben, wenn $L_{a_1} = L_{a_2} = 0$. In diesem Fall können a_1 und a_2 durch eine Aktivität $a = (i, j)$ mit $L_a = U_a = 0$ ersetzt werden. Wir können also annehmen, dass es niemals zwei einander entgegengesetzte Aktivitäten im EAN gibt.

Lemma 2.58 ([Sch04])**.** *Gegeben sei eine Probleminstanz $(\mathcal{N}, L, U)$ von (TT0) mit $\mathcal{N} = (\mathcal{E}, \mathcal{A})$. Wir definieren das* Hilfsnetzwerk $\mathcal{N}' := (\mathcal{E}, \mathcal{A}')$ *mit*

$$\mathcal{A}' := \mathcal{A} \cup \{(j, i) \mid (i, j) \in \mathcal{A}\},$$

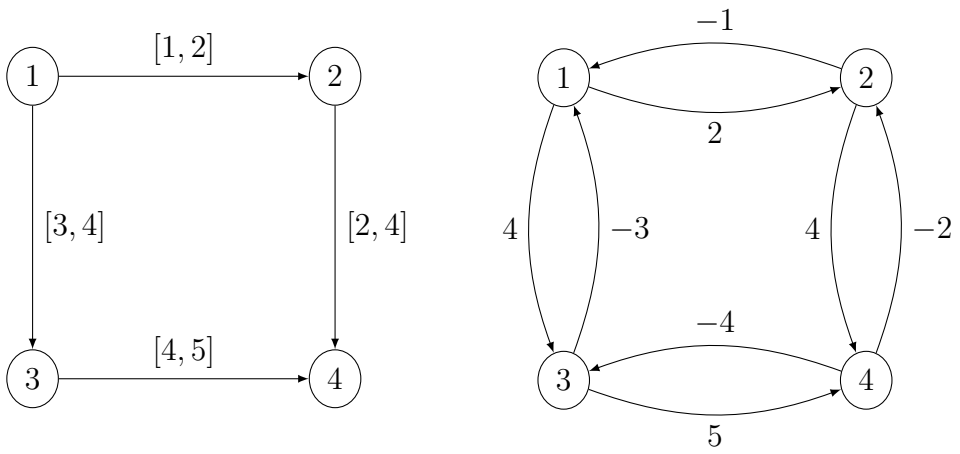

Abb. 2.5: Beispiel für eine Probleminstanz $(\mathcal{N}, L, U)$ von (TT0) und das zugehörige Hilfsnetzwerk $\mathcal{N}'$ mit Kantenlängen μ

sowie

$$\mu_{a'} := \begin{cases} U_{a'}, & \text{falls } a' \in \mathcal{A}, \\ -L_a, & \text{falls } a' = (j,i) \text{ und } a = (i,j) \in \mathcal{A}. \end{cases}$$

Ein Fahrplan π ist genau dann zulässig für $\mathcal{N}$, wenn $\pi_j - \pi_i \leq \mu_{a'}$ für alle $a' = (i,j) \in \mathcal{A}'$.

Beweis.

$$\begin{aligned} & \pi \text{ ist zulässiger Fahrplan in } \mathcal{N} \\ \Leftrightarrow\ & \forall\, a = (i,j) \in \mathcal{A} : \pi_j - \pi_i \leq U_a \text{ und } \pi_j - \pi_i \geq L_a \\ \Leftrightarrow\ & \forall\, a = (i,j) \in \mathcal{A} : \pi_j - \pi_i \leq U_a \text{ und } \pi_i - \pi_j \leq -L_a \\ \Leftrightarrow\ & \forall\, a' = (i,j) \in \mathcal{A}' : \pi_j - \pi_i \leq \mu_{a'} \\ \Leftrightarrow\ & \pi \text{ ist zulässiger Fahrplan in } \mathcal{N}' \end{aligned}$$

□

Satz 2.59 ([Sch04, Theorem 4.13])**.** *Sei $(\mathcal{N}, L, U)$ eine Probleminstanz von (TT0). Es existiert ein zulässiger Fahrplan π in $\mathcal{N}$ genau dann, wenn $\mathcal{N}'$ keinen gerichteten Kreis negativer Länge bzgl. μ enthält.*

Beweis. Angenommen, es gibt einen zulässigen Fahrplan π in $\mathcal{N}$. Nach dem letzten Lemma gilt für alle $a' \in \mathcal{A}'$, dass $\pi_j - \pi_i \leq \mu_{a'}$. Ist $C = (i_0, a_1, \ldots, i_{n-1}, a_n, i_n = i_0)$ ein beliebiger Kreis in $\mathcal{N}'$, so gilt:

$$0 = \sum_{j=1}^{n} i_j - i_{j-1} \leq \sum_{a' \in \mathcal{A}'} \mu_{a'} = \text{length}_\mu(C).$$

Wir setzen nun voraus, dass $\mathcal{N}'$ keinen gerichteten Kreis negativer Länge enthält und geben einen Algorithmus an, der in Polynomialzeit einen zulässigen Fahrplan bestimmt.

Der Algorithmus wählt ein beliebiges Ereignis $i_0 \in \mathcal{E}$ und setzt $\pi_{i_0} = 0$. Anschließend bestimmt er zu jedem Ereignis j in der gleichen Zusammenhangskomponente wie i_0 einen kürzesten gerichteten Weg W in $\mathcal{N}'$ und setzt $\pi_j := \text{length}_\mu(W)$. Dies ist immer möglich, da $\mathcal{N}'$ keine gerichteten Kreise negativer Länge enthält. Gibt es mehrere Zusammenhangskomponenten, so wiederholt der Algorithmus die Prozedur für jede Zusammenhangskomponente.

Der so konstruierte Fahrplan π ist zulässig, denn ist $a' = (i, j) \in \mathcal{A}'$ und sind W_i, W_j kürzeste Wege von i_0 nach i bzw. j, so gilt:

$$\pi_j = \text{length}_\mu(W_j) \leq \text{length}_\mu(W_i) + \mu_{a'} = \pi_i + \mu_{a'},$$

also $\pi_j - \pi_i \leq \mu_{a'}$. Nach dem letzten Lemma ist π somit zulässig. □

Den ersten Teil des folgenden Satzes haben wir durch die Angabe eines Algorithmus gleich mitbewiesen. Der zweite Teil wird in dieser Arbeit nicht bewiesen; der Beweis basiert auf einer Reduktion des Hamiltonkreisproblems und kann in jeder der beiden referenzierten Quellen nachgelesen werden.

Satz 2.60 ([Sch04, SU89])**.** *(TT0) lässt sich in polynomieller Zeit lösen, (PTT0) ist NP-vollständig.*

(TT0) und (PTT0) lassen sich als ganzzahlige lineare Programme mit konstanter Zielfunktion formulieren. Da die Nebenbedingungen mit denen der im Abschnitt 2.5.3 beschriebenen Optimierungsprobleme (TT1) bzw. (PTT1) übereinstimmen, erhält man die Formulierungen für (TT0) bzw. (PTT0) als ganzzahlige lineare Programme einfach durch Weglassen der Zielfunktion aus denen für (TT1) bzw. (PTT1). Es wird deshalb auf eine explizite Angabe an dieser Stelle verzichtet.

2.5.2 Erzeugung des Ereignis-Aktivitäts-Netzwerks

Nach der Linienplanung liegt neben dem Public Transportation Network $PTN = (V, E)$ das Linienkonzept $LC = (\mathcal{L}, (f_l)_{l \in \mathcal{L}})$ bzgl. des Planungsintervalls T vor, d. h. f_l ist die Häufigkeit, mit der Linie l während der Zeitdauer T befahren wird; die Frequenz ist also $\frac{f_l}{T}$. Wir gehen davon aus, dass PTN und Linienkonzept gerichtet sind, anderenfalls müssen Kanten und Linien für jede Richtung betrachtet werden.

Definition 2.61.

- Eine *Fahrt* t ist das Befahren einer Linie von Anfang bis Ende durch ein Fahrzeug. Sie ist gegeben durch die Linie l und eine Fahrtnummer i.

- Die von einem Linienkonzept LC im Planungszeitraum D bestimmte Menge von Fahrten ist
$$\mathcal{T} = \left\{ (l,i) \,\middle|\, l \in \mathcal{L}, i \in \{1, \dots, \frac{D \cdot f_l}{T}\} \right\}.$$
Für $t = (l,i) \in \mathcal{T}$ sei $\text{line}(t) = l$.

Notation 2.62.

- Für $l \in \mathcal{L}$ sei $\text{start}(l) \in V$ die Station, an der l startet und $\text{end}(l) \in V$ die Station an der l endet.
- Für $t \in \mathcal{T}$ sei $\text{start}(t) := \text{start}(\text{line}(t))$ und $\text{end}(t) := \text{end}(\text{line}(t))$.

Im Folgenden sind der aperiodische und der periodische Fall zu unterscheiden: Im aperiodischen Fall wird für *Fahrten*, im periodischen Fall für *Linien* geplant. Aus einem gegebenen Linienkonzept lassen sich Ereignisse und zugehörige Aktivitäten sowohl für die aperiodische Fahrplanoptimierung, als auch für die periodische Fahrplanoptimierung erzeugen.

Definition 2.63.

- Im **aperiodischen Fall** heißt ein Tripel (v,t,arr) mit $v \in V,\ t \in \mathcal{T}$ *Ankunftsereignis* und steht für die Ankunft von Fahrt t an der Station v. Genauso heißt ein Tripel (v,t,dep) *Abfahrtsereignis*.
- Die vom Linienkonzept LC vorgegebenen Mengen von Ankunfts- und Abfahrtsereignissen sind
$$\mathcal{E}_{arr} := \bigcup_{t \in \mathcal{T}} \mathcal{E}_{arr,t}, \text{ wobei} \qquad \mathcal{E}_{arr,t} := \{(v,t,arr) \mid v \in \text{line}(t) \setminus \{\text{start}(t)\}\},$$
$$\mathcal{E}_{dep} := \bigcup_{t \in \mathcal{T}} \mathcal{E}_{dep,t}, \text{ wobei} \qquad \mathcal{E}_{dep,t} := \{(v,t,dep) \mid v \in \text{line}(t) \setminus \{\text{end}(t)\}\}.$$
- Weiterhin bestimmt das Linienkonzept folgende Mengen von Aktivitäten
$$\mathcal{A}_{drive} := \bigcup_{t \in \mathcal{T}} \mathcal{A}_{drive,t}, \qquad (\textit{Fahraktivitäten})$$
$$\text{wobei } \mathcal{A}_{drive,t} := \{((v_1,t,dep),(v_2,t,arr)) \mid (v_1,v_2) \in \text{line}(t)\},$$
$$\mathcal{A}_{wait} := \bigcup_{t \in \mathcal{T}} \mathcal{A}_{wait,t}, \qquad (\textit{Wartaktivitäten})$$
$$\text{wobei } \mathcal{A}_{wait,t} := \{((v,t,arr),(v,t,dep)) \mid v \in \text{line}(t) \setminus \{\text{start}(t), \text{end}(t)\}\}.$$

Notation 2.64. Für $t \in \mathcal{T}$ sei

- $\text{dep}(t) := (\text{start}(t), t, dep)$ das erste Abfahrts- und $\text{arr}(t) := (\text{end}(t), t, arr)$ das letzte Ankunftsereignis von t,

- $\mathcal{E}_t := \mathcal{E}_{arr,t} \cup \mathcal{E}_{dep,t}, \qquad \mathcal{A}_t := \mathcal{A}_{drive,t} \cup \mathcal{A}_{wait,t}.$

Weitere Aktivitäten, die in der Praxis vorkommen können, sind Umsteigeaktivitäten und so genannte Headway- („Abstandshaltungs“-)Aktivitäten. Diese sind für den aperiodischen Fall schwierig zu bestimmen, da vor Fahrplanerstellung die Reihenfolge der unterschiedlichen Fahrten noch nicht feststeht. Entweder wird eine vernünftig erscheinende Reihenfolge vorgegeben, oder man bedient sich einer über das Feasible Differential Problem hinausgehender Methode und verwendet *Paare potenzieller Aktivitäten*, d. h. Paare von zeitlichen Beschränkungen, von denen genau eine eingehalten werden muss. Dies kann in der IP-Formulierung (siehe Abschnitt 2.5.3) durch zusätzliche Entscheidungsvariablen gewährleistet werden, wodurch das Problem allerdings NP-schwer wird.

Die in dieser Arbeit untersuchten Modelle verwenden nur feste und keine potenziellen Aktivitäten. Dabei ist es unerheblich, um welche Art von Aktivitäten es sich handelt. Es ist also möglich, sowohl feste Umsteigeaktivitäten $\mathcal{A}_{change}$ als auch feste Headway-Aktivitäten $\mathcal{A}_{headway}$ zu verwenden. Im experimentellen Teil werden allerdings keine Headway-Bedingungen verwendet. Die Umsteigeaktivitäten werden erzeugt, indem zuerst ein periodisches EAN erzeugt wird, dazu ein periodischer Fahrplan bestimmt wird und anschließend das periodische EAN wie in Abschnitt 2.5.4 beschrieben „ausgerollt“ wird.

Definition 2.65.

- Im **periodischen Fall** heißt ein Tripel (v, l, arr) mit $v \in V,\ l \in \mathcal{L}$ *Ankunftsereignis* und steht für die Ankunft von Linie l an der Station v. Analog heißt ein Tripel (v, l, dep) *Abfahrtsereignis*.

- Die vom Linienkonzept LC vorgegebenen Mengen von Ankunfts- und Abfahrtsereignissen sind

$$\mathcal{E}_{arr} := \bigcup_{l \in \mathcal{L}} \mathcal{E}_{arr,l}, \text{ wobei} \qquad \mathcal{E}_{arr,l} := \{(v, l, arr) \mid v \in l \setminus \{\text{start}(l)\}\},$$
$$\mathcal{E}_{dep} := \bigcup_{l \in \mathcal{L}} \mathcal{E}_{dep,l}, \text{ wobei} \qquad \mathcal{E}_{dep,l} := \{(v, l, dep) \mid v \in l \setminus \{\text{end}(l)\}\}.$$

- Weiterhin bestimmt das Linienkonzept folgende Mengen von Aktivitäten

$$\mathcal{A}_{drive} := \bigcup_{l \in \mathcal{L}} \mathcal{A}_{drive,l}, \qquad (\textit{Fahraktivitäten})$$
$$\text{wobei } \mathcal{A}_{drive,l} := \{((v_1, l, dep), (v_2, l, arr)) \mid (v_1, v_2) \in l\},$$
$$\mathcal{A}_{wait} := \bigcup_{l \in \mathcal{L}} \mathcal{A}_{wait,l}, \qquad (\textit{Warteaktivitäten})$$
$$\text{wobei } \mathcal{A}_{wait,l} := \{((v, l, arr), (v, l, dep)) \mid v \in l \setminus \{\text{start}(l), \text{end}(l)\}\}.$$

Im periodischen Fall geben die Umsteige- und Headway-Aktivitäten keine Reihenfolge vor. In [Sch04] wird angenommen, dass zu haltende Umsteigeverbindungen vorgegeben sind. Eine Möglichkeit, diese vorzugeben ist, wie in [Sie11] für jede Station, an der sich mehrere Linien kreuzen, ein Umstieg von jeder dieser Linien in jede andere dieser Linien vorgesehen werden, mit Ausnahme von Umstiegen in Linien, die in die Rückrichtung fahren. Die Anzahl der Umsteigeaktivitäten kann noch weiter eingeschränkt werden, indem z. B. für zwei Linien, die ein Stück parallel fahren nicht an jeder Station auf diesem Stück einen Umstieg vorgesehen wird: Fahren die beiden Linien gleich schnell, reicht es aus, eine Umsteigeaktivität an der ersten oder an der letzten Station einzuplanen.

Definition 2.66 (Event-Activity-Network zu gegebenem Linienkonzept)**.** Das vom Linienkonzept LC (sowie ggf. von Umsteigeverbindungen) vorgegebene EAN ist

$$\mathcal{N} := (\mathcal{E}, \mathcal{A}) := (\mathcal{E}_{arr} \cup \mathcal{E}_{dep}, \mathcal{A}_{drive} \cup \mathcal{A}_{wait} \cup \mathcal{A}_{change} \cup \mathcal{A}_{headway}).$$

Definition 2.57 sieht vor, dass alle Ereignisse die gleiche Periode haben. Für verschiedene Linien l können allerdings unterschiedliche Bedienhäufigkeiten f_l gegeben sein, d. h. die entsprechenden Ankünfte und Abfahrten sollen sich mit linienspezifischer Periode $\frac{T}{f_l}$ wiederholen. Dies kann entweder durch ein Attribut für jedes Ereignis oder mithilfe des in [Sie11] vorgestellten Verfahrens `frequency_as_multiplicity` modelliert werden. In diesem Verfahren wird für das kleinste gemeinsame Vielfache der vorkommenden Perioden geplant. Jede Linie wird so oft in das EAN aufgenommen, wie sie innerhalb dieser Zeit bedient wird, wobei gleiche zeitliche Abstände durch zusätzliche *Synchronisationsaktivitäten* erzwungen werden. In [Sie11] wird der Algorithmus zur Bestimmung des EAN angegeben, welcher zur Erzeugung der Instanzen im experimentellen Teil verwendet wurde.

Zu jeder Aktivität muss weiterhin die minimal und die maximal erlaubte Dauer $0 \leq L_a \leq U_a$ bestimmt werden.

- Für Fahraktivitäten $a = ((v_1, t, dep), (v_2, t, arr)) \in \mathcal{A}_{drive}$ ist L_a normalerweise die minimal nötige Dauer für die Fahrt von v_1 nach v_2. Diese kann z. B. anhand der Kantenlänge $d_{(v_1,v_2)}$ im PTN berechnet werden. Die obere Schranke könnte z. B. die maximale Fahrzeit sein, die Kunden noch akzeptieren werden oder die maximale Zeit, die das Gleis blockiert sein darf, sein. Sie kann z. B. durch Addition oder Multiplikation der unteren Schranke mit einer Konstanten gewonnen werden.
- Für Warteaktivitäten $((v, t, arr), (v, t, dep)) \in \mathcal{A}_{wait}$ kann L_a z. B. die Zeit sein, die Fahrgäste typischerweise zum Ein- und Aussteigen an der Station v benötigen. Diese Zahl kann z. B. für alle Warteaktivitäten gleich oder haltestellenspezifisch sein. Als obere Schranke kann beispielsweise die Zeit gewählt werden, die ein Gleis maximal durch einen wartenden Zug belegt sein darf.
- Für Umsteigeaktivitäten $((v, t_1, arr), (v, t_2, dep)) \in \mathcal{A}_{change}$ ist die untere Schranke die Zeit, die Fahrgäste brauchen, um an der Station v von einem Zug in einen anderen umzusteigen. Sie kann für alle Umsteigeaktivitäten gleich sein oder von der Haltestelle

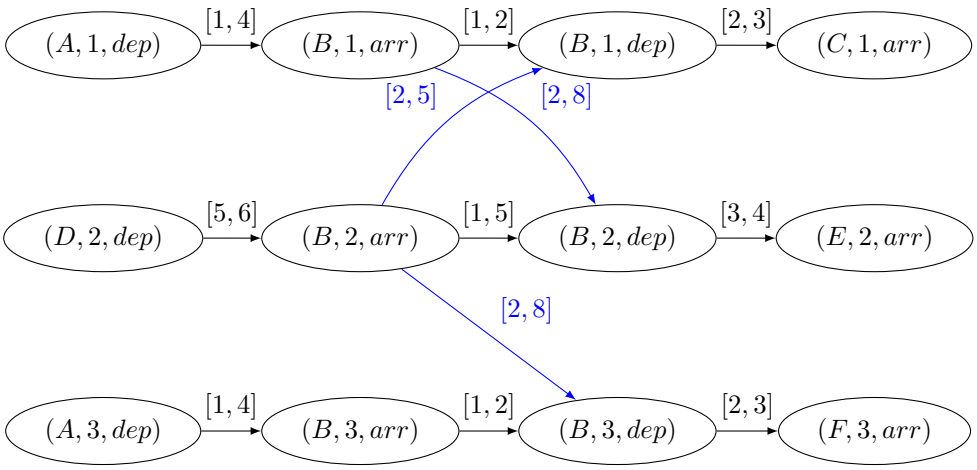

Abb. 2.6: Beispiel für ein Ereignis-Aktivitäts-Netzwerk mit unteren und oberen Schranken

v abhängen. Die obere Schranke kann z. B. die Zeit sein, die Kunden maximal bereit sind zu warten. Wird für den periodischen Fall ohne Headway-Aktivitäten $U_a := L_a + T - 1$ für alle $a \in \mathcal{A}_{change}$ gewählt, so gibt es stets einen zulässigen periodischen Fahrplan, weil die Bedingungen der Umsteigeaktivitäten dann immer erfüllt sind.

2.5.3 Optimale Fahrpläne

Als Qualitätsmaß für zulässige Fahrpläne wird die Gesamtreisezeit aller Passagiere verwendet. Um diese zu bestimmen, werden Informationen darüber benötigt, wie die Passagiere fahren. Dazu ist klassischerweise für jede Aktivität $a \in \mathcal{A}$ eine Zahl $c_a \in \mathbb{N}_0$ gegeben, die angibt, wie viele Passagiere diese Aktivität voraussichtlich nutzen werden. Das Problem, zu gegebenem EAN sowie Schranken und Passagierzahlen für alle Aktivitäten einen zulässigen aperiodischen/periodischen Fahrplan mit minimaler Gesamtreisezeit zu bestimmen, wird als (TT1) bzw. (PTT1) bezeichnet.

Die Zahlen c_a können entweder in Form von Querschnittsbelastungen und den Umsteigerzahlen vorgegeben sein oder anhand der Werte der OD-Matrix unter der Annahme berechnet werden, dass alle Fahrgäste auf möglichst schnellen Wegen fahren. Dazu wird für jedes Paar $(u, v) \in V^2$ ein kürzester Weg von u nach v im EAN bestimmt, wobei als Längen die geschätzten Zeiten der Aktivitäten verwendet werden. Dieses Vorgehen ist bloß eine Näherung, da die Zeiten der Aktivitäten und somit die kürzesten Wege im EAN vom Fahrplan abhängen und sich deshalb die kürzesten Wege nach der Festlegung der Fahrzeiten ändern können. Das in [Sie11] beschriebene ganzzahlige lineare Programm (ODPESP) bestimmt einen Fahrplan π derart, dass die Gesamtfahrzeit aller Fahrgäste auf zu π gehörenden kürzesten Wegen nicht größer ist als die Gesamtfahrzeit bei irgendeinem Fahrplan π' auf zu π' gehörenden kürzesten Wegen. Näherungsweise wird das Problem von dem Verfahren *Retimetabling*, das ebenfalls in [Sie11] beschrieben wird, berücksichtigt, welches

iterativ abwechselnd einen Fahrplan zu festen Wegen der Fahrgäste und die Fahrgastwege zu einem festen Fahrplan berechnet. Eine entsprechende Heuristik für die aperiodische Fahrplanoptimierung wird in [Anh12] beschrieben.

Für (TT1) und (PTT1) gibt es mehrere IP-Formulierungen [Sch04], von denen in dieser Arbeit allerdings nur die folgende benötigt wird:

$$\begin{aligned}
\min \quad & \sum_{a=(i,j)\in\mathcal{A}} c_a(\pi_j - \pi_i) & \\
\text{s.d.} \quad & \pi_j - \pi_i \leq U_a & \forall\, a=(i,j)\in\mathcal{A} \\
& \pi_j - \pi_i \geq L_a & \forall\, a=(i,j)\in\mathcal{A} \\
& \pi_i \in \mathbb{Z} & \forall\, i\in\mathcal{E}
\end{aligned}$$

Satz 2.67. *(TT1) ist polynomiell lösbar*

Beweis. Die Koeffizientenmatrix des obigen ganzzahligen Programms ist nach [Sch04] total unimodular, da jede Zeile genau eine 1 und eine -1 enthält. Außerdem ist die rechte Seite des Ungleichungssystems ganzzahlig. Somit sind alle Voraussetzungen von Korollar 2.47 erfüllt. □

(PTT1) kann folgendermaßen mithilfe der in Abschnitt 2.5.1 eingeführten Modulo-Parameter als ganzzahliges lineares Programm formuliert werden.

$$\begin{aligned}
\min \quad & \sum_{a=(i,j)\in\mathcal{A}} c_a(\pi_j - \pi_i) + z_a \cdot T & \\
\text{s.d.} \quad & \pi_j - \pi_i + z_a \cdot T \leq U_a & \forall\, a=(i,j)\in\mathcal{A} \\
& \pi_j - \pi_i + z_a \cdot T \geq L_a & \forall\, a=(i,j)\in\mathcal{A} \\
& \pi_i \in \mathbb{Z} & \forall\, i\in\mathcal{E} \\
& z_a \in \mathbb{Z} & \forall\, a\in\mathcal{A}
\end{aligned}$$

Satz 2.68. *Die Entscheidungsversion von (PTT1) ist NP-vollständig.*

Beweis. Offenbar lässt sich eine gegebene Lösung in polynomieller Zeit überprüfen. (PTT0) lässt sich durch $c_a := 0$ für alle $a \in \mathcal{A}$ auf (PTT1) reduzieren. □

2.5.4 Ausrollen des Ereignis-Aktivitäts-Netzwerks

Ein periodisches EAN $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ kann in ein aperiodisches EAN $\mathcal{N}' = (\mathcal{E}', \mathcal{A}')$ umgewandelt werden, indem jedes Ereignis i durch $\frac{D}{T}$ Ereignisse $(i,1), \ldots, (i, \frac{D}{T})$ ersetzt wird. Dieses Vorgehen wird als *Ausrollen* des periodischen EANs bezeichnet. Wie in Abschnitt 2.5.2 beschrieben, ist in einem aperiodischen EAN im Gegensatz zum periodischen EAN

die Reihenfolge von Ereignissen, zwischen denen eine Aktivität existiert, festgelegt. Aus diesem Grund kann ein periodisches EAN nicht eindeutig ausgerollt werden. Für jede Aktivität $a = (i_1, i_2) \in \mathcal{A}$ kann ein $z_a \in \{0, \ldots, \frac{D}{T} - 1\}$ bestimmt werden, so dass $((i_1, 1), (i_2, 1 + z_a)) \in \mathcal{A}'$. Dann ist

$$\mathcal{A}' = \{((i_1, k), (i_2, k + z_a)) \mid a = (i_1, i_2) \in \mathcal{A},\ k \in \{1, \ldots, \frac{D}{T} - z_a\}\}.$$

Es ist möglich, dass es für $z \in \{0, \ldots, \frac{D}{T} - 1\}^{\mathcal{A}}$ keinen zulässigen Fahrplan in $\mathcal{N}'$ gibt.

Wenn zu dem periodischen EAN bereits ein periodischer Fahrplan $\pi \in \{0, \ldots, T-1\}^{\mathcal{E}}$ feststeht, können für z_a die zugehörigen kleinstmöglichen Werte der Modulo-Parameter aus der IP-Formulierung gewählt werden. Dieses Verfahren wird zur Erzeugung der Testinstanzen aus Abschnitt 3.4 angewendet. Der aperiodische Fahrplan π' mit $\pi'_{(i,k)} = \pi_i + (k-1)T$ heißt dann der *ausgerollte Fahrplan* zu π.

2.6 Umlaufplanung

Jede Fahrt $t \in \mathcal{T}$ (siehe Definition 2.61) muss von einem Fahrzeug bedient werden. Dabei könnte jede Fahrt theoretisch von einem eigenen Fahrzeug durchführt werden. Allerdings können Kosten gespart werden, wenn mehrere Fahrten vom gleichen Fahrzeug übernommen werden. Das Ziel der Umlaufplanung ist es, eine Zuordnung von Fahrzeugen zu Fahrten zu finden, so dass alle Fahrten abgedeckt werden und die Kosten möglichst gering gehalten werden. Wir betrachten dabei den Fall, dass alle Fahrzeuge vom gleichen Typ sind und es nur ein Fahrzeugdepot gibt. Einen Überblick über unterschiedliche mathematische Modelle für diesen Fall sowie für mehrere Depots oder mehrere Fahrzeugtypen geben [BK09] und [Uff10]. Diese Arbeiten enthalten auch weiterführende Literaturverweise. In [AGKS06] wird ein Modell vorgestellt, bei dem die Zusammenstellung von Zügen aus mehreren Waggons für eine einzige Linie bestimmt wird.

Definition 2.69 (Trip-Graph)**.** Zu einer Menge $\mathcal{T}$ von Fahrten ist der *Trip-Graph* definiert als $G := (\mathcal{T}, \mathcal{C})$, wobei $(t_1, t_2) \in \mathcal{C}$ bedeutet, dass Fahrt t_2 nach Fahrt t_1 vom selben Fahrzeug ausgeführt werden kann. t_1 und t_2 heißen dann *kompatibel*.

Wie die Menge $\mathcal{C}$ bestimmt werden kann, wird in Abschnitt 2.6.3 beschrieben. Der Trip-Graph übernimmt in der Umlaufplanung die Rolle des EANs in der Fahrplanoptimierung. Obwohl ersterer nicht sehr verbreitet ist und in der Literatur nur ab und zu der Veranschaulichung dient (z. B. [Uff10, Abb. 2.2]), sind in Analogie zur Fahrplanoptimierung die Modelle zur Umlaufplanung in dieser Arbeit anhand des Trip-Graphen formuliert.

Definition 2.70 (Fahrzeugroute)**.** Eine *Fahrzeugroute* R ist ein gerichteter Weg im Trip-Graphen.

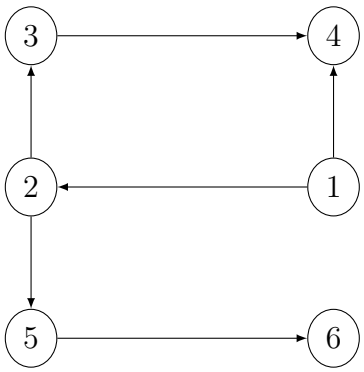

Abb. 2.7: Beispiel für einen Trip-Graphen

Anschaulich Da wir nur einen Fahrzeugtyp betrachten, reicht es, anstatt einer expliziten Zuordnung von Fahrzeugen zu Fahrten, bloß eine Menge von Fahrzeugrouten zu betrachten.

Definition 2.71 (Umlaufplan)**.** Sei $G = (\mathcal{T}, \mathcal{C})$ ein Trip-Graph. Eine Menge $\mathcal{R}$ von Fahrzeugrouten heißt *Umlaufplan* zu G, falls jede Fahrt $t \in \mathcal{T}$ in genau einer Fahrzeugroute $R \in \mathcal{R}$ enthalten ist.

2.6.1 Minimierung der Fahrzeuganzahl

Das Problem, zu gegebenem Trip-Graphen G einen Umlaufplan minimaler Kardinalität zu finden, wird als (VS1) bezeichnet. Die IP-Formulierung wurde schon 1970 von J. L. Saha aufgestellt [Sah70]. Sie basiert auf folgender Idee:

Lemma 2.72 ([Sch13])**.** *Sei $G = (\mathcal{T}, \mathcal{C})$ ein Trip-Graph ohne gerichtete Kreise und $\mathcal{R}$ ein Umlaufplan zu G und*

$$E(\mathcal{R}) := \{e \in \mathcal{C} \mid \exists\, R \in \mathcal{R} : e \in E(R)\}.$$

Dann gilt:

$$|\mathcal{R}| = |\mathcal{T}| - |E(\mathcal{R})|.$$

Somit kann, anstatt $|\mathcal{R}|$ zu minimieren, äquivalent $|E(\mathcal{R})|$ maximiert werden. Das Ziel ist es also, eine möglichst große Menge $\mathcal{C}'$ von Kanten auszuwählen, so dass es einen Umlaufplan $\mathcal{R}$ gibt mit $\mathcal{C}' = E(\mathcal{R})$.

Beispiel. Die in Abbildung 2.8 markierte Kantenmenge $\mathcal{C}'$ legt folgenden Umlaufplan fest, wobei die Wege durch ihre Knotenfolgen angegeben werden:

$$\mathcal{R} = \{(1, 2, 5, 6), (3, 4)\}.$$

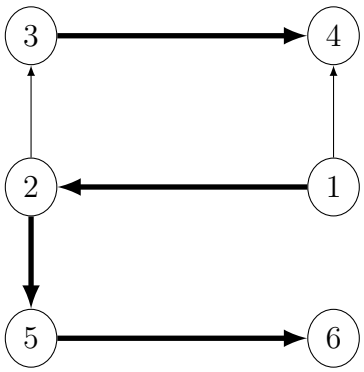

Abb. 2.8: Trip-Graph mit markierter Kantenmenge, zu der es einen Umlaufplan gibt

Lemma 2.73 ([Sch13])**.** *Sei $G = (\mathcal{T}, \mathcal{C})$ ohne gerichtete Kreise und $\mathcal{C}' \subseteq \mathcal{C}$. Genau dann gibt es einen Umlaufplan $\mathcal{R}$ mit $\mathcal{C}' = E(\mathcal{R})$, wenn jeder Knoten $t \in \mathcal{T}$ höchstens eine eingehende und höchstens eine ausgehende Kante in $\mathcal{C}'$ besitzt.*

Beweis. Angenommen, es gibt einen Umlaufplan $\mathcal{R}$ mit $\mathcal{C}' = E(\mathcal{R})$. Sei $t \in \mathcal{T}$. Nach der Definition eines Umlaufplans gibt es genau ein $R \in \mathcal{R}$ mit $t \in V(R)$. Also ist R die einzige Route, die nach t eingehende oder von t ausgehende Kanten enthält. Da G keine gerichteten Kreise hat, enthält R den Knoten t nur einmal, also auch nur höchstens eine ein- und eine ausgehende Kante.

Für die Umkehrung konstruieren wir den Umlaufplan $\mathcal{R}$ zu $\mathcal{C}'$ wie in Algorithmus 1.

Algorithmus 1 Erzeugung eines Umlaufplans aus einer Menge von Kanten

```
Initialisiere T' := T und R := ∅.
while ∃ t ∈ T' ohne eingehende Kante aus C' do
    Erzeuge leere Fahrzeugroute R.
    Füge t zu R hinzu.
    Entferne t aus T'.
    while ∃ s ∈ T' : (t, s) ∈ C' do
        Füge (t, s) zu R hinzu.
        Füge s zu R hinzu.
        Entferne s aus T'.
        Setze t := s.
    end while
    Füge R zu R hinzu.
end while
```

$\mathcal{T}'$ enthält offenbar nach jeder Iteration die noch nicht abgedeckten Fahrten, da zu Beginn ist $\mathcal{T}' = \mathcal{T}$ ist und jede Fahrt aus $\mathcal{T}'$ entfernt wird, sobald sie von einer Fahrzeugroute abgedeckt wurde.

Die von dem Algorithmus konstruierte Menge $\mathcal{R}$ von Fahrzeugrouten ist tatsächlich ein Umlaufplan, denn:

- Da nur Fahrten aus $\mathcal{T}'$ zu Routen hinzugefügt werden, wird jede Fahrt von höchstens einer Route abgedeckt.
- Alle Fahrten werden von mindestens einer Fahrzeugroute abgedeckt, denn der Graph

  $$(\mathcal{T}', \{(i,j) \in \mathcal{C}' \mid i,j \in \mathcal{T}'\})$$

  ist in jeder Iteration ein Teilgraph von G. Demnach enthält er niemals gerichtete Kreise. Folglich gibt es, solange $\mathcal{T}' \neq \emptyset$ ist, einen Knoten $t \in \mathcal{T}'$ ohne eingehende Kante aus $\mathcal{C}'$ (sonst ließe sich ein gerichteter Kreis konstruieren). Infolgedessen gilt, wenn der Algorithmus terminiert, $\mathcal{T}' = \emptyset$, d. h. es gibt keine unabgedeckten Fahrten mehr.

Es gilt $E(\mathcal{R}) \subseteq \mathcal{C}'$, denn der Algorithmus fügt nur Kanten aus $\mathcal{C}'$ zu Routen hinzu. Sei nun $e \in \mathcal{C}'$ und t der Knoten, an dem e startet. Dieser wird wie oben gezeigt vom Algorithmus betrachtet und zu einer Fahrzeugroute hinzugefügt. Anschließend prüft der Algorithmus, ob es eine von t ausgehende Kante gibt. Da nach Voraussetzung e die einzige ausgehende Kante ist, fügt der Algorithmus e direkt nach t zur Fahrzeugroute hinzu. Dies gilt für alle $e \in \mathcal{C}'$, also ist $\mathcal{C}' \subseteq E(\mathcal{R})$. □

Das Lemma bildet das Fundament zur folgenden IP-Formulierung, wobei als Variablen $z_{(t_1,t_2)}$ für alle $(t_1,t_2) \in \mathcal{C}$ verwendet werden mit

$$z_{(t_1,t_2)} = \begin{cases} 1, & \text{falls } (t_1,t_2) \in E(\mathcal{R}), \\ 0, & \text{sonst.} \end{cases}$$

$$\begin{aligned} \max \quad & \sum_{(t_1,t_2)\in\mathcal{C}} z_{(t_1,t_2)} \\ \text{so dass} \quad & \sum_{t_2:(t_1,t_2)\in\mathcal{C}} z_{(t_1,t_2)} \leq 1 && \forall\, t_1 \in \mathcal{T} \\ & \sum_{t_1:(t_1,t_2)\in\mathcal{C}} z_{(t_1,t_2)} \leq 1 && \forall\, t_2 \in \mathcal{T} \\ & z_{(t_1,t_2)} \in \{0,1\} && \forall\, (t_1,t_2) \in \mathcal{C} \end{aligned}$$

Lemma 2.74 ([Sch13])**.** *(VS1) lässt sich in polynomieller Zeit lösen.*

Beweis. Die Koeffizientenmatrix der obigen IP-Formulierung ist total unimodular, da jede jede Spalte der Koeffizientenmatrix genau eine Eins in der oberen und genau eine Eins in der unteren Spalte enthält. Weiterhin ist die rechte Seite ist ganzzahlig, also lässt sich das IP in polynomieller Zeit lösen. Ausgehend von der Lösung z^* kann ein Umlaufplan mithilfe

von Algorithmus 1 bestimmt werden, wobei $\mathcal{C}' = \{(t_1, t_2) \in \mathcal{C} \mid z^*_{(t_1,t_2)} = 1\}$. Dies verläuft ebenfalls in Polynomialzeit: Der Algorithmus kann so implementiert werden, dass für jede Fahrt t und für jede Kante e höchstens einmal geprüft wird, ob e eine in t eingehende Kante ist, und höchstens einmal, ob e eine aus t ausgehende Kante ist. Also kann die Laufzeit mit $\mathcal{O}(|\mathcal{T}||\mathcal{C}|)$ abgeschätzt werden. □

2.6.2 Minimierung der Umlaufkosten

Das eigentliche Ziel im Umlaufplanungsschritt ist es, die Kosten für das Verkehrsunternehmen zu minimieren. Im letzten Abschnitt wurde davon ausgegangen, dass diese hauptsächlich durch die Anzahl der benötigten Fahrzeuge bestimmt werden, so dass eine Minimierung der Fahrzeuganzahl auch zu einer Minimierung der Kosten führt. Allerdings verursachen Leerfahrten ebenfalls Kosten und es ist möglich, dass zur Einsparung eines Fahrzeugs übermäßig viele und lange Leerfahrten erforderlich sind. Das zeigt, dass die Regel, die Fahrzeuganzahl zu minimieren, ihr eigentliches Ziel, die Kosten zu senken, verfehlen kann. Dantzig bezeichnet in seinem Aufsatz [Dan02] über die Anfänge der linearen Programmierung feste Regeln, die der Erreichung eines höheren Ziels dienen sollen, als *ground rules* und entschied „that the *ad hoc* ground rules had to be discarded and replaced by an explicit objective function“, m. a. W. dass man das Ziel selbst im Auge behalten und nicht ein Zwischenziel zum Selbstzweck erheben solle.

In diesem Sinne werden im Modell (VS2) die tatsächlich anfallenden Umlaufkosten minimiert. Dazu sind die folgenden Kosten gegeben:

- Die Kosten c_{t_1,t_2} für den möglichen Übergang von t_1 nach t_2, für alle $(t_1, t_2) \in \mathcal{C}$,
- die Kosten $c_{0,t}$ für die Fahrt vom Depot zur ersten Station von Fahrt t, für alle $t \in \mathcal{T}$,
- die Kosten $c_{t,0}$ für die Fahrt von der letzten Station von Fahrt t zum Depot, für alle $t \in \mathcal{T}$,
- die auf den Planungszeitraum D umgerechneten vom Fahrpensum unabhängigen Kosten α eines Fahrzeugs (Anschaffung, Wartung, Steuern usw.).

Die Kosten eines Umlaufplans unterteilen sich in die vom Umlaufplan unabhängigen Kosten, die bei der Bedienung der Fahrten anfallen, und die vom Umlaufplan abhängigen Kosten für Fahrzeuge und Leerfahrten. Im Umlaufplanungsschritt müssen nur letztere betrachtet werden. Sie berechnen sich für einen gegebenen Umlaufplan $\mathcal{R}$ als $\sum_{R \in \mathcal{R}} C(R)$, wobei für $R = (t_1, e_1, t_2, \ldots, e_{L-1}, t_L)$

$$C(R) = \alpha + c_{0,t_1} + \sum_{i=1}^{L-1} c_{t_i,t_{i+1}} + c_{t_i,0}.$$

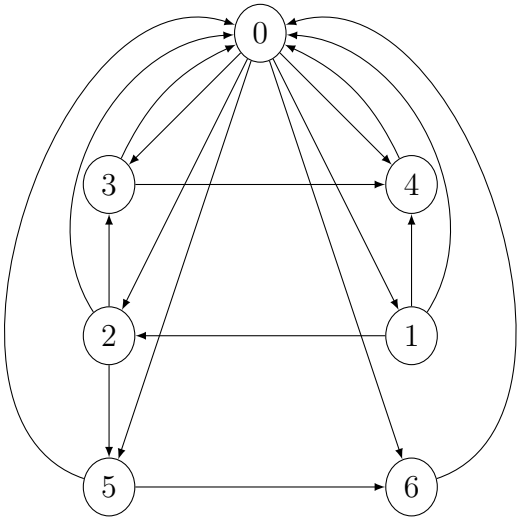

Abb. 2.9: Zum Trip-Graphen aus Abb. 2.7 passender erweiterter Trip-Graph

Das Problem (VS2) besteht darin, einen Umlaufplan mit minimalen Kosten zu bestimmen, wobei eine vorgegebene Anzahl d von Fahrzeugen nicht überschritten werden darf.

Dazu erweitern wir den Trip-Graphen G um einen weiteren Knoten 0, welcher das Depot repräsentiert, und fügen Kanten von diesem zu jedem anderen und von jedem anderen zu diesem ein. Wir nennen den so erhaltenen Graphen

$$\tilde{G} = (\tilde{\mathcal{T}}, \tilde{\mathcal{C}}) = (\mathcal{T} \cup \{0\}, \mathcal{C} \cup \{(0,t), (t,0) \mid t \in \mathcal{T}\})$$

den *erweiterten Trip-Graphen*. Wir ordnen nun jeder Kante $e \in \tilde{\mathcal{C}}$ Kosten C_e folgendermaßen zu

$$C_e := \begin{cases} c_{t_1,t_2}, & \text{für } e = (t_1, t_2) \in \mathcal{C}, \\ c_{0,t} + \frac{\alpha}{2} & \text{für } e = (0,t) \text{ mit } t \in \mathcal{T}, \\ c_{t,0} + \frac{\alpha}{2} & \text{für } e = (t,0) \text{ mit } t \in \mathcal{T}. \end{cases}$$

Wir definieren eine *erweiterte Fahrzeugroute* $\tilde{R}$ als gerichteten Kreis im erweiterten Trip-Graphen, welcher den Knoten 0 enthält, und einen *erweiterten Umlaufplan* $\tilde{\mathcal{R}}$ als eine Menge von erweiterten Fahrzeugrouten, so dass jede Fahrt in genau einer erweiterten Fahrzeugroute aus $\tilde{\mathcal{R}}$ enthalten ist.

Ist $\tilde{R}$ eine erweiterte Fahrzeugroute in $\tilde{G}$, so bezeichnen wir mit route($\tilde{R}$) die zugehörige Fahrzeugroute in G, welche man durch Entfernen des Knotens 0 und der beiden angrenzenden Kanten aus $\tilde{R}$ enthält. Offenbar ist $C(\text{route}(\tilde{R})) = \sum_{e \in E(\tilde{R})} C_e$. Weiter definieren wir für einen erweiterten Umlaufplan $\tilde{\mathcal{R}}$ den gewöhnlichen Umlaufplan $\mathcal{R}(\tilde{\mathcal{R}}) := \{\text{route}(\tilde{R}) \mid \tilde{R} \in \tilde{\mathcal{R}}\}$.

Die Idee der folgenden IP-Formulierung ist es, eine Kantenmenge $\mathcal{C}'$ auszuwählen, aus der sich ein erweiterter Umlaufplan $\tilde{\mathcal{R}}$ mit minimalen Kosten $\sum_{\tilde{R} \in \tilde{\mathcal{R}}} \sum_{e \in E(\tilde{R})} C_e$ konstruieren lässt. Dazu gehört der gewöhnliche Umlaufplan $\mathcal{R}(\tilde{\mathcal{R}})$ mit minimalen Kosten.

Beispiel. Aus der in Abbildung 2.10 markierten Kantenmenge $\mathcal{C}'$ lässt sich der erweiterte Umlaufplan konstruieren, zu dem der Umlaufplan aus dem letzten Beispiel gehört.

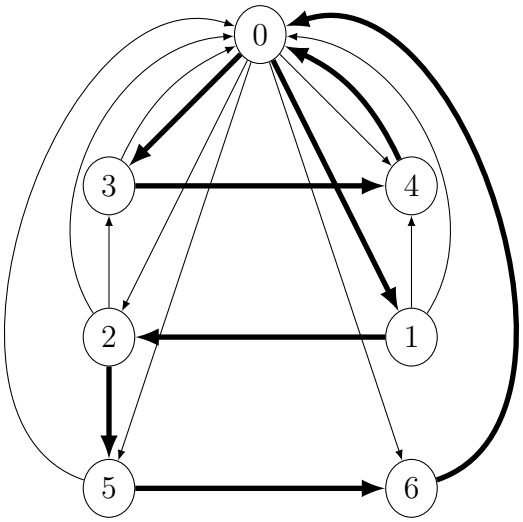

Abb. 2.10: Erweiterter Trip-Graph mit markierter Kantenmenge, zu der es einen erweiterten Umlaufplan gibt

Lemma 2.75. *Sei $G = (\mathcal{T}, \mathcal{C})$ ein Trip-Graph ohne gerichtete Kreise, $\tilde{G} = (\mathcal{T} \cup \{0\}, \tilde{\mathcal{C}})$ der zugehörige erweiterte Trip-Graph und $\mathcal{C}' \subseteq \tilde{\mathcal{C}}$. Genau dann gibt es einen erweiterten Umlaufplan $\tilde{\mathcal{R}}$ mit $\mathcal{C}' = E(\tilde{\mathcal{R}}) := \{e \in \tilde{\mathcal{C}} \mid \exists\, \tilde{R} \in \tilde{\mathcal{R}} : e \in E(\tilde{R})\}$, wenn jeder Knoten $t \in \mathcal{T}$ genau eine ein- und eine ausgehende Kante in $\mathcal{C}'$ besitzt.*

Beweis. Ist $\tilde{\mathcal{R}}$ ein erweiterter Umlaufplan, so ist jede Fahrt $t \in \mathcal{T}$ in genau einer erweiterten Fahrzeugroute $\tilde{R}$ enthalten. Also ist $\tilde{R}$ die einzige erweiterte Fahrzeugroute, die überhaupt nach t eingehende oder von t ausgehende Kanten enthält. Da $\tilde{R}$ ein Kreis ist, enthält $\tilde{R}$ den Knoten t genau einmal, also genau eine ein- und eine ausgehende Kante.

Für die Umkehrung betrachte man $\mathcal{C}' \cap \mathcal{C}$. In dieser Menge enthält jede Fahrt höchstens eine ein- und eine ausgehende Kante, also gibt es einen dazu passenden Umlaufplan $\mathcal{R}$ in G. Die Start- und Endfahrten der Fahrzeugrouten $R \in \mathcal{R}$ enthalten allerdings ebenfalls eine ein- bzw. ausgehende Kante in $\mathcal{C}'$. Diese muss also vom bzw. zum Knoten 0 sein. Somit lässt sich aus $\mathcal{R}$ ein zu $\mathcal{C}'$ passender erweiterter Umlaufplan konstruieren, indem der Endknoten jeder Route $R \in \mathcal{R}$ mit dem Knoten 0 und der Knoten 0 mit dem Startknoten von R verbunden wird. □

Auf dem Lemma fußt die folgende IP-Formulierung mit Variablen z_e für $e \in \tilde{\mathcal{C}}$. Bedingung (2.4) stellt sicher, dass nicht mehr als d Fahrzeuge benötigt werden.

$$\begin{aligned}
&\min \sum_{e \in \tilde{\mathcal{C}}} C_e z_e \\
&\text{s. d.} \sum_{\substack{t_2 \in \mathcal{T} \cup \{0\} \\ (t_1,t_2) \in \tilde{\mathcal{C}}}} z_{(t_1,t_2)} = 1 && \forall\, t_1 \in \mathcal{T} && (2.2)\\
&\sum_{\substack{t_1 \in \mathcal{T} \cup \{0\} \\ (t_1,t_2) \in \tilde{\mathcal{C}}}} z_{(t_1,t_2)} = 1 && \forall\, t_2 \in \mathcal{T} && (2.3)\\
&\sum_{t \in \mathcal{T}} z_{(0,t)} \leq d && && (2.4)\\
&z_e \in \{0,1\} && \forall\, e \in \tilde{\mathcal{C}} && (2.5)
\end{aligned}$$

Bemerkung 2.76. Für jede zulässige Lösung des obigen IPs gilt

$$\sum_{t \in \mathcal{T}} z_{(t,0)} = \sum_{t \in \mathcal{T}} z_{(0,t)} \leq d.$$

Beweis. Nach Lemma 2.75 kann man zu $\mathcal{C}' := \{e \in \tilde{\mathcal{C}} \mid z_e = 1\}$ einen erweiterten Umlaufplan $\tilde{\mathcal{R}}$ finden mit $\mathcal{C}' = E(\tilde{\mathcal{R}})$. Da jede erweiterte Fahrzeugroute ein gerichteter Kreis ist, der den Knoten 0 enthält, folgt aus $\sum_{t \in \mathcal{T}} z_{(0,t)} \leq d$, dass $|\tilde{\mathcal{R}}| \leq d$, woraus wiederum obige Bedingung folgt. □

Ersetzt man in Bedingungen (2.2) und (2.3) die Konstante 1 durch neue Variablen, von denen man fordert, dass sie 1 sind, und ergänzt man eine weitere Variable, die den Wert der beiden Summen aus der Bemerkung enthalten soll, sowie Bedingungen, die fordern, dass diese tatsächlich gleich jeder der beiden obigen Summen und $\leq d$ ist, so erhält man folgendes zum obigen IP äquivalentes ganzzahliges Programm:

$$
\begin{aligned}
\min \quad & \sum_{e \in \tilde{\mathcal{C}}} C_e z_e & \\
\text{so dass} \quad & \sum_{\substack{t_2 \in \mathcal{T} \cup \{0\} \\ (t_1,t_2) \in \tilde{\mathcal{C}}}} z_{(t_1,t_2)} = z_{t_1} & \forall\, t_1 \in \mathcal{T} \\
& \sum_{\substack{t_1 \in \mathcal{T} \cup \{0\} \\ (t_1,t_2) \in \tilde{\mathcal{C}}}} z_{(t_1,t_2)} = z_{t_2} & \forall\, t_2 \in \mathcal{T} \\
& \sum_{t \in \mathcal{T}} z_{(0,t)} = z_0 & \\
& \sum_{t \in \mathcal{T}} z_{(t,0)} = z_0 & \\
& 0 \leq z_e \leq 1 & \forall\, e \in \tilde{\mathcal{C}} \\
& 1 \leq z_t \leq 1 & \forall\, t \in \mathcal{T} \\
& 0 \leq z_0 \leq d & \\
& z_e \in \mathbb{Z} & \forall\, e \in \tilde{\mathcal{C}} \\
& z_t \in \mathbb{Z} & \forall\, t \in \mathcal{T} \\
& z_0 \in \mathbb{Z} &
\end{aligned}
$$

Bringt man die neuen Variablen auf die linke Seite, so erkennt man, dass es sich beim obigen IP um die Formulierung eines Netzwerkflussproblems auf dem Graphen, der entsteht, wenn man im erweiterten Trip-Graphen jeden Knoten durch zwei durch eine Kanten verbundene Knoten ersetzt, handelt. In dem so erzeugten Netzwerk hat jeder Knoten Bedarf 0, die unteren drei Bedingungen lassen sich als Schranken an den Fluss auf den Kanten interpretieren. Die Idee stammt aus [BGAB83] und wird in Abbildung 2.11 veranschaulicht.

Lemma 2.77 ([Sch13])**.** *(VS2) lässt sich in Polynomialzeit lösen.*

Beweis. Wir haben oben gezeigt, dass sich (VS2) in ein Netzwerkflussproblem umformulieren lässt. Da der Bedarf an jedem Knoten und die Schranken an jeder Kante ganzzahlig sind, können wir nach Korollar 2.50 die LP-Relaxation lösen. □

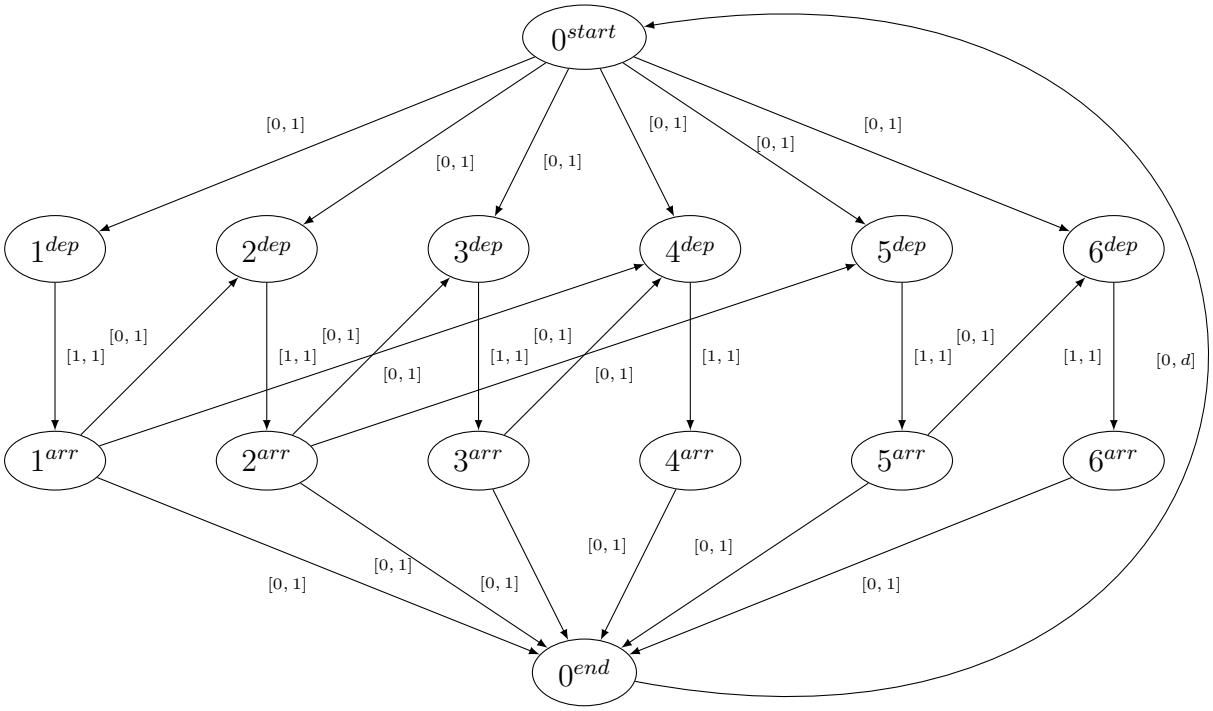

Abb. 2.11: Modellierung von (VS2) als Netzwerkflussproblem am Beispiel des erweiterten Trip-Graphen aus Abbildung 2.9

2.6.3 Bestimmung der kompatiblen Fahrten

In den klassischen Ansätzen wird die Kantenmenge $\mathcal{C}$ aus dem PTN und dem Fahrplan π konstruiert. Dabei wird stets von einem aperiodischen Fahrplan ausgegangen; wurde periodisch geplant, so wird das EAN und der Fahrplan vorher ausgerollt (siehe Abschnitt 2.5.4). Die Menge $\mathcal{C}$ kann dann unterschiedlich definiert werden, z. B.

$$\mathcal{C} := \mathcal{C}_\alpha := \{(t_1, t_2) \in \mathcal{T}^2 \mid \pi_{\mathrm{arr}(t_1)} + d(\mathrm{end}(t_1), \mathrm{start}(t_2)) \leq \pi_{\mathrm{dep}(t_2)}\},$$

wobei $d(v_1, v_2)$ die Fahrzeit von v_1 nach v_2 auf einem kürzesten Weg im PTN ist, oder

$$\mathcal{C} := \mathcal{C}_\beta := \{(t_1, t_2) \in \mathcal{T}^2 \mid \mathrm{end}(t_1) = \mathrm{start}(t_2),\ \pi_{\mathrm{arr}(t_1)} \leq \pi_{\mathrm{dep}(t_2)}\}.$$

Beide oben beschriebene Definitionen haben, falls die Fahrzeit jeder Fahrt > 0 ist, die Eigenschaft, dass $G = (\mathcal{T}, \mathcal{C})$ keine gerichteten Kreise enthalten kann.

3 Integration von Fahrplanoptimierung und Umlaufplanung

Obwohl Umlaufplanung und Fahrplanoptimierung schon seit den 70er- bzw. 80er Jahren untersucht werden, tauchen die ersten dem Autor bekannten Veröffentlichungen zu deren Integration erst vor etwa 10 Jahren auf.

In [LP02, LM07] wird untersucht, wie die Minimierung der Fahrzeuganzahl im Periodic Event Scheduling Problem berücksichtigt werden kann. Dazu werden die Fahr- und Wartezeiten der Fahrzeuge zu den Fahr- und Umsteigezeiten der Fahrgäste in die Zielfunktion passend gewichtet mit aufgenommen. Die Entscheidung, welche Übergänge zwischen Fahrten stattfinden, wird durch zusätzliche Variablen modelliert, die in einer quadratischen Zielfunktion auftauchen. Zur Lösung wird ein genetischer Algorithmus verwendet. Das Problem wird als bikriterielles Optimierungsproblem mit Zielfunktionen Fahrgast- und Fahrzeugwartezeit aufgefasst und es werden für verschiedene Gewichtungen in der Zielfunktion Lösungen gefunden, die die Pareto-Optima approximieren sollen.

In [vvv08] wird ausgehend von einem optimalen periodischen Fahrplan ein Lokaler Suchalgorithmus vorgestellt, bei dem mittels Simulated Annealing in jedem Schritt eine kleine Veränderung am Fahrplan vorgenommen wird und als Goal-Funktion die Kosten des zugehörigen optimalen Umlaufplans verwendet werden. Ein ähnlicher Ansatz wird auch in [GH10] verfolgt, wobei allerdings aperiodisch geplant und als Goal-Funktion eine gewichtete Summe von vier Bewertungskriterien, nämlich Anzahl und Qualität von Umsteigeverbindungen, Gleichmäßigkeit der Fahrzeiten einer Linie, Fahrzeuganzahl und Kosten für Leerfahrten, verwendet wird. Zuvor wird eine multikriterielle Formulierung des Problems angegeben. Im Gegensatz zu unserem Ansatz wird allerdings für feste Fahr- und Wartezeiten geplant. Eine Formulierung des integrierten Fahrplan-Umlaufplan-Problems als bikriterielles Optimierungsproblem wird in [IRRS11] gegeben. Für die Fahrplanoptimierung werden für Paare von Fahrten Synchronisationsknoten vorgegeben, an denen es wünschenswert ist, dass eine Fahrt eine bestimmte Zeit nach einer anderen stattfindet. Dies kann entweder aufgrund eines möglichen Umstiegs der Fall sein, oder um Pulkbildung zu vermeiden. Wird dieser Abstand eingehalten, so heißen die Fahrten synchronisiert. Das Ziel der Fahrplanoptimierung ist es nun, die Anzahl der Synchronisationen zu maximieren. In [CM12] wird ein Modell entwickelt, welches neben den aperiodischen Fahrzeiten und Fahrzeugrouten auch die Zuordnung von Waggons zu Zügen sowie die Bedienhäufigkeiten der Linien berücksichtigt, was dazu führt, dass das Modell recht groß wird.

Ein vollkommen anderer Ansatz zur Integration von Linienplanung, Fahrplanoptimierung und Umlaufplanung wird in [MS09] dargelegt: Die dort vorgestellte Idee ist, im ersten Schritt Fahrten zu bestimmen, diese im zweiten Schritt in Linien zu zerteilen und schließlich im dritten Schritt Fahrzeiten für die Linien festzulegen.

Wir betrachten in dieser Arbeit den Fall, dass der Fahrplan aperiodisch geplant wird, keine Headway-Entscheidungen zu treffen sind und, genauso wie in Abschnitt 2.6, dass alle Fahrzeuge vom gleichen Typ sind und es nur ein Depot gibt. Der Hauptunterschied dieser Arbeit zu fast allen oben erwähnten Ansätzen ist, dass das in dieser Arbeit sowohl das zugrunde liegende Fahrplanoptimierungsproblem als auch das Umlaufplanproblem in polynomieller Zeit lösbar sind. Im Gegensatz dazu ist in allen Ansätzen, in denen ein periodischer Fahrplan geplant wird, das Fahrplanoptimierungsproblem nach Satz 2.60 NP-schwer. Das gilt wie in [IRRS11] angegeben auch für den Fall, dass die Anzahl von Synchronisationen maximiert werden soll. In dem in [CM12] beschriebenen Modell ist das Umlaufplanproblem mit Zuordnung von Zugeinheiten zu Zügen nach [AGKS06, Theorem 1] NP-schwer.

Diese spezielle Eigenschaft des in dieser Arbeit untersuchten Problems zieht zwei Konsequenzen nach sich:

1. Die Komplexität des integrierten Fahrplan-Umlaufplan-Problems ist nicht a priori klar und muss untersucht werden. Wir werden sehen, dass das integrierte Problem selbst dann NP-schwer ist, wenn alle Fahr- und Wartezeiten fest sind.

2. Es ist möglich, eine Heuristik zu entwickeln, die iterativ sowohl das Fahrplan- als auch das Umlaufplanproblem löst. Eine solche Heuristik unterscheidet sich von den bisher beschriebenen, da letztere meist nur das Umlaufplanproblem exakt lösen und beim Fahrplanoptimierungsschritt lediglich zufällige Veränderungen durchführen.

Genauso wie bei den Problemen (VS1) und (VS2) verwenden wir eine Menge $\mathcal{C}$ von Paaren kompatibler Fahrten. Im Unterschied zu den in Abschnitt 2.6.3 beschriebenen Kompatibilitätsbegriffen wird die Menge $\mathcal{C}$ ohne vorliegenden Fahrplan bestimmt. Sie soll alle Paare von Fahrten enthalten, die prinzipiell hintereinander vom gleichen Fahrzeug ausgeführt werden können und kann unterschiedlich gewählt werden, beispielsweise $\mathcal{C} := \mathcal{T}^2$ oder $\mathcal{C} := \{(t_1, t_2) \mid \text{end}(t_1) = \text{start}(t_2)\}$. Dabei kann es im Gegensatz zu Abschnitt 2.6 sein, dass der Trip-Graph $G = (\mathcal{T}, \mathcal{C})$ gerichtete Kreise enthält.

3.1 Exakte Modelle

Das bikriterielle integrierte Fahrplan-Umlaufplan-Problem mit Minimierung der Fahrzeiten und der Fahrzeuganzahl (TTVS1) lässt sich folgendermaßen präzisieren:

Gegeben ist ein von einem PTN und Linienkonzept abgeleitetes EAN $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ mit Fahrtenmenge $\mathcal{T}$, Schranken $L_a \leq U_a$ und Gewichten c_a für alle $a \in \mathcal{A}$, ferner eine Menge $\mathcal{C}$ von Paaren kompatibler Fahrten und für jedes Paar $(t_1, t_2) \in \mathcal{C}$ eine untere und eine obere Schranke $L_{(t_1,t_2)} \leq U_{(t_1,t_2)}$ für die Übergangszeiten.

Gesucht ist ein zulässiger aperiodischer Fahrplan π und ein Umlaufplan $\mathcal{R}$, so dass

$$\forall\, e = (t_1, t_2) \in E(\mathcal{R}) : \pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} \in [L_{(t_1,t_2)}, U_{(t_1,t_2)}] \tag{3.1}$$

und dass gleichzeitig $\sum_{a=(i,j)\in\mathcal{A}} c_a(\pi_j - \pi_i)$ und die Fahrzeuganzahl $|\mathcal{R}|$ minimiert werden.

Das Problem kann folgendermaßen als bikriterielles lineares ganzzahliges Programm mit Variablen $\pi_i,\ i \in \mathcal{E}$ für die Zeiten der Ereignisse und $z_{(t_1,t_2)}$ für $(t_1, t_2) \in \mathcal{C}$ modelliert werden. Letztere geben an, ob Fahrt t_2 direkt nach Fahrt t_1 vom gleichen Fahrzeug durchgeführt wird.

$$\min \left(\sum_{a=(i,j)\in\mathcal{A}} c_a(\pi_j - \pi_i), \quad - \sum_{(t_1,t_2)\in\mathcal{C}} z_{(t_1,t_2)} \right),$$

so dass

$$\begin{aligned}
\pi_j - \pi_i &\leq U_a && \forall\, a = (i,j) \in \mathcal{A} && (3.2)\\
\pi_j - \pi_i &\geq L_a && \forall\, a = (i,j) \in \mathcal{A} && (3.3)\\
\sum_{t_2:(t_1,t_2)\in\mathcal{C}} z_{(t_1,t_2)} &\leq 1 && \forall\, t_1 \in \mathcal{T} && (3.4)\\
\sum_{t_1:(t_1,t_2)\in\mathcal{C}} z_{(t_1,t_2)} &\leq 1 && \forall\, t_2 \in \mathcal{T} && (3.5)\\
\pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} &\leq M_1(1 - z_{(t_1,t_2)}) + U_{(t_1,t_2)} && \forall\, (t_1,t_2) \in \mathcal{C} && (3.6)\\
\pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} &\geq -M_2(1 - z_{(t_1,t_2)}) + L_{(t_1,t_2)} && \forall\, (t_1,t_2) \in \mathcal{C} && (3.7)\\
\pi_i &\in \mathbb{Z} && \forall\, i \in \mathcal{E} && (3.8)\\
z_{(t_1,t_2)} &\in \{0,1\} && \forall\, (t_1,t_2) \in \mathcal{C}, && (3.9)
\end{aligned}$$

wobei M_1, M_2 groß genug gewählt werden (siehe Proposition 3.2).

Aus den Werten der Variablen $z_{(t_1,t_2)}$ kann nach dem folgenden Lemma wie im Beweis zu Lemma 2.74 ein Umlaufplan mit $|\mathcal{R}| = |\mathcal{T}| - \sum_{(z_1,z_2)\in\mathcal{C}} z_{(t_1,t_2)}$ bestimmt werden.

Lemma 3.1. *Sei $G = (\mathcal{T}, \mathcal{C})$ und seien folgende Voraussetzungen erfüllt:*

1. *Für jede Fahrt $t \in \mathcal{T}$ ist $\sum_{a \in \mathcal{A}_t} L_a > 0$.*

2. *Für jeden Übergang $(t_1, t_2) \in \mathcal{C}$ gilt $L_{(t_1,t_2)} \geq 0$.*

Ist dann (π, z) eine zulässige Lösung des obigen ganzzahligen linearen Programms, so gibt es genau einen Umlaufplan mit

$$E(\mathcal{R}) = \mathcal{C}' := \{(t_1, t_2) \in \mathcal{C} \mid z_{(t_1,t_2)} = 1\}.$$

Beweis. Wir wollen den zweiten Teil des Beweises von Lemma 2.73 auf obigen Fall übertragen. Die Bedingungen (3.4) und (3.5) sorgen dafür, dass jede Fahrt höchstens eine Vorgänger- und höchstens eine Nachfolgerfahrt hat. Im Unterschied zur Situation in Lemma 2.73 kann G gerichtete Kreise enthalten. Im Beweis wird allerdings nur verwendet, dass $(\mathcal{T}, \mathcal{C}')$ keine gerichteten Kreise enthält. Dies ist der Fall, denn angenommen $(t_0, t_1, \ldots, t_n = t_0)$ wäre die Knotenfolge eines solchen Kreises, so wäre

$$\begin{aligned} 0 &< \sum_{k=1}^{n} \sum_{a \in \mathcal{A}_{t_k}} L_a + \sum_{k=1}^{n} L_{(t_{k-1},t_k)} \\ &\overset{(3.3),(3.7)}{\leq} \sum_{k=1}^{n} \sum_{a=(i,j) \in \mathcal{A}_{t_k}} (\pi_j - \pi_i) + \sum_{k=1}^{n} (\pi_{\mathrm{dep}(t_k)} - \pi_{\mathrm{arr}(t_{k-1})}) \\ &= \sum_{k=1}^{n} (\pi_{\mathrm{arr}(t_k)} - \pi_{\mathrm{dep}(t_k)}) + \sum_{k=1}^{n} (\pi_{\mathrm{dep}(t_k)} - \pi_{\mathrm{arr}(t_{k-1})}) = 0. \end{aligned}$$

Somit lässt sich der Beweis von Lemma 2.73 übertragen. Die Eindeutigkeit des Umlaufplans folgt ebenfalls daraus, dass es an jedem Knoten nur höchstens eine ausgewählte eingehende und ausgewählte ausgehende Kante gibt. Somit gibt es nur eine Möglichkeit, die Knoten zu Wegen zuzuordnen, welche genau die ausgewählten Kanten enthalten. □

Bedingungen (3.6) und (3.7) aus der IP-Formulierung codieren Bedingung (3.1): $(t_1, t_2) \in E(\mathcal{R})$ bedeutet gerade, dass $z_{(t_1,t_2)} = 1$ ist. In diesem Fall ist $M_i(1 - z_{(t_1,t_2)}) = 0,\ i \in \{1, 2\}$ und folglich können die beiden Bedingungen nur erfüllt sein, wenn $\pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} \in [L_{(t_1,t_2)}, U_{(t_1,t_2)}]$ ist. Ist umgekehrt $(t_1, t_2) \in \mathcal{C} \setminus E(\mathcal{R})$, so ist die zugehörige Variable $z_{(t_1,t_2)} = 0$ und somit $M_i(1 - z_{(t_1,t_2)}) = M_i,\ i \in \{1, 2\}$. Sind M_1, M_2 *groß genug*, so sind die Bedingungen unabhängig von den Werten von $\pi_{\mathrm{dep}(t_2)}$ und $\pi_{\mathrm{arr}(t_1)}$ erfüllt.

Proposition 3.2. *Sei $\mathcal{N}$ ein zusammenhängendes EAN und $\mathcal{W}(\mathrm{arr}(t_1), \mathrm{dep}(t_2))$ die Menge aller Wege in $\mathcal{N}$ von $\mathrm{arr}(t_1)$ nach $\mathrm{dep}(t_2)$. Dann sind*

$$M_1 \geq \max_{(t_1,t_2)\in\mathcal{C}} \left(\min_{W\in\mathcal{W}(\mathrm{arr}(t_1),\mathrm{dep}(t_2))} \Big(\sum_{a\in W^+} U_a - \sum_{a\in W^-} L_a \Big) - U_{(t_1,t_2)} \right)$$

$$M_2 \geq \max_{(t_1,t_2)\in\mathcal{C}} \left(\min_{W\in\mathcal{W}(\mathrm{arr}(t_1),\mathrm{dep}(t_2))} \Big(\sum_{a\in W^-} U_a - \sum_{a\in W^+} L_a \Big) + L_{(t_1,t_2)} \right)$$

groß genug.

Beweis. Es ist zu zeigen, dass, wenn die Bedingungen (3.2) bis (3.5) erfüllt sind und für $(t_1, t_2) \in \mathcal{C}$ die Variable $z_{(t_1,t_2)} = 0$ ist, die Bedingungen (3.6) und (3.7) für (t_1, t_2) automatisch ebenfalls erfüllt sind.

Da $\mathcal{N}$ zusammenhängend ist, gibt es einen Weg $W = (i_0, a_1, \ldots, a_n, i_n)$ von $\mathrm{arr}(t_1)$ nach $\mathrm{dep}(t_2)$. Dafür gilt:

$$\begin{aligned}\pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} &= \sum_{j=0}^{n-1} (\pi_{i_{j+1}} - \pi_{i_j}) \\ &= \sum_{j:(i_j,i_{j+1})\in E(W)} (\pi_{i_{j+1}} - \pi_{i_j}) - \sum_{j:(i_{j+1},i_j)\in E(W)} (\pi_{i_j} - \pi_{i_{j+1}}).\end{aligned}$$

Nun werden die Summanden der beiden Summen mithilfe der Bedingungen (3.2) und (3.3) auf zwei Arten abgeschätzt:

1. Es gilt $\pi_{i_{j+1}} - \pi_{i_j} \overset{(3.2)}{\leq} U_{(i_j,i_{j+1})}$ für alle j mit $a = (i_j, i_{j+1}) \in W^+ \subset \mathcal{A}$ und $\pi_{i_j} - \pi_{i_{j+1}} \overset{(3.3)}{\geq} L_{(i_{j+1},i_j)}$ für alle j mit $a = (i_j, i_{j+1}) \in W^- \subset \mathcal{A}$ und somit

   $$\pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} \leq \sum_{a\in W^+} U_a - \sum_{a\in W^-} L_a.$$

   Da dies für alle Wege $W \in \mathcal{W}(\mathrm{arr}(t_1), \mathrm{dep}(t_2))$ gilt, folgt:

   $$\begin{aligned}\pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} &\leq \min_{W\in\mathcal{W}(\mathrm{arr}(t_1),\mathrm{dep}(t_2))} \Big(\sum_{a\in W^+} U_a - \sum_{a\in W^-} L_a \Big) - U_{(t_1,t_2)} + U_{(t_1,t_2)} \\ &\leq \max_{(t_1',t_2')\in\mathcal{C}} \left(\min_{W\in\mathcal{W}(\mathrm{arr}(t_1'),\mathrm{dep}(t_2'))} \Big(\sum_{a\in W^+} U_a - \sum_{a\in W^-} L_a \Big) - U_{(t_1',t_2')} \right) + U_{(t_1,t_2)} \\ &\leq M_1 + U_{(t_1,t_2)}.\end{aligned}$$

2. Es gilt $\pi_{i_{j+1}} - \pi_{i_j} \overset{(3.3)}{\geq} L_{(i_j,i_{j+1})}$ für alle j mit $a = (i_j, i_{j+1}) \in W^+ \subset \mathcal{A}$ und $\pi_{i_j} - \pi_{i_{j+1}} \overset{(3.2)}{\leq} U_{(i_{j+1},i_j)}$ für alle j mit $a = (i_j, i_{j+1}) \in W^- \subset \mathcal{A}$ und deshalb

$$\pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} \geq \sum_{a \in W^+} L_a - \sum_{a \in W^-} U_a.$$

Da dies für alle Wege $W \in \mathcal{W}(\mathrm{arr}(t_1), \mathrm{dep}(t_2))$ gilt, ergibt sich:

$$\begin{aligned}
\pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} &\geq \max_{W \in \mathcal{W}(\mathrm{arr}(t_1),\mathrm{dep}(t_2))} \Big(\sum_{a \in W^+} L_a - \sum_{a \in W^-} U_a \Big) \\
&= - \left(\min_{W \in \mathcal{W}(\mathrm{arr}(t_1),\mathrm{dep}(t_2))} \Big(\sum_{a \in W^-} U_a - \sum_{a \in W^+} L_a \Big) + L_{(t_1,t_2)} \right) + L_{(t_1,t_2)} \\
&\geq - \max_{(t_1',t_2') \in \mathcal{C}} \left(\min_{W \in \mathcal{W}(\mathrm{arr}(t_1'),\mathrm{dep}(t_2'))} \Big(\sum_{a \in W^-} U_a - \sum_{a \in W^+} L_a \Big) + L_{(t_1',t_2')} \right) + L_{(t_1,t_2)} \\
&= -M_2 + L_{(t_1,t_2)}.
\end{aligned}$$

□

Bemerkung 3.3. Die unteren Schranken für M_1 und M_2 lassen sich in polynomieller Zeit berechnen.

Beweis. Sei $\mathcal{N}'$ das in Lemma 2.58 eingeführte Hilfsnetzwerk zu $(\mathcal{N}, L, U)$ mit Kantenlängen $\mu_{a'}$. Sei $(t_1, t_2) \in \mathcal{C}$. Dann ist

$$\min_{w \in \mathcal{W}(\mathrm{arr}(t_1),\mathrm{dep}(t_2))} \Big(\sum_{a \in w^+} U_a - \sum_{a \in w^-} L_a \Big)$$

gerade die Länge eines kürzesten gerichteten Weges von $\mathrm{arr}(t_1)$ nach $\mathrm{dep}(t_2)$ in $\mathcal{N}'$. Genauso ist

$$\min_{W \in \mathcal{W}(\mathrm{arr}(t_1),\mathrm{dep}(t_2))} \Big(\sum_{a \in W^-} U_a - \sum_{a \in W^+} L_a \Big)$$

die Länge eines kürzesten gerichteten Weges von $\mathrm{dep}(t_2)$ nach $\mathrm{arr}(t_1)$. Beide lassen sich z. B. mit dem Bellman-Ford-Algorithmus in polynomieller Zeit berechnen. Die Gesamtlaufzeit liegt dann in

$$\mathcal{O}(|\mathcal{E}||\mathcal{A}||\mathcal{C}|) \subseteq \mathcal{O}(|\mathcal{E}||\mathcal{A}||\mathcal{T}|^2).$$

□

Beispiel. Wir betrachten das EAN $\mathcal{N}$ in Abbildung 3.1, welches aus zwei Fahrten besteht, zwischen denen es eine Umsteigeaktivität gibt. Die gestrichelten Kanten stellen mögliche Übergänge zwischen den Fahrten dar und gehören formal nicht zu $\mathcal{N}$. Die obere Fahrt sei τ_1, die untere τ_2 mit $\mathcal{C} := \mathcal{T}^2$ und $L_{(\tau_1,\tau_2)} = L_{(\tau_2,\tau_1)} = 1,\ U_{(\tau_1,\tau_2)} = U_{(\tau_2,\tau_1)} = 60$.

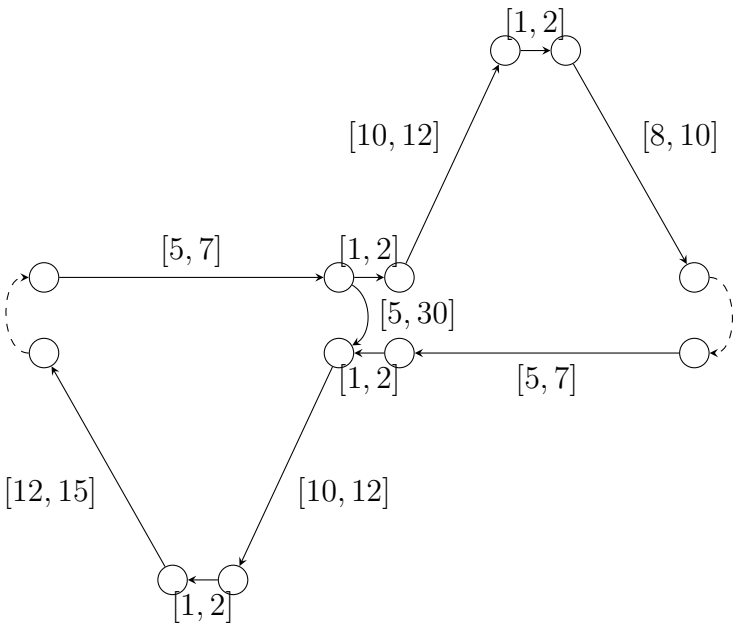

Abb. 3.1: Beispielhaftes EAN mit potenziellen Übergängen

Wir berechnen die Schranke an M_1 aus der obigen Proposition: Sei W_1 der einzige Weg in $\mathcal{N}$ von $\text{arr}(\tau_1)$ nach $\text{dep}(\tau_2)$ und W_2 der einzige Weg von $\text{arr}(\tau_2)$ nach $\text{dep}(\tau_1)$. Dann erhalten wir:

$$\begin{aligned} M_1 &\geq \max\left\{ \sum_{a\in W_1^+} U_a - \sum_{a\in W_1^-} L_a - U_{(\tau_1,\tau_2)}, \sum_{a\in W_2^+} U_a - \sum_{a\in W_2^-} L_a - U_{(\tau_2,\tau_1)} \right\} \\ &= \max\{4-60, -33-60\} = \max\{-56,-93\} = -56 \end{aligned}$$

Für M_2 erhalten wir:

$$\begin{aligned} M_2 &\geq \max\left\{ \sum_{a\in W_1^-} U_a - \sum_{a\in W_1^+} L_a + L_{(\tau_1,\tau_2)}, \sum_{a\in W_2^-} U_a - \sum_{a\in W_2^+} L_a + L_{(\tau_2,\tau_1)} \right\} \\ &= \max\{30+1, 66+1\} = \max\{31,67\} = 67 \end{aligned}$$

Es ist leicht einsichtig, dass für einen zulässigen Fahrplan gelten muss:

$$\begin{aligned} \pi_{\text{dep}(\tau_2)} - \pi_{\text{arr}(\tau_1)} &\in [-30, 4] \\ \pi_{\text{dep}(\tau_1)} - \pi_{\text{arr}(\tau_2)} &\in [-66, -33] \end{aligned}$$

Da $[-66,-33] \cap [1,60] = \emptyset$, kann der entsprechende Übergang in keinem zulässigen Fahrplan ausgewählt werden. Man könnte also die Kante (τ_2, τ_1) in einem Preprocessing-Schritt aus $\mathcal{C}$ entfernen.

Lemma 3.4 (Reduktionslemma)**.** *Seien t_1, t_2 zwei Fahrten mit $(t_1, t_2) \in \mathcal{C}$. Gibt es einen Weg W im EAN von $\operatorname{arr}(t_1)$ nach $\operatorname{dep}(t_2)$, so dass*

$$\sum_{a \in W^+} L_a - \sum_{a \in W^-} U_a > U_{(t_1,t_2)} \quad \textit{oder} \quad \sum_{a \in W^+} U_a - \sum_{a \in W^-} L_a < L_{(t_1,t_2)},$$

so kann das Paar (t_1, t_2) aus der Kompatibilitätsrelation gestrichen werden, ohne Zulässigkeit und Zielfunktionswert zu verändern.

Beweis. Das Entfernen des Übergangs (t_1, t_2) kann nur die Zulässigkeit oder den Zielfunktionswert verändern, wenn es eine zulässige Lösung $(\pi, \mathcal{R})$ gibt mit $(t_1, t_2) \in E(\mathcal{R})$. Diese bleibt zulässig, wenn zum EAN $\mathcal{N}$ eine Aktivität $(\operatorname{arr}(t_1), \operatorname{dep}(t_2))$ mit Spann $[L_{t_1,t_2}, U_{t_1,t_2}]$ hinzugefügt wird. Dazu lässt sich das Hilfsnetzwerk $\mathcal{N}'$ mit Kantenlängen $\mu_{a'}$ bilden.

Ist oben die erste Ungleichung erfüllt, so hat der gerichtete Kreis in $\mathcal{N}'$, der die Umlaufaktivität in positiver und den Weg W in umgekehrter Richtung durchläuft, die Länge

$$\underbrace{\sum_{a \in W^+} (-L_a) + \sum_{a \in W^-} U_a}_{= -\left(\sum_{a \in W^+} L_a - \sum_{a \in W^-} U_a\right)} + U_{(t_1,t_2)} < 0.$$

Im Fall der zweiten Ungleichung hat der umgekehrte Kreis die Länge

$$\sum_{a \in w^+} U_a + \sum_{a \in w^-} (-L_a) - L_{(t_1,t_2)} < 0.$$

In beiden Fällen kann es für das um die Umlaufaktivität erweiterte EAN nach Satz 2.59 keinen zulässigen Fahrplan geben; es gibt also keinen zulässigen Fahrplan in $\mathcal{N}$, der diesen Übergang verwendet. □

Wir betrachten nun das integrierte Fahrplan-Umlaufplan-Problem mit Minimierung der Fahrzeiten und der Umlaufkosten (TTVS2). Dabei sind zusätzlich zu den Eingabedaten für (TTVS1) Kosten c_{t_1,t_2} für alle $(t_1, t_2) \in \mathcal{C}$, $c_{0,t}$ und $c_{t,0}$ für alle $t \in \mathcal{T}$ und α, wie in Abschnitt 2.6.2 beschrieben, sowie eine maximale Fahrzeuganzahl d gegeben. Gesucht ist nun ein zulässiger aperiodischer Fahrplan π und ein Umlaufplan $\mathcal{R}$, so dass Bedingung (3.1) erfüllt ist, $|\mathcal{R}| \leq d$ ist und die Kosten $\sum_{R \in \mathcal{R}} C(R)$ (siehe Abschnitt 2.6.2) minimal sind.

Wir verwenden zur Formulierung von (TTVS2) den in Abschnitt 2.6.2 eingeführten erweiterten Trip-Graphen $\tilde{G} = (\mathcal{T} \cup \{0\}, \tilde{\mathcal{C}})$ und die zugehörigen Kosten. Damit kann (TTVS2) folgendermaßen als IP formuliert werden, wobei analog zu (VS2) neben den Variablen π_i die Variablen z_e für $e \in \tilde{\mathcal{C}}$ verwendet werden. Die Betriebskosten werden dabei als nur von der zurückzulegenden Entfernung abhängig angenommen. Die Zeiten der Fahrzeugübergänge und der Fahrten werden bei der Berechnung der Betriebskosten nicht berücksichtigt, da das Verhältnis von Übergangszeit zu Linienfahrzeit für die Kosten nicht von Belang ist und die an einem Tag für ein Fahrzeug anfallenden Personalkosten in die Fahrzeugkosten α integriert werden können.

$$\min \left(\sum_{a=(i,j)\in\mathcal{A}} c_a(\pi_j - \pi_i), \quad \sum_{e\in\tilde{\mathcal{C}}} C_e z_e \right)$$

so dass

$$\pi_j - \pi_i \leq U_a \qquad \forall\, a = (i,j) \in \mathcal{A} \tag{3.10}$$

$$\pi_j - \pi_i \geq L_a \qquad \forall\, a = (i,j) \in \mathcal{A} \tag{3.11}$$

$$\sum_{\substack{t_2\in\mathcal{T}\cup\{0\} \\ (t_1,t_2)\in\mathcal{C}}} z_{(t_1,t_2)} = 1 \qquad \forall\, t_1 \in \mathcal{T} \tag{3.12}$$

$$\sum_{\substack{t_1\in\mathcal{T}\cup\{0\} \\ (t_1,t_2)\in\mathcal{C}}} z_{(t_1,t_2)} = 1 \qquad \forall\, t_2 \in \mathcal{T} \tag{3.13}$$

$$\sum_{t\in\mathcal{T}} z_{(0,t)} \leq d \tag{3.14}$$

$$\pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} \leq M_1(1 - z_{(t_1,t_2)}) + U_{(t_1,t_2)} \qquad \forall\, (t_1,t_2) \in \tilde{\mathcal{C}} \tag{3.15}$$

$$\pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} \geq -M_2(1 - z_{(t_1,t_2)}) + L_{(t_1,t_2)} \qquad \forall\, (t_1,t_2) \in \tilde{\mathcal{C}} \tag{3.16}$$

$$\pi_i \in \mathbb{Z} \qquad \forall\, i \in \mathcal{E} \tag{3.17}$$

$$z_e \in \{0,1\} \qquad \forall\, e \in \tilde{\mathcal{C}}. \tag{3.18}$$

Bemerkung 3.5. Unter den Voraussetzungen von Lemma 3.1 lässt sich aus einer zulässigen Lösung des obigen ganzzahligen Programms ein eindeutiger erweiterter Umlaufplan $\tilde{\mathcal{R}}$ konstruieren mit

$$E(\tilde{\mathcal{R}}) = \mathcal{C}' := \{e \in \tilde{\mathcal{C}} \mid z_e = 1\}.$$

Der Beweis dafür verläuft analog zum Beweis von Lemma 3.1.

Wie in [Sch07] beschrieben, gibt es verschiedene Lösungsansätze für multikriterielle Probleme. Das vorliegende Problem kann in jede der dort beschriebenen Formen umgewandelt werden. Exemplarisch wird die Formulierung dafür angegeben, dass das Budget B als Schranke für die zweite Zielfunktion vorgegeben wird und die Reisezeiten unter Einhaltung dieser Schranke minimiert werden sollen. Die IP-Formulierung lautet

$$\min \sum_{a=(i,j)\in\mathcal{A}} c_a(\pi_j - \pi_i),$$

so dass

$$\begin{aligned}
\pi_j - \pi_i &\leq U_a && \forall\, a=(i,j)\in\mathcal{A}\\
\pi_j - \pi_i &\geq L_a && \forall\, a=(i,j)\in\mathcal{A}\\
\sum_{\substack{t_2\in\mathcal{T}\cup\{0\}\\(t_1,t_2)\in\mathcal{C}}} z_{(t_1,t_2)} &= 1 && \forall\, t_1\in\mathcal{T}\\
\sum_{\substack{t_1\in\mathcal{T}\cup\{0\}\\(t_1,t_2)\in\mathcal{C}}} z_{(t_1,t_2)} &= 1 && \forall\, t_2\in\mathcal{T}\\
\sum_{t\in\mathcal{T}} z_{(0,t)} &\leq d\\
\pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} &\leq M_1(1-z_{(t_1,t_2)}) + U_{(t_1,t_2)} && \forall\,(t_1,t_2)\in\tilde{\mathcal{C}}\\
\pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} &\geq -M_2(1-z_{(t_1,t_2)}) + L_{(t_1,t_2)} && \forall\,(t_1,t_2)\in\tilde{\mathcal{C}}\\
\sum_{e\in\tilde{\mathcal{C}}} C_e z_e &\leq B\\
\pi_i &\in \mathbb{Z} && \forall\, i\in\mathcal{E}\\
z_e &\in \{0,1\} && \forall\, e\in\tilde{\mathcal{C}}.
\end{aligned}$$

Wir untersuchen nun die Komplexität der Entscheidungsversionen (siehe Definition 2.53) von (TTVS1) und (TTVS2).

Satz 3.6. *Die Entscheidungsversionen der beiden Probleme (TTVS1) und (TTVS2) sind NP-vollständig, selbst dann, wenn die Dauer aller Fahrten festgelegt ist.*

Beweis. Die Entscheidungsversionen der Probleme (TTVS1) und (TTVS2) liegen offensichtlich in **NP**, denn für einen gegebenen Umlaufplan und Fahrplan lassen sich Zulässigkeit und Zielfunktionswert in polynomieller Zeit verifizieren. Außerdem ist die Codierung beider Pläne nicht länger als die Codierung der Eingabe.

Zum Beweis der NP-Schwere reduzieren wir im ersten Schritt, in Anlehnung an den Beweis für die NP-Vollständigkeit des „delay management problem with integrated rolling stock circulation“ (DMC) aus [Sch09], die Entscheidungsversion des Problems (MIN-SAT) auf die Entscheidungsversion von (TTVS1) und im zweiten diese dann auf die Entscheidungsversion von (TTVS2).

Wir geben zunächst das Problem (MIN-SAT) wieder:

Gegeben ist eine Menge $\mathfrak{X} := \{x_1,\dots,x_n\}$ von Booleschen Variablen und eine Menge $\mathfrak{C} := \{C_1,\dots,C_m\}$ disjunktiver Klauseln, d. h.

$$C_j = l_1^{(j)} \vee l_2^{(j)} \vee \dots \vee l_{k_j}^{(j)}, \text{ mit } l_r^{(j)} \in \mathfrak{L} := \{x_1,\dots,x_n,\overline{x_1},\dots,\overline{x_n}\},\ r=1,\dots,k_j, j=1,\dots,m.$$

Gesucht ist eine Variablenbelegung $v\colon \mathfrak{X} \to \{\text{true}, \text{false}\}$, so dass möglichst wenige Klauseln erfüllt sind, genauer: so dass für möglichst viele $C \in \mathfrak{C}$ gilt: $\overline{v}(C) = \text{false}$, wobei $\overline{v}$ die von v induzierte Auswertungsfunktion $\text{Form}(\mathfrak{X}) \to \{\text{true}, \text{false}\}$ ist (siehe [EMC+01, Def. 13.3.1]).

Einen Beweis für die NP-Vollständigkeit von (MIN-SAT) liefert z. B. [KKM94].

Die Entscheidungsversion von (MIN-SAT) enthält als zusätzliche Eingabe die Schranke k für den Zielfunktionswert. Wir konstruieren nun aus einer Instanz von (MIN-SAT) und der Schranke k eine Instanz der Entscheidungsversion von (TTVS1) folgendermaßen: Sei $z \in V$ die Station, an der die Fahrzeugübergänge möglich sind und seien $u, v_1, \ldots, v_n$, $w_1, \ldots, w_m \in V$ weitere Stationen.

- Wir definieren eine Fahrt 0 von u nach z.
- Für jede Variable $x_i \in \mathfrak{X}$ definieren wir eine Fahrt true_i von z nach v_i sowie zwei Fahrten $x_i, \overline{x_i}$ von v_i nach z.
- Für jede Klausel $C_j \in \mathfrak{C}$ wird eine weitere Fahrt von z nach w_j erzeugt.

Dadurch sind die Mengen der Fahrten, der Ereignisse und der Fahraktivitäten folgendermaßen bestimmt:

$$\begin{aligned} \mathcal{T} &= \{0\} \cup \{\text{true}_i, x_i, \overline{x}_i \mid i = 1, \ldots, n\} \cup \{C_j \mid j = 1, \ldots, m\}, \\ \mathcal{E} &= \{\text{dep}(t), \text{arr}(t) \mid t \in \mathcal{T}\}, \\ \mathcal{A}_{drive} &= \{(\text{dep}(t), \text{arr}(t)) \mid t \in \mathcal{T}\}. \end{aligned}$$

Als Schranken für alle Fahraktivitäten $a \in \mathcal{A}_{drive}$ setzen wir $L_a := U_a := 0$. Das bisherige EAN ist in Abbildung 3.2 dargestellt.

- Wir definieren die Menge kompatibler Fahrten folgendermaßen:

  $$\mathcal{C} := \{(t_1, t_2) \mid \exists\, i \in \{1, \ldots, n\} : t_1 = \text{true}_i \wedge t_2 \in \{x_i, \overline{x_i}\}\}$$

  und setzen als Schranken $L_{(t_1,t_2)} := U_{(t_1,t_2)} := 0$ für alle $(t_1, t_2) \in \mathcal{C}$. Die potenziellen Übergänge sind beispielhaft für $i = 1$ in Abbildung 3.3 als gestrichelte Kanten dargestellt.

- Als Aktivitätenmenge definieren wir:

  $$\mathcal{A} := \mathcal{A}_{drive} \cup \mathcal{A}_{change},$$

  wobei

  $$\mathcal{A}_{change} := \mathcal{A}_0 \cup \mathcal{A}_{Klausel} \cup \mathcal{A}_{Bewertung}.$$

  Dabei ist

  $$\mathcal{A}_0 := \{(\text{arr}(0), \text{dep}(\text{true}_i)) \mid i = 1, \ldots, n\}$$

  mit den Schranken $L_a := U_a := 1$ für alle $a \in \mathcal{A}_0$ (siehe Abbildung 3.4),

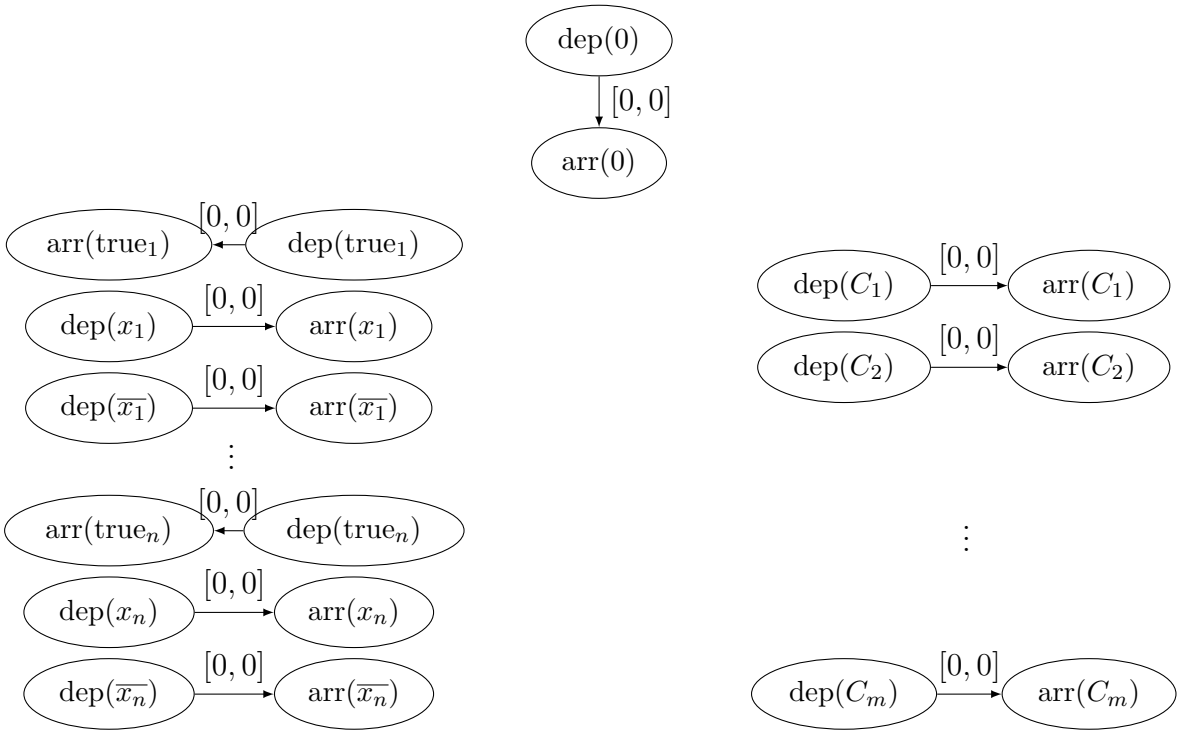

Abb. 3.2: Ereignisse und Fahraktivitäten in der Konstruktion

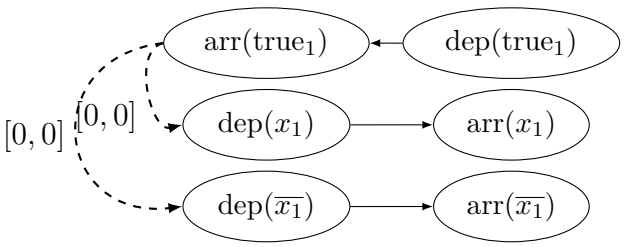

Abb. 3.3: Kompatibilitäten in der Konstruktion

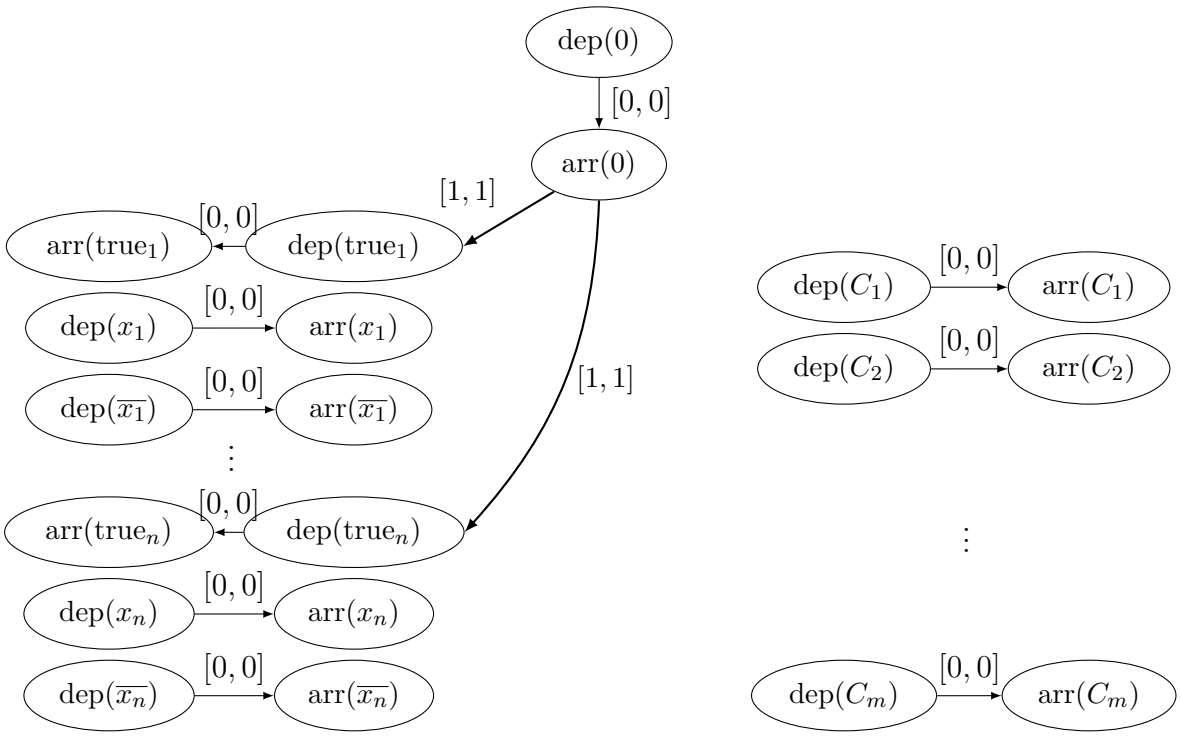

Abb. 3.4: Aktivitäten $\mathcal{A}_0$ in der Konstruktion

$$\mathcal{A}_{Klausel} := \{(\text{arr}(l), \text{dep}(C_j)) \mid l \in \mathfrak{L}, j \in \{1 \ldots, m\}, l \in C_j\}$$

mit den Schranken $L_a := 0,\ U_a := 1$ für alle $a \in \mathcal{A}_{Klausel}$ (siehe Abbildung 3.5) und

$$\mathcal{A}_{Bewertung} := \{(\text{arr}(0), \text{dep}(C_j)) \mid j = 1, \ldots, m\}$$

mit den Schranken $L_a := 0,\ U_a := 1$ für $a \in \mathcal{A}_{Bewertung}$ (siehe Abbildung 3.6).

- Als Koeffizienten der Zielfunktion wählen wir

$$c_a := \begin{cases} 1, & \text{für } a \in \mathcal{A}_{Bewertung}, \\ 0, & \text{sonst.} \end{cases}$$

- Als Schranken k_1 für die Reisezeiten und k_2 für die Fahrzeuganzahl wählen wir

$$k_1 := k$$
$$k_2 := 2n + m + 1.$$

Behauptung: Es gibt genau dann eine zulässige Lösung $(\pi, \mathcal{R})$ des integrierten Fahrplan-Umlaufplan-Problems, so dass $\sum_{a=(i,j)\in\mathcal{A}} c_a(\pi_j - \pi_i) \leq k_1$ und $|\mathcal{R}| \leq k_2$, wenn es eine Variablenbelegung zu $\mathfrak{C}$ mit höchstens k erfüllten Klauseln gibt.

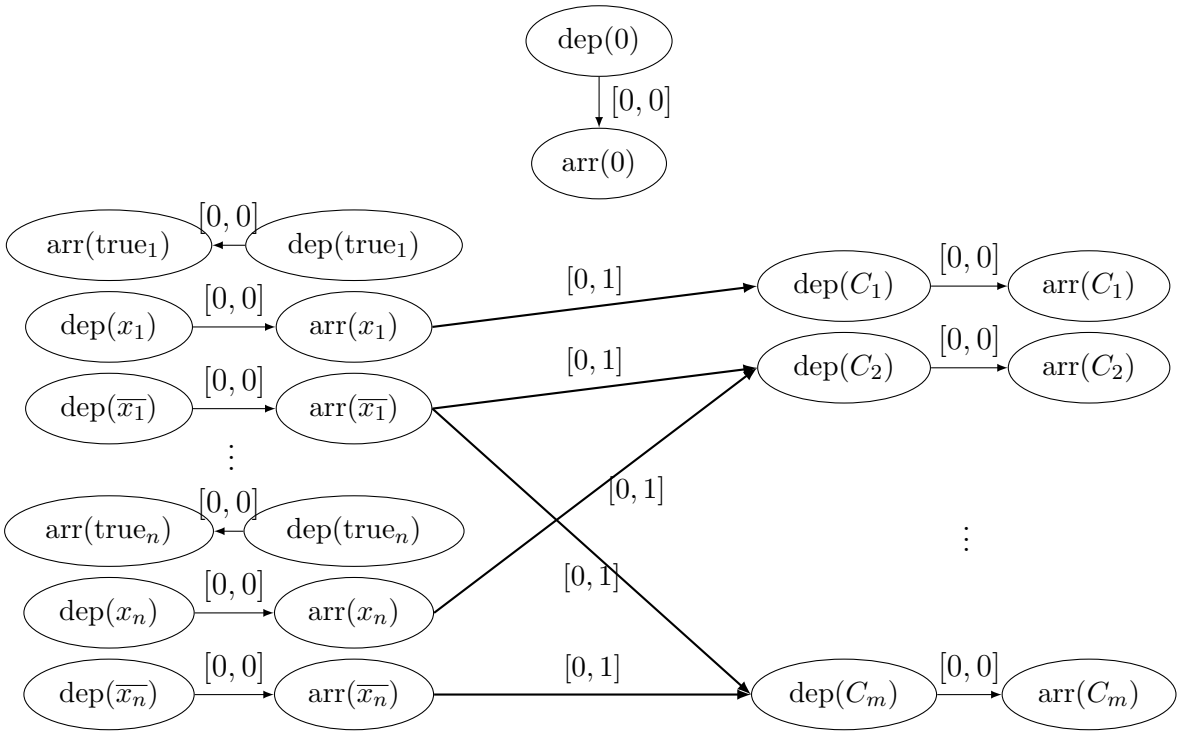

Abb. 3.5: Beispielhafte Aktivitäten $\mathcal{A}_{Klausel}$ in der Konstruktion

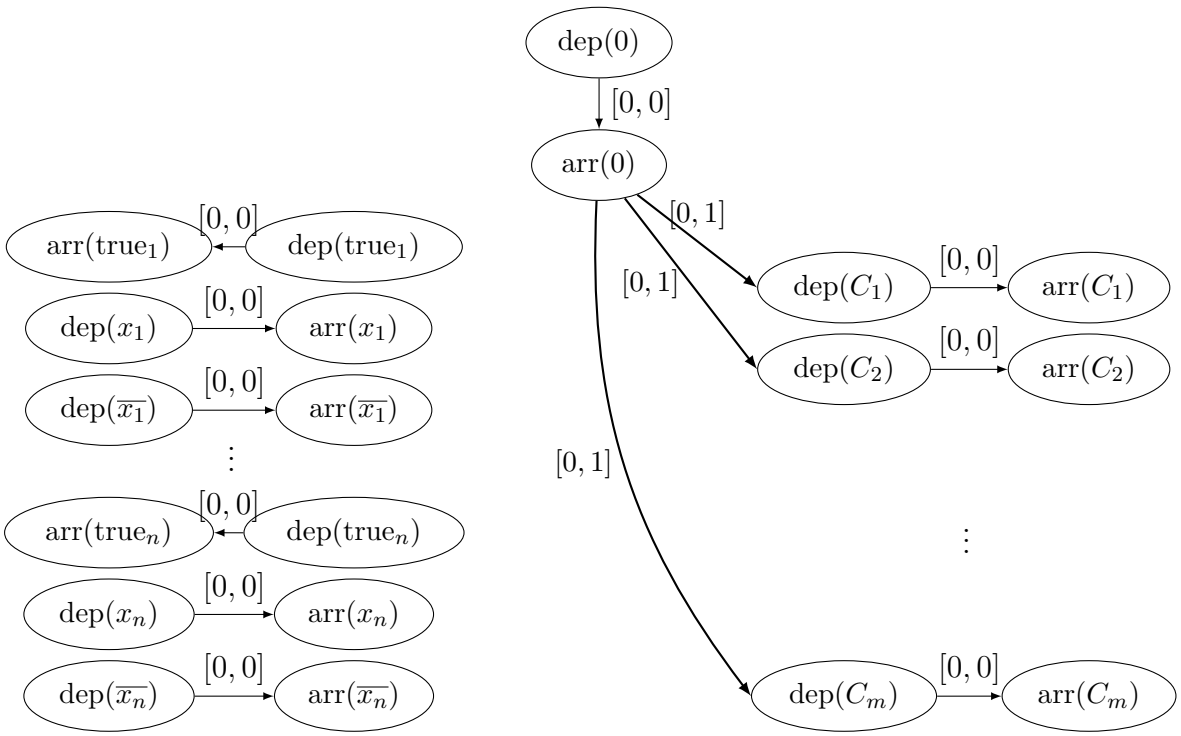

Abb. 3.6: Aktivitäten $\mathcal{A}_{Bewertung}$ in der Konstruktion

„⇒“ Gegeben sei ein zulässiger Fahrplan π mit Zielfunktionswert $\leq k_1$ und ein dazu passender Umlaufplan $\mathcal{R}$ mit $|\mathcal{R}| \leq k_2 = 2n + m + 1$. Da die Fahrten 0 und $C_j,\ j \in \{1, \ldots, m\}$ mit keiner anderen kompatibel sind, benötigen diese bereits $m + 1$ Fahrzeuge. Es bleiben also noch $2n$ Fahrzeuge für die $3n$ anderen Fahrten. Deshalb muss für jedes $i \in \{1, \ldots, n\}$ stets einer der beiden Übergänge (true_i, x_i) oder $(\text{true}_i, \overline{x}_i)$ ausgewählt worden sein. Wir definieren nun daraus die Variablenbelegung

$$v(x_i) := \begin{cases} \text{true}, & \text{falls } (\text{true}_i, x_i) \in E(\mathcal{R}), \\ \text{false}, & \text{falls } (\text{true}_i, \overline{x}_i) \in E(\mathcal{R}). \end{cases}$$

Nach Voraussetzung gilt:

$$\sum_{a=(i,j)\in\mathcal{A}} c_a(\pi_j - \pi_i) = \sum_{j=1}^{m} 1 \cdot (\pi_{\text{dep}(C_j)} - \pi_{\text{arr}(0)}) \leq k_1 = k.$$

Da $\pi_{\text{dep}(C_j)} - \pi_{\text{arr}(0)} \in \{0, 1\}$ für $j = 1, \ldots, m$ ist, ist die Differenz für höchstens k Klauseln Eins und für mindestens $m - k$ Klauseln $C_{j_1}, \ldots, C_{j_{m-k}}$ ist die Differenz Null. Sei nun $j \in \{j_1, \ldots, j_{m-k}\}$, d. h.

$$\pi_{\text{dep}(C_j)} - \pi_{arr(0)} = 0$$

und sei $l_i \in \{x_i, \overline{x_i}\}$ ein Literal mit $l_i \in C_j$. Dann ist $(\text{true}_i, l_i) \notin E(\mathcal{R})$, denn sonst wäre

$$\begin{aligned} &\pi_{\text{dep}(C_j)} - \pi_{\text{arr}(0)} \\ = &\underbrace{(\pi_{\text{dep}(C_j)} - \pi_{\text{arr}(l_i)})}_{\in\{0,1\},\ \text{da } (\text{arr}(l_i),\text{dep}(C_j))\in\mathcal{A}_{Klausel}} + \underbrace{(\pi_{\text{arr}(l_i)} - \pi_{\text{dep}(l_i)})}_{=0,\ \text{da } (\text{dep}(l_i),\text{arr}(l_i))\in\mathcal{A}_{drive}} + \underbrace{(\pi_{\text{dep}(l_i)} - \pi_{\text{arr}(\text{true}_i)})}_{=0,\ \text{da } (\text{true}_i,l_i)\in E(\mathcal{R})} \\ &+ \underbrace{(\pi_{(A,\text{arr}(\text{true}_i)} - \pi_{\text{dep}(\text{true}_i)})}_{=0,\ \text{da } (\text{dep}(\text{true}_i),\text{arr}(\text{true}_i))\in\mathcal{A}_{drive}} + \underbrace{(\pi_{\text{dep}(\text{true}_i)} - \pi_{\text{arr}(0)})}_{=1,\ \text{da } (\text{arr}(0),\text{dep}(\text{true}_i))\in\mathcal{A}_0} \geq 1 \end{aligned}$$

Ist $l_i = x_i$, so ist $\overline{v}(l_i) = v(x_i) = \text{false}$, ist $l_i = \overline{x_i}$, so ist ebenfalls $\overline{v}(l_i) = \neg v(x_i) = \neg\text{true} = \text{false}$. Es gilt also für alle Literale $l_i \in C_j$, dass $\overline{v}(l_i) = \text{false}$ und somit $\overline{v}(C_j) = \text{false}$. Dies gilt wiederum für alle $j \in \{j_1, \ldots, j_{m-k}\}$, also gibt es höchstens k erfüllte Klauseln.

„⇐“ Gegeben sei eine Variablenbelegung v, so dass höchstens k Formeln erfüllt sind und mindestens $m - k$ nicht erfüllt sind. Wir wählen, falls $v(x_i) = \text{true}$, den Übergang (true_i, x_i), und falls $v(x_i) = \text{false}$ den Übergang $(\text{true}_i, \overline{x}_i)$. Wir wählen weiter als

Fahrplan

$$\pi_i := \begin{cases} 1, & \text{für } i \in \{\text{dep}(\text{true}_i), \text{arr}(\text{true}_i) \mid i = 1, \ldots, n\} \\ & \cup \{\text{dep}(l), \text{arr}(l) \mid l \in \mathfrak{L},\ \overline{v}(l) = \text{true}\} \\ & \cup \{\text{dep}(C_j), \text{arr}(C_j) \mid j = 1, \ldots, m,\ \overline{v}(C_j) = \text{true}\}, \\ 0, & \text{sonst.} \end{cases}$$

Zunächst zeigen wir die Zulässigkeit. Sei dazu $x_a := \pi_j - \pi_i$ für $a = (i, j) \in \mathcal{A}$. Dabei unterscheiden wir folgende Fälle:

- $a \in \mathcal{A}_{drive}$. Dann ist $x_a = 0 \in [0, 0]$, da für jede Fahrt Abfahrt und Ankunft zur gleichen Zeit stattfinden.
- $a \in \mathcal{A}_0$. Dann ist $x_a = \pi_{\text{dep}(\text{true}_i)} - \pi_{\text{arr}(0)} = 1 - 0 = 1 \in [1, 1]$.
- $a \in \mathcal{A}_{Klausel}$. Sei $a = (\text{arr}(l), \text{dep}(C_j))$ für ein $l \in C_j$. Ist $\pi_{\text{arr}(l)} = 0$, so ist die Bedingung $x_a \in [0, 1]$ erfüllt. Anderenfalls ist nach Konstruktion $\overline{v}(l) = \text{true}$. Da $l \in C_j$, ist auch die ganze Klausel C_j erfüllt, also ist nach Konstruktion auch $\pi_{\text{dep}(C_j)} = 1$. Somit ist stets $x_a \in [0, 1]$.
- $a \in \mathcal{A}_{Bewertung}$. Da $\pi_{(B,0,arr)} = 0$, ist $x_a \in [0, 1]$.

Da für alle i ein Übergang ausgewählt wird, werden $2n + m + 1 = k_2$ Fahrzeuge benötigt. Weiter ist

$$\sum_{a=(i,j)\in\mathcal{A}} c_a(\pi_j - \pi_i) = \sum_{j=1}^{m} \underbrace{\pi_{\text{dep}(C_j)}}_{=1 \Leftrightarrow \overline{v}(C_j)=\text{true}} - \underbrace{\pi_{\text{arr}(0)}}_{=0} \leq k = k_1.$$

Wir reduzieren nun die Entscheidungsversion von (TTVS1) auf die von (TTVS2). Dazu setzen wir alle Kosten auf 0 und setzen die maximal erlaubte Fahrzeuganzahl $d := k_2$. Als neue Schranke k_2 für die Kosten wählen wir 0. □

Die letzte Reduktion zeigt, dass (TTVS1) problemlos als Spezialfall von (TTVS2) aufgefasst werden kann. Wir beschränken uns deshalb im nächsten Kapitel auf die Untersuchung von (TTVS2).

3.2 Heuristiken

Da obige Probleme NP-schwer sind, können sie in annehmbarer Zeit vermutlich lediglich heuristisch gelöst werden. In verschiedenen bisher unternommenen Ansätzen zur Integration zweier Planungsschritte wurde ein iteratives Verfahren verwendet, welches abwechselnd die beiden beteiligten Planungsschritte mit unterschiedlichen Eingaben löst (zum Beispiel

Passagier-Routing mit Linienplanung, siehe [JJ13, Sch11]; Passagier-Routing mit Fahrplanoptimierung, siehe [Sie11, Anh12, Sch11]). Um diesen Ansatz auf unser Problem zu übertragen, müssen wir eine Möglichkeit finden, das Ergebnis des Umlaufplanproblems in das Fahrplanoptimierungsproblem einfließen zu lassen.

Dazu genügt es, das aperiodische EAN um *Umlaufaktivitäten*

$$\mathcal{A}^{\mathcal{R}}_{vehicle} := \{(\mathrm{arr}(t_1), \mathrm{dep}(t_2)) \mid (t_1, t_2) \in E(\mathcal{R})\}$$

zu erweitern. Dadurch ergibt sich wieder eine Instanz des aperiodischen Fahrplanoptimierungsproblem, d. h. die Komplexitätsklasse ändert sich durch die Vorgabe von Fahrzeugrouten nicht.

Nach Satz 2.67 und Lemma 2.77 hat im vorliegenden Fall eine Iteration eines Algorithmus des oben beschriebenen Typs polynomielle Laufzeit. Im Unterschied zu den obigen Heuristiken ist die Lösung jeder Iteration auch in der darauf folgenden Iteration noch zulässig. Als Konsequenz daraus ergibt sich Lemma 3.7, das eine Aussage zum Algorithmus 2 beinhaltet.

Algorithmus 2 Abwechselnde Berechnung von optimalem Fahrplan und optimalem Umlaufplan beginnend mit dem Fahrplan

Eingabe: Fahrten $\mathcal{T}$, zugehöriges EAN $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ mit Schranken $L_a \leq U_a \ \forall\, a \in \mathcal{A}$, Menge $\mathcal{C}$ von möglichen Übergängen zwischen Fahrten sowie Schranken $L_{(t_1,t_2)} \leq U_{(t_1,t_2)}$ für die Übergänge

Berechne optimalen Fahrplan π zu $\mathcal{N}$
repeat
  Berechne zu π passenden optimalen Umlaufplan $\mathcal{R}$
  Berechne optimalen Fahrplan π zu $(\mathcal{E}, \mathcal{A} \cup \mathcal{A}^{\mathcal{R}}_{vehicle})$
end repeat

Lemma 3.7. *Arbeitet der Solver zur Berechnung der optimalen Lösungen in Algorithmus 2 deterministisch, so ist es möglich, dass in der ersten Iteration ein Fixpunkt erreicht wird, d. h., dass sich die Lösung nach dem ersten Schritt nicht mehr ändert.*

Beweis. Sei $Z^{(0)}$ der Zielfunktionswert (Gesamtreisezeit) des primär optimal berechneten Fahrplans. Für den Zielfunktionswert $Z^{(1)}$ des im ersten Schleifendurchlauf berechneten Fahrplans gilt:

- $Z^{(1)} \geq Z^{(0)}$, da das primär gelöste Problem ohne Umlaufaktivitäten eine Relaxation des Problems in der Schleife mit fixierten Umläufen ist (siehe Lemma 2.42).
- Der primär berechnete Fahrplan ist nach Berechnung des ersten Umlaufplans weiterhin zulässig, da der Umlaufplan für diesen erstellt wurde.

Also ist der primär berechnete Fahrplan auch nach Berechnung des Umlaufplans eine Optimallösung und kann demnach erneut vom Solver ausgegeben werden. Da der Solver deterministisch arbeitet, wird er in allen folgenden Schritten stets das gleiche Ergebnis liefern. □

Bemerkung 3.8.

1. Es kann sein, dass ein (deterministisch arbeitender) Solver nach dem Hinzufügen der Umlaufaktivitäten eine andere Optimallösung findet, zu der es einen anderen optimalen Umlaufplan gibt, usw. Dabei kann sich die Gesamtreisezeit nicht verändern, die Umlaufkosten können sich jedoch verringern.
2. Das gleiche gilt, wenn zunächst ein optimaler Umlaufplan erstellt wird, zu dem ein passender Fahrplan berechnet wird, anschließend wieder ein Umlaufplan usw. In diesem Fall kann nur die Gesamtreisezeit abnehmen, nicht aber die Umlaufkosten.

Es gibt folgende Möglichkeiten, das Problem, dass der Algorithmus eventuell nur die Startlösung findet, zu umgehen:

1. Es wird mit einer anderen zulässigen Startlösung angefangen.
2. Die Kompatibilitätsrelation bzw. die Bedingungen (3.15) und (3.16) in der IP-Formulierung werden in einem der beiden Planungsschritte relaxiert.

In dieser Arbeit wird nur der erste Ansatz weiterverfolgt. Wird mit einem zulässigen Fahrplan angefangen, so ergibt sich der generische Algorithmus 3.

Algorithmus 3 Generische iterative Heuristik mit gegebenem Fahrplan

Eingabe: Fahrten $\mathcal{T}$, zugehöriges EAN $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ mit Schranken $L_a \leq U_a \; \forall\, a \in \mathcal{A}$, Menge $\mathcal{C}$ von möglichen Übergängen zwischen Fahrten, Schranken $L_{(t_1,t_2)} \leq U_{(t_1,t_2)}$ für die Übergänge, zulässiger Fahrplan π, Abbruchkriterium A

Ausgabe: (möglichst guter) zulässiger Fahrplan π und dazu passender Umlaufplan $\mathcal{R}$

repeat

Berechne tatsächliche Kompatibilitäten

$$\mathcal{C}_\pi := \{(t_1, t_2) \mid \pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} \in [L_{(t_1,t_2)}, U_{(t_1,t_2)}]\}$$

Berechne den optimalen Umlaufplan $\mathcal{R}$ zu $G = (\mathcal{T}, \mathcal{C}_\pi)$ durch Lösen von (VS2).
Löse (TT1) auf $\mathcal{N} \cup \mathcal{A}^{\mathcal{R}}_{vehicle}$ und erhalte den Fahrplan π.

until A.

Es ist alternativ möglich, mit einem Umlaufplan in $G = (\mathcal{T}, \mathcal{C})$ anzufangen, zu welchem es einen passenden Fahrplan gibt. Eine notwendige Bedingung dafür ist, dass der Umlaufplan keine gerichteten Kreise im Trip-Graphen enthält, so dass die Summe der Fahrzeiten aller enthaltenen Fahrten > 0 ist.

Hierzu kann Algorithmus 4 aufgestellt werden.

Algorithmus 4 Generische iterative Heuristik mit gegebenem Umlaufplan

Eingabe: Fahrten $\mathcal{T}$, zugehöriges EAN $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ mit Schranken $L_a \leq U_a \ \forall\, a \in \mathcal{A}$, Menge $\mathcal{C}$ von möglichen Übergängen zwischen Fahrten, Schranken $L_{t_1,t_2} \leq U_{t_1,t_2}$ für die Übergänge, Umlaufplan $\mathcal{R}$, zu dem es einen passenden Fahrplan gibt, Abbruchkriterium A

Ausgabe: (möglichst guter) zulässiger Fahrplan π und dazu passender Umlaufplan $\mathcal{R}$

repeat
Löse (TT1) auf $(\mathcal{E}, \mathcal{A} \cup \mathcal{A}^{\mathcal{R}}_{vehicle})$ und erhalte Fahrplan π.
Berechne tatsächliche Kompatibilitäten

$$\mathcal{C}_\pi := \{(t_1, t_2) \mid \pi_{\mathrm{dep}(t_2)} - \pi_{\mathrm{arr}(t_1)} \in [L_{(t_1,t_2)}, U_{(t_1,t_2)}]\}$$

Berechne den optimalen Umlaufplan $\mathcal{R}$ zu $G = (\mathcal{T}, \mathcal{C}_\pi)$ durch Lösen von (VS2).
until A

3.3 Verallgemeinerung und Untersuchung der Heuristik

Wir untersuchen die Idee der oben beschriebenen Heuristik für beliebige ganzzahlige lineare Programme der Form

(IP1)

$$\min\ (c^T\pi,\, C^T z),$$

so dass

$$A\pi \leq U \tag{3.19}$$
$$Bz \leq V \tag{3.20}$$
$$D\pi \leq W + M\left(\mathbf{1} - \begin{pmatrix} z_I \\ z_I \end{pmatrix}\right) \tag{3.21}$$
$$\pi \in \mathbb{Z}^n \tag{3.22}$$
$$z = \begin{pmatrix} z_I \\ z_{II} \end{pmatrix} \in \{0,1\}^m = \{0,1\}^{m_I + m_{II}}, \tag{3.23}$$

wobei $c \in \mathbb{R}^n$, $C \in \mathbb{R}^m$, $A \in \mathbb{R}^{\mu \times n}$, $U \in \mathbb{R}^\mu$, $B \in \mathbb{R}^{\nu \times m}$, $V \in \mathbb{R}^\nu$, $D \in \mathbb{R}^{2m_I \times n}$, $W \in \mathbb{R}^{2m_I}$ mit $n, m, \mu, \nu, m_I \in \mathbb{N}$ und $M \in \mathbb{R}_{>0}$ so groß ist, dass $D_i\pi - W_i \leq M$ für alle $i \in \{1, \ldots, 2m_I\}$ und alle zulässigen $\pi \in \mathbb{Z}^n$. Der Variablenvektor z ist dabei in zwei Teile z_I, z_{II} aufgeteilt, so dass die Variablen aus z_I in höchstens zwei Nebenbedingungen, welche auch

π enthalten, vorkommen, während die Variablen z_{II} lediglich in den Nebenbedingungen (3.20) und in der Zielfunktion vorkommen.

Es ist also einerseits $c^T\pi$ zu minimieren, so dass $A\pi \leq U$, andererseits ist $C^T z$ zu minimieren, so dass $Bz \leq V$ ist. Der Zusammenhang zwischen den Variablen π und z wird durch die Bedingung (3.21) ausgedrückt. Die Zusammenhangsbedingung lässt sich so interpretieren, dass, immer wenn ein z_i aus z_I auf Eins gesetzt wird, zwei zusätzliche Bedingungen für π erfüllt sein müssen, beziehungsweise immer wenn eine Bedingung $D_i\pi \leq W_i$ nicht erfüllt ist, die zugehörige Variable $z_i = 0$ sein muss.

Notation 3.9. Zu $A \in \mathbb{R}^{m\times n}$ sei $A^+ := (\max\{a_{ij}, 0\})_{\substack{1\leq i\leq m\\ 1\leq j\leq n}}$ der *Positivteil* und $A^- := (-A)^+$ der *Negativteil* von A.

Lemma 3.10. *(TTVS1) und (TTVS2) lassen sich als ganzzahlige lineare Programme der obigen Form formulieren.*

Beweis. Wir zeigen die Behauptung zunächst für (TTVS2): Sei $\tilde{G} = (\mathcal{T} \cup \{0\}, \tilde{\mathcal{C}})$ der erweiterte Trip-Graph. Wir setzen

- $n := |\mathcal{E}|$,
- $m := |\tilde{\mathcal{C}}|$.

Somit können wir die Vektoren c und π mit $i \in \mathcal{E}$ indizieren und die Vektoren C und z mit $e \in \tilde{\mathcal{C}}$. Damit definieren wir für $i \in \mathcal{E}$

$$c_i := \sum_{a=(j,i)\in\mathcal{A}} c_a - \sum_{a=(i,j)\in\mathcal{A}} c_a.$$

Die Werte der C_e für $e \in \tilde{\mathcal{C}}$ bestimmen wir wie in Abschnitt 2.6.2 beschrieben. Wir setzen weiter:

- $A := \begin{pmatrix} -A_{\mathcal{N}}^T \\ A_{\mathcal{N}}^T \end{pmatrix}$, wobei $A_{\mathcal{N}}$ die Knoten-Kanten-Inzidenzmatrix von $\mathcal{N}$ ist,
- $U := \begin{pmatrix} U' \\ -L \end{pmatrix}$, wobei U' und L die gegebenen Schranken zu den Aktivitäten sind,
- $B := \begin{pmatrix} (A_{\tilde{G}})_{\mathcal{T}}^+ \\ (A_{\tilde{G}})_{\mathcal{T}}^- \\ -(A_{\tilde{G}})_{\mathcal{T}}^+ \\ -(A_{\tilde{G}})_{\mathcal{T}}^- \\ (A_{\tilde{G}})_0^+ \end{pmatrix}$, wobei $(A_{\tilde{G}})_{\mathcal{T}}$ alle zu Knoten $t \in \mathcal{T}$ gehörenden Zeilen der Knoten-Kanten-Inzidenzmatrix von $\tilde{G}$ enthält und $(A_{\tilde{G}})_0$ die zum Depot-Knoten gehörende Zeile 0 darstellt,

- $V := \begin{pmatrix} 1 \\ \vdots \\ 1 \\ d \end{pmatrix}$,
- $D := \begin{pmatrix} \Delta \\ -\Delta \end{pmatrix}$, wobei $\Delta = (\delta_{e,i})_{\substack{e \in \mathcal{C} \\ i \in \mathcal{E}}}$ mit $\delta_{e,i} = \begin{cases} 1, & \text{falls } i = \text{dep}(t_2), \\ -1, & \text{falls } i = \text{arr}(t_1), \\ 0, & \text{sonst,} \end{cases}$
- $W := \begin{pmatrix} U \\ -L \end{pmatrix}$, wobei U und L die gegebenen Schranken zu den Übergängen sind.

z_I enthält die zu den Übergängen $(t_1, t_2) \in \mathcal{C}$ gehörenden Variablen.

Obwohl wir oben gesehen haben, dass man (TTVS1) als Spezialfall von (TTVS2) auffassen kann, betrachten wir aufgrund ihrer Einfachheit die direkte Darstellung von (TTVS1) als (IP1): In diesem Fall kann $B := \begin{pmatrix} A_G^+ \\ A_G^- \end{pmatrix}$, $V := \mathbf{1}$ und $C := -\mathbf{1}$ gesetzt werden, wobei A_G die Knoten-Kanten-Inzidenzmatrix des Trip-Graphen G ist und es gilt $z_I = z$. □

Es werden nun die Algorithmen 3 und 4 auf die oben beschriebene IP-Formulierung verallgemeinert. Dazu betrachten wir die beiden Teilprobleme, die entstehen, wenn ein Variablentyp vorgegeben ist.

Lemma 3.11.

1. *Sei $\pi \in \mathbb{Z}^n$ mit $A\pi \leq U$ und seien z_π und C_π die Teilvektoren von z bzw. C, die durch Streichen der Einträge $C_i, z_i,\ i = 1, \ldots, m_I$ entstehen, an denen $D_i\pi > W_i$ oder $D_{m_I+i}\pi > W_{m_I+i}$ ist, sowie B_π die Teilmatrix von B, die durch Streichen der entsprechenden Spalten entsteht. Ferner sei z^* eine Optimallösung des folgenden ganzzahligen Programms.*

   (IP1$_\pi$)

   $$\min\ C_\pi^T z_\pi,$$

   so dass

   $$\begin{aligned} B_\pi z_\pi &\leq V \\ z_\pi &\in \{0,1\}^{m'}. \end{aligned}$$

   Dann ist

   $$\tilde{z}^* = \text{argmin}_z\{C^T z \mid (\pi, z)\ \textit{ist zulässig für}\ (\text{IP1})\},$$

   wobei

   $$\tilde{z}_i^* := \begin{cases} 0, & \textit{falls } D_i\pi > W_i \textit{ oder } D_{m_I+i}\pi > W_{m_I+i}, \\ z_i^*, & \textit{sonst.} \end{cases}$$

2. *Sei $z \in \{0,1\}^m$ mit $Bz \leq W$. Für eine Optimallösung π^* des folgenden ganzzahligen Programms*

   (IP1$_z$)

   $$\min\ c^T\pi,$$

   so dass

   $$\begin{aligned} A\pi &\leq U \\ D_z\pi &\leq W \\ \pi &\in \mathbb{Z}^n, \end{aligned}$$

   gilt:

   $$\pi^* = \operatorname{argmin}_\pi\{c^T\pi \mid (\pi, z) \text{ ist zulässig für (IP1)}\},$$

   wobei D_z die Teilmatrix von D ist, die genau die Zeilen D_i, D_{m_I+i} mit $z_i = 1$ enthält.

Beweis.

1. Es wird gezeigt, dass für ein festes π gilt:

   $$z_\pi \text{ ist zulässig für (IP1}_\pi) \Leftrightarrow (\pi, \tilde{z}) \text{ ist zulässig für (IP1)},$$

   wobei $\tilde{z}$ wie oben definiert wird. Daraus folgt die Behauptung, da $C_\pi^T z_\pi = C^T\tilde{z}$.

   „⇐“ Sei $(\pi, \tilde{z})$ zulässig für (IP1) und sei $B_{\overline{\pi}}$ die Matrix, welche die Spalten von B_i mit $D_i\pi > W_i$ oder $D_{m_I+i}\pi > W_{m_I+i}$ enthält. Dann ist

   $$B_\pi z_\pi = \begin{pmatrix} B_\pi & B_{\overline{\pi}} \end{pmatrix} \begin{pmatrix} z_\pi \\ 0 \end{pmatrix} = B\tilde{z} \leq V.$$

   „⇒“ Sei z_π zulässig für (IP1$_\pi$). Die Bedingung (3.19) ist nach Voraussetzung erfüllt. Überdies gilt:

   $$B\tilde{z} = \begin{pmatrix} B_\pi & B_{\overline{\pi}} \end{pmatrix} \begin{pmatrix} z_\pi \\ 0 \end{pmatrix} = B_\pi z_\pi \leq V,$$

   also ist (3.20) ebenfalls erfüllt. Für Indizes $i \in \{1, \ldots, m_I\}$ mit $D_i\pi \leq W_i$ und $D_{m_I+i}\pi \leq W_{m_I+i}$ ist Bedingung (3.21) erfüllt. Für alle anderen Indizes i ist $\tilde{z}_i = 0$. Also ist $D_i\pi - W_i \leq M = M(1 - \tilde{z}_i)$ und $D_{m_I+i}\pi - W_{m_I+i} \leq M(1 - \tilde{z}_i)$, d. h. Bedingung (3.21) ist auch in diesem Fall erfüllt.

2. Analog zu oben wird gezeigt, dass für ein festes z gilt:

   $$\pi \text{ ist zulässig für (IP1}_z) \Leftrightarrow (\pi, z) \text{ ist zulässig für (IP1)}.$$

   Daraus folgt die Behauptung.

„⇐“ Sei (π, z) zulässig für (IP1). Dann ist $A\pi \leq U$. Für Indizes i mit $z_i = 1$ ist zudem $D_i\pi \leq W_i + M(1 - z_i) = W_i$ und genauso $D_{m_I+i}\pi \leq W_{m_I+i}$.

„⇒“ Sei π zulässig für (IP1_z), dann erfüllt π Bedingung (3.19). Bedingung (3.20) ist nach Voraussetzung erfüllt. Zudem gilt für Indizes $i \in \{1, \ldots, m_I\}$ mit $z_i = 1$, dass $D_i\pi \leq W_i$ und $D_{m_I+i}\pi \leq W_{m_I+i}$. Somit ist Bedingung (3.21) für diese erfüllt. Für Indizes mit $z_i = 0$ gilt $D_i\pi - W_i \leq M = M(1 - z_i)$, weshalb die Bedingung gleichermaßen erfüllt ist.

□

Mit diesen Teilproblemen lassen sich wieder zwei Varianten der Heuristik – Algorithmen 5 und 6 – formulieren, je nachdem mit welchem Teilproblem angefangen wird.

Algorithmus 5 Iterative Heuristik für (IP1) mit vorgegebenen π-Werten

Eingabe: $c \in \mathbb{R}^n,\ C \in \mathbb{R}^m,\ A \in \mathbb{R}^{\mu\times n},\ U \in \mathbb{R}^\mu,\ B \in \mathbb{R}^{\nu\times m},\ V \in \mathbb{R}^\nu,\ D \in \mathbb{R}^{2m_I\times n},\ W \in \mathbb{R}^{2m_I}$, wobei $n, m, \mu, \nu, m_I \in \mathbb{N}$ sowie ein Startvektor $\pi^{(0)} \in \mathbb{Z}^n$, für den es ein $z \in \{0,1\}^m$ gibt, für das $(\pi^{(0)}, z)$ zulässig für (IP1) ist.
Ausgabe: (möglichst gute) zulässige Lösung $(\pi, z) \in \mathbb{Z}^n \times \{0,1\}^m$.

Setze $i := 1$.
repeat
  Löse $(\text{IP1}_{\pi^{(i-1)}})$ und fixiere erhaltene Lösung $z^{(i)}$.
  Löse $(\text{IP1}_{z^{(i)}})$ und fixiere erhaltene Lösung $\pi^{(i)}$.
  $i := i + 1$.
until Abbruchkriterium

Algorithmus 6 Iterative Heuristik für (IP1) mit vorgegebenen z-Werten

Eingabe: $c \in \mathbb{R}^n,\ C \in \mathbb{R}^m,\ A \in \mathbb{R}^{\mu\times n},\ U \in \mathbb{R}^\mu,\ B \in \mathbb{R}^{\nu\times m},\ V \in \mathbb{R}^\nu,\ D \in \mathbb{R}^{2m_I\times n},\ W \in \mathbb{R}^{2m_I}$, wobei $n, m, \mu, \nu, m_I \in \mathbb{N}$ sowie ein Startvektor $z^{(0)} \in \{0,1\}^m$, für den es ein $\pi \in \mathbb{Z}^n$ gibt, für das $(\pi, z^{(0)})$ zulässig für (IP1) ist.
Ausgabe: (möglichst gute) zulässige Lösung $(\pi, z) \in \mathbb{Z}^n \times \{0,1\}^m$.

Setze $i := 1$.
repeat
  Löse $(\text{IP1}_{z^{(i-1)}})$ und fixiere erhaltene Lösung $\pi^{(i)}$.
  Löse $(\text{IP1}_{\pi^{(i)}})$ und fixiere erhaltene Lösung $z^{(i)}$.
  $i := i + 1$.
until Abbruchkriterium

Bemerkung 3.12. Sind die Matrizen B und $\begin{pmatrix} A \\ D \end{pmatrix}$ total unimodular und die Vektoren U, V und W ganzzahlig, so haben Algorithmen 5 und 6 polynomielle Laufzeit. In der Anwendung auf (TTVS2) ist dies der Fall, denn:

- In der Matrix
$$\begin{pmatrix} (A_{\tilde{G}})^+_{\mathcal{T}} \\ (A_{\tilde{G}})^-_{\mathcal{T}} \\ (A_{\tilde{G}})^+_0 \end{pmatrix}$$
(siehe Beweis zu Lemma 3.10) enthält jede Spalte zu $z_{(t_1,t_2)}$ mit $t_1, t_2 \in \mathcal{T}$ und jede Spalte zu $z_{(0,t)}$ mit $t \in \mathcal{T}$ genau zweimal den Eintrag 1 und jede Spalte zu $z_{(t,0)}$ mit $t \in \mathcal{T}$ genau eine Eins. Partitioniert man die Zeilen in die zu $(A_{\tilde{G}})^+_{\mathcal{T}}$ gehörenden Zeilen zusammen mit $(A_{\tilde{G}})^+_0$ und in die zu $(A_{\tilde{G}})^-_{\mathcal{T}}$ gehörenden Zeilen, so ist die Bedingung aus Satz 2.48 erfüllt. Aus Lemma 2.46 folgt nun die totale Unimodularität von B.
- Die Zeilen aus D können als zu den oben eingeführten Umlaufaktivitäten gehörend aufgefasst werden. Damit kann
$$\begin{pmatrix} A \\ D \end{pmatrix}$$
als Koeffizientenmatrix einer Instanz von (TT1) angesehen werden. Diese ist nach dem Beweis von Satz 2.67 total unimodular.

Lemma 3.13.

1. *In Algorithmus 5 gilt für alle Iterationen $i = 1, 2, \ldots$:*
$$(c^T\pi^{(i)}, C^T z^{(i+1)}) \leq (c^T\pi^{(i-1)}, C^T z^{(i)}).$$
2. *In Algorithmus 6 gilt für alle Iterationen $i = 1, 2, \ldots$:*
$$(c^T\pi^{(i+1)}, C^T z^{(i)}) \leq (c^T\pi^{(i)}, C^T z^{(i-1)}).$$

Beweis. Wir beweisen die erste Behauptung; der Beweis lässt sich die zweite übertragen. Da in jeder Iteration i der Wert $z^{(i)}$ Lösung von $(\mathrm{IP1}_{\pi^{(i-1)}})$ ist, gilt nach Lemma 3.11, Teil 1, dass $(\pi^{(i-1)}, z^{(i)})$ eine zulässige Lösung von (IP1) ist. Nach Teil 2 des gleichen Lemmas, ist $\pi^{(i)} = \operatorname{argmin}_\pi\{c^T\pi \mid (\pi, z^{(i)}) \text{ zulässig}\}$. Insbesondere gilt $c^T\pi^{(i)} \leq c^T\pi^{(i-1)}$. Da nun $(\pi^{(i)}, z^{(i)})$ zulässig ist und wieder nach Teil 1 des Lemmas $z^{(i+1)} = \operatorname{argmin}_z\{C^T z \mid (\pi^{(i)}, z) \text{ zulässig}\}$, ist $C^T z^{(i+1)} \leq C^T z^{(i)}$. □

Beispiele zeigen für Algorithmus 5 oder 6, dass $(c^T\pi^{(2)}, C^T z^{(2)}) \ll (c^T\pi^{(1)}, C^T z^{(1)})$ gelten kann, d. h., dass der Algorithmus sich nach der ersten Iteration in beiden Zielfunktionswerten verbessert. Da solche Beispiele allerdings relativ groß sind und die Ergebnisse im praktischen Teil die Aussage belegen, verzichten wir auf die Angabe solcher Beispiele an dieser Stelle.

Lemma 3.14. *Seien folgende Bedingungen erfüllt:*

1. *Die Abbildungen $\pi \mapsto c^T\pi$ und $z \mapsto C^T z$ sind auf dem zulässigen Bereich injektiv.*

2. *Sind $(\pi^*, z^*), (\pi, z)$ zulässige Lösungen von* (IP1) *mit $(c^T\pi^*, C^T z^*) \ll (c^T\pi, C^T z)$, dann gibt es eine zulässige Lösung (π', z'), so dass entweder*

$$c^T\pi' = c^T\pi^* \wedge C^T z' = C^T z$$

oder

$$c^T\pi' = c^T\pi \wedge C^T z' = C^T z^*.$$

Das Abbruchkriterium sei, dass eine bereits erhaltene Lösung erneut als Ergebnis berechnet wird. Dann terminieren die Algorithmen 5 und 6 unabhängig vom Startvektor bei einer effizienten Lösung.

Beweis. Wir betrachten Algorithmus 5. Ist $(\pi^{(i-1)}, z^{(i)})$ eine effiziente Lösung, gilt nach Lemma 3.13 $(c^T\pi^{(i)}, C^T z^{(i+1)}) \leq (c^T\pi^{(i-1)}, C^T z^{(i)})$, andererseits gilt nach Definition von Effizienz $(c^T\pi^{(i)}, C^T z^{(i+1)}) \not\leq (c^T\pi^{(i-1)}, C^T z^{(i)})$. Also gilt Gleichheit. Da beide Zielfunktionen auf dem zulässigen Bereich injektiv sind, folgt hieraus sogar $(\pi^{(i)}, z^{(i+1)}) = (\pi^{(i-1)}, z^{(i)})$, was zum Abbruch des Algorithmus führt.

Es bleibt zu zeigen: Falls $(\pi^{(i-1)}, z^{(i)})$ nicht effizient ist, terminiert der Algorithmus nicht. Nach Lemma 3.11 gilt

$$z^{(i)} = \text{argmin}_z\{C^T z \mid (\pi^{(i-1)}, z) \text{ zulässig für (IP1)}\},$$

so dass es kein z' gibt mit $(\pi^{(i-1)}, z')$ zulässig und $C^T z' < C^T z^{(i)}$. Da $\pi \mapsto c^T\pi$ injektiv ist, gibt es auch keine zulässige Lösung (π', z') von (IP1) mit $c^T\pi' = c^T\pi^{(i-1)}$ und $C^T z' < C^T z^{(i)}$.

Da $(\pi^{(i-1)}, z^{(i)})$ nicht effizient ist, gibt es allerdings eine zulässige Lösung (π^*, z^*), so dass $(c^T\pi^*, C^T z^*) < (c^T\pi^{(i-1)}, C^T z^{(i)})$. Aufgrund des zuvor Gezeigten muss $c^T\pi^* < c^T\pi^{(i-1)}$ sein. Nach Voraussetzung existiert dann eine zulässige Lösung (π'', z''), so dass $c^T\pi'' = c^T\pi^* < c^T\pi^{(i-1)}$ und $C^T z'' = C^T z^{(i)}$, denn der andere Fall ist laut oben unmöglich. Die Abbildung $z \mapsto C^T z$ ist ebenfalls injektiv, weshalb $z'' = z^{(i)}$ sein muss. Also gilt:

$$\text{min}_\pi\{c^T\pi \mid (\pi, z^{(i)}) \text{ zulässig für (IP1)}\} \leq c^T\pi'' < c^T\pi^{(i-1)},$$

d. h. der Algorithmus findet ein $\pi^{(i)} \neq \pi^{(i-1)}$ mit kleinerem Zielfunktionswert, weshalb er noch nicht terminiert. Für Algorithmus 6 lässt sich der Beweis analog führen. □

Bemerkung 3.15. Ist eine der beiden obigen Bedingungen nicht erfüllt, so ist es möglich, dass Algorithmus 5 oder 6 bei einer zulässigen Lösung (π^s, z^s) terminiert, die nicht einmal schwach effizient ist, selbst dann, wenn $z = 0$ zulässig ist und (π^s, z^s) nicht von den lexikographisch optimalen Lösungen (bei unterschiedlicher Sortierung der Zielfunktionen) dominiert wird.

Beispiel. Im folgenden ganzzahligen linearen Programm ist die erste Voraussetzung aus dem Lemma erfüllt, die zweite Voraussetzung jedoch nicht:

$$\begin{aligned} &\min\ (\pi_1 + 2\pi_2,\ -2z_1 - z_2) \\ \text{so dass}\ \ &-\pi \leq \begin{pmatrix} -1 \\ -1 \end{pmatrix} + M(1 - z) \\ &\pi_i \in \{0, 1\} && \forall i \in \{1, 2\} \\ &z_i \in \{0, 1\} && \forall i \in \{1, 2\} \end{aligned}$$

Wird Algorithmus 5 mit dem Startvektor $\pi^{(0)} = (1, 0)^T$ gestartet, wird im ersten Schritt $z^{(1)} = (1, 0)^T$ gefunden. Da nun $\pi_1 \geq 1$ sein muss, wird wieder die Lösung $\pi^{(1)} = (1, 0)^T$ gefunden. Da keine Änderung stattfindet, terminiert der Algorithmus mit $(\pi^{(1)}, z^{(1)})$, wobei $(c^T\pi^{(1)}, C^T z^{(1)}) = (2, -1)$. Das Gleiche geschieht, wenn Algorithmus 6 mit Startvektor $z^{(0)} = (1, 0)^T$ gestartet wird. Allerdings ist (π^*, z^*) mit $\pi^* = (0, 1)^T,\ z^* = (0, 1)^T$ ebenfalls zulässig und in beiden Zielfunktionswerten besser: $(c^T\pi^*, C^T z) = (1, -2)$. Der Bildraum der Zielfunktion ist in Abb. 3.7 dargestellt. Dabei ist die vom Algorithmus gefundene Lösung rot und die dominierende Lösung grün dargestellt.

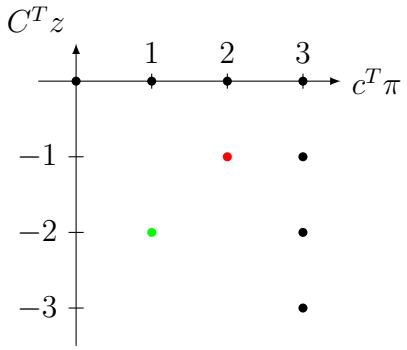

Abb. 3.7: Bild des zulässigen Bereichs für das bikriterielle ganzzahlige Programm aus Beispiel 3.3

Beispiel. Im unten stehenden ganzzahligen linearen Programm ist nur die zweite Voraussetzung erfüllt.

$$\begin{aligned} & \min\ (2\pi_1 - \pi_2 + 2\pi_3,\ -z_1 - 2z_2) \\ \text{so dass}\ & \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \end{pmatrix} \pi \leq \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\ & \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix} \pi \leq \begin{pmatrix} -1 \\ -1 \end{pmatrix} + M(1-z) \\ & \pi \in \{0,1\}^3 \\ & z \in \{0,1\}^2 \end{aligned}$$

Wird Algorithmus 5 mit dem Startvektor $\pi^{(0)} = (1,0,0)^T$ gestartet, findet der Algorithmus die Lösung $z^{(1)} = (1,0)^T$. Somit muss im nächsten Schritt $\pi_1 = 1$ gelten. Die beste zulässige Lösung, die das erfüllt, ist wieder $\pi^{(1)} = (1,0,0)^T$, also terminiert der Algorithmus. Die gefundene Lösung hat die Zielfunktionswerte $(c^T\pi^{(1)}, C^T z^{(1)}) = (2,-1)$. Allerdings ist die Lösung (π^*, z^*) mit $\pi^* = (0,1,1)^T$, $z^* = (0,1)^T$ in beiden Zielfunktionswerten besser: $(c^T\pi^*, C^T z^*) = (1,-2)$. Abb. 3.8 zeigt den zugehörigen Bildraum; es ist ersichtlich, dass die zweite Voraussetzung erfüllt ist. Der blaue Punkt gehört zu der Lösung $\pi = (0,0,1)^T$, $z = (0,1)^T$, welche vom Algorithmus nicht gefunden wird.

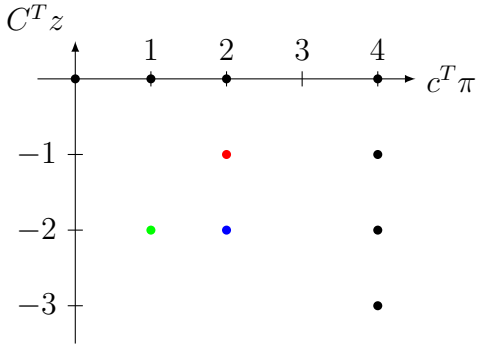

Abb. 3.8: Bild des zulässigen Bereichs für das bikriterielle ganzzahlige Programm aus Beispiel 3.3

3.4 Implementierung

In diesem Abschnitt wird die Implementierung der oben beschriebenen Heuristik vorgestellt. Diese ist in das Projekt LinTim der Universität Göttingen, welches in Abschnitt 3.4.2 vorgestellt wird, eingebettet. Dadurch ist es möglich, die Heuristik mit anderen Planungsschritten der Verkehrsplanung zu verbinden und die in LinTim vorhandenen Datensätze zu kombinieren. Außerdem ist ein Vergleich mit den bereits vorhandenen Routinen

möglich. Zunächst wird die Implementierung der allgemeinen Heuristik aus Algorithmus 5 vorgestellt, welche unabhängig von LinTim funktioniert. Anschließend wird auf deren Anwendung zur Bestimmung von Fahrplan und Umlaufplan eingegangen. Für die Implementierung wurden folgende Technologien verwendet:

- Die Programmiersprache *Java*,
- der *Xpress-Optimizer* von der *Fair Isaac Corporation* sowie die zugehörige Schnittstelle *Xpress-Mosel* (64-bit v3.4.0),
- das Build-Management-Tool *make*,
- das freie Kommandozeilenprogramm `md5sum`.

3.4.1 Algorithmus 5

Die hier vorgestellte Implementierung des Algorithmus 5 zur heuristischen Lösung von (IP1) kann nicht nur dazu verwendet werden, wie unten beschrieben, die Probleme (TTVS1) und (TTVS2) heuristisch zu lösen, sondern kann dank ihrer Allgemeinheit auch auf jedes andere Optimierungsproblem angewendet werden, das sich in der Form von (IP1) schreiben lässt. Die Eingabe wird dem Programm in Form von Dateien zugeführt. Da die Dateien von *Mosel* verwendet werden, müssen sie in dem folgenden, in Backus-Naur-Form beschriebenen Format vorliegen:

```
<Datei>       ::= (<Init_Zahl>|<Init_Vektor>)
                  [<Datei>]

<Init_Zahl>   ::= <Name>":" <Wert>

<Init_Vektor> ::= <Name>":" "["<Werte>"]"

<Werte>       ::= <Wert>["," <Werte>]
```

Matrizen werden dabei wie Vektoren angegeben, wobei ihre Einträge zeilenweise durchiteriert werden.

Die Eingabeparameter müssen in den in Tabelle 3.1 angegebenen Dateien unter den dort angegebenen Namen abgelegt sein.

Parametername in Algorithmus 5	Dateiname	Name in der Datei
n	sizes.dat	`dimPi`
m	sizes.dat	`dimZ`
μ	sizes.dat	`rowsA`
ν	sizes.dat	`rowsB`
m_I	sizes.dat	`halfRowsD`
c	smallC.dat	`c`
C	bigC.dat	`C`
A	A.dat	`A`
U	U.dat	`U`
B	B.dat	`B`
V	V.dat	`V`
D	D.dat	`D`
W	W.dat	`W`
$\pi^{(0)}$	pi.dat	`pi`

Tab. 3.1: Namen und Dateien der Eingabeparameter von Algorithmus 5

Die Hauptroutine `Algo5.run()` wurde in *Java* folgendermaßen programmiert:

```
int iterations = 0;
do {
   System.out.println("Iteration␣"+(++iterations));
   Mosel.exec(SOURCEPATH+"IP1pi.mos");
   Mosel.exec(SOURCEPATH+"IP1z.mos");
} while (solutionChanged() && MAX_ITERATIONS > iterations);
```

Darin werden durch die beiden Methodenaufrufe `Mosel.exec()` *Mosel*-Programme gestartet, welche die beiden in Lemma 3.11 formulierten ganzzahligen Programme (IP1_π) und (IP1_z) lösen und im Folgenden genauer vorgestellt werden.

Das Programm IP1pi.mos liest zuerst die Eingabe aus den in Tabelle 3.1 angegebenen Dateien (1. Schritt) und berechnet anschließend die Menge von Indizes i, für die weder $D_i\pi > W_i$ noch $D_{m_I+i}\pi > W_{m_I+i}$ ist, um entsprechend viele Entscheidungsvariablen zu deklarieren (2. Schritt). Mit diesen löst es das IP (3. Schritt), generiert daraus den in Lemma 3.11 definierten Vektor $\tilde{z}^*$ (4. Schritt) und schreibt diesen in die Datei z.dat (5. Schritt). Die entscheidenden Schritte 2 bis 4 werden durch den folgenden Quellcode implementiert.

```
chosenIndices := union(i in 1..dimZ) {i}

forall(i in 1..halfRowsD)
  if(sum(j in 1..dimPi) D(i,j) * pi(j) > W(i) or
  sum(j in 1..dimPi) D(i+halfRowsD,j) * pi(j) > W(i+halfRowsD))
     then
    chosenIndices -= {i}
```

```
  end-if

declarations
  z: array(chosenIndices) of mpvar
  zSol: array(1..dimZ) of real
end-declarations

forall(j in chosenIndices)
  z(j) is_binary

forall(i in 1..rowsB)
  sum(j in chosenIndices) B(i,j)*z(j) <= V(i)

minimize(sum(j in chosenIndices) C(j) * z(j))

forall(j in 1..dimZ) do
  if(j in chosenIndices) then
    zSol(j) := getsol(z(j))
  else
    zSol(j) := 0
  end-if
end-do
```

In Programm IP1z.mos wird ebenfalls erst die Eingabe aus den in Tabelle 3.1 angegebenen Dateien und der Datei z.dat gelesen (1. Schritt). Anschließend werden die Nebenbedingungen des IPs aufgestellt (2. Schritt). Dabei werden die Nebenbedingungen $D_i\pi \leq W_i$ und $D_{m_i+i}\pi \leq W_{m_I+i}$ nur dann aufgenommen, wenn $z_i = 1$ ist. Nachdem das IP gelöst wird (3. Schritt), wird das Ergebnis in die Datei pi.dat geschrieben (4. Schritt). Insbesondere ändert sich der Inhalt der Datei pi.dat durch Ausführung des obigen Programms. Der folgende Quellcode ist für die beiden Schritte 2 und 3 verantwortlich.

```
forall(j in 1..dimPi)
  pi(j) is_integer

forall(i in 1..rowsA)
  sum(j in 1..dimPi) A(i,j)*pi(j) <= U(i)

forall(i in 1..halfRowsD)
  if(z(i) = 1) then
    sum(j in 1..dimPi) D(i,j) * pi(j) <= W(i)
    sum(j in 1..dimPi) D(i+halfRowsD,j) * pi(j) <= W(i+halfRowsD)
  end-if
```

Die Methode `solutionChanged()` des obigen Java-Programms verwendet das Kommandozeilenprogramm `md5sum`, um die MD5-Prüfsumme der Dateien pi.dat und z.dat zu be-

rechnen und vergleicht diese mit allen vorher berechneten Prüfsummen.

3.4.2 LinTim und Ein- und Ausgabeformat

LinTim ist eine an der Universität Göttingen entwickelte Softwaresammlung zum Lösen mathematischer Probleme in der Verkehrsplanung. Wie in [SS09, GSS12] beschrieben, ist das Ziel von LinTim, die Interaktion unterschiedlicher Planungsschritte experimentell zu untersuchen. Dies wird durch einheitliche, einfach gehaltene Dateiformate ermöglicht.

Bis dato sind Algorithmen zur Linienplanung, zur periodischen und aperiodischen Fahrplanoptimierung, zum Verspätungsmanagement und zur Umlaufplanung in LinTim enthalten. Zwischen den Planungsschritten wandeln Konvertierungsalgorithmen die Ausgabe eines vorausgehenden Planungsschrittes in die Eingabe des folgenden um. In LinTim ist ein Algorithmus vorhanden, der aus einem Linienkonzept ein EAN erzeugt sowie ein Algorithmus, der einen periodischen Fahrplan in einen aperiodischen ausrollt. Die Algorithmen der einzelnen Planungsschritte werden mithilfe von *make*-Targets gestartet.

Die Algorithmen zur Umlaufplanung sind im Rahmen der Diplomarbeit [Uff10] entstanden und dort beschrieben. Für den Vergleich mit den hier entwickelten Algorithmen eignen sich das Minimal Decomposition Model 2 (Minimierung der Fahrzeuganzahl) und das Netzwerkfluss-Modell (Minimierung der Umlaufkosten).

Alle Ein- und Ausgabedateien von LinTim liegen in einem CSV-Format mit dem Trennzeichen „;“ vor. Sie enthalten einen Header, der die Bedeutung der Spalten angibt. Neben den oben genannten existieren Konfigurationsdateien, welche Parameter enthalten, mit denen das Verhalten von LinTim gesteuert werden kann. In ihnen werden z. B. die Speicherorte der Ein- und Ausgabedateien angegeben. Der einzige fest kodierte Dateiname ist basis/Config.cnf. Diese Datei neben einigen Parametern Verweise auf weitere Konfigurationsdateien.

Für (TTVS1) besteht die Eingabe, wie in Abschnitt 3.1 beschrieben, aus einem aperiodischen EAN $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ mit Schranken $L_a \leq U_a$ und Gewichten c_a für die enthaltenen Aktivitäten a, einer Menge $\mathcal{T}$ von Fahrten, einer Menge $\mathcal{C}$ von Paaren kompatibler Fahrten (*Kompatibilitäten*) zusammen mit unteren und oberen Schranken $L_e \leq U_e,\ e \in \mathcal{C}$. Für (TTVS2) kommen noch Kosten für die Übergänge zwischen zwei Fahrten c_{t_1,t_2} sowie zwischen Depot und einer Fahrt $c_{0,t},\ c_{t,0}$, Fahrzeugkosten α und eine maximale Fahrzeuganzahl d hinzu. Wir geben im Folgenden an, wie diese in LinTim vorliegen müssen. Wie in den vorhandenen Routinen in LinTim stimmt das Format der unten beschriebenen Eingabedateien mit dem der Ausgabedateien vorheriger Planungsschritte überein.

1. Der Parameter `default_events_expanded_file` gibt den Speicherort der Datei an, welche die Informationen über die Ereignisse enthält. Sie besteht aus den folgenden Spalten:

```
# event-id; periodic-id; type; time; passengers
```

Die vorgestellten Algorithmen benötigen lediglich die `event-id` zur Identifikation der Ereignisse $i \in \mathcal{E}$.

2. Die dem Parameter `default_activities_expanded_file` zugeordnete Datei enthält Informationen zu den Aktivitäten. Sie wird gegenwärtig genauso wie die Datei für die Ereignisse durch die Targets `ro-rollout` oder `tim-timetable` erzeugt und ist folgendermaßen aufgebaut.

   ```
   # activity-id; periodic-id; type; tail-event-id; head-event-id;
   # lower-bound; passengers
   ```

   Für die Algorithmen dieser Arbeit ist die zweite Spalte nicht von Bedeutung. Jede Zeile der Datei repräsentiert eine Aktivität $a \in \mathcal{A}$, beispielsweise steht die Zeile

   ```
   2; 1; "drive"; 2; 7; 1020; 9199.0
   ```

   für die Aktivität $a_2 = (2, 7) \in \mathcal{A}_{drive}$ mit $L_a = 1020,\ c_a = 9199$. Da dabei keine oberen Schranken für die Aktivitäten vorgesehen sind, wurde dafür immer 10000 gewählt. Wird diese Datei mit einem der LinTim-Targets erzeugt, so enthält diese neben den Fahr-, Warte- und Umsteigeaktivitäten auch Paare potenzieller Headway-Aktivitäten, welche in unserem Verfahren keine Berücksichtigung finden.

3. Die Datei, deren Speicherort der Parameter `default_trips_file` angibt, beinhaltet die Informationen für die Fahrten. Der Header sieht folgendermaßen aus:

   ```
   # start-id; periodic-start-id; start-station; start-time;
   # end-id; periodic-end-id; end-station; end-time; line
   ```

   Jede Zeile steht für eine Fahrt $t \in \mathcal{T}$. Für die implementierten Algorithmen haben die erste, dritte, fünfte und siebte Spalte Relevanz. Die erste Spalte enthält die Identifikationsnummer von $\text{dep}(t)$ (siehe Notation 2.64), die dritte enthält die Nummer der Station $\text{start}(t)$ (siehe Notation 2.62). Entsprechend enthält die fünfte Spalte die Nummer von $\text{arr}(t)$ und die siebte die von $\text{end}(t)$. Die Datei wird bisher durch das Target `ro-trips` erzeugt. Da das mit `ro-rollout` erzeugte EAN auch Ereignisse enthalten kann, welche keine ein- oder ausgehende Fahr- oder Warteaktivität haben, entstehen dabei auch Fahrten der Länge 0. Um dies zu unterbinden, wurde das Target `tim-vs-trips` erstellt, welches diese Fahrten aussortiert.

4. Für die Kompatibilitäten gibt es in LinTim bisher noch kein vordefiniertes Format. Deshalb wurde eine Routine geschrieben, die aus der Menge aller Fahrten und den Abständen zwischen zwei Stationen auf einem kürzesten Weg im PTN eine Menge von Kompatibilitäten sowie Schranken und Kosten dieser erzeugt. Je nach Wert des Parameters `tim_vs_all_compatibilities` werden dabei Kompatibilitäten zwischen jedem Paar von verschiedenen Fahrten erzeugt, oder nur solche Kompatibilitäten (t_1, t_2) mit $\text{end}(t_1) = \text{start}(t_2)$. Als Mindestdauer der Kompatibilität wird die

Summe der Distanz von end(t_1) nach start(t_2) im PTN (d. h. der Länge eines kürzesten Weges) und des Parameterwertes von `vs_min_distance` verwendet, für die Höchstdauer wird die Summe der Distanz von end(t_1) nach start(t_2) und des Parameterwertes von `vs_max_distance` verwendet. Die Kosten werden als Produkt der Distanz und des Wertes von `vs_cost_factor` berechnet.

Das Target `compatibilities` berechnet Kompatibilitäten wie oben beschrieben und gibt diese in die vom Parameter `default_compatibilities_file` bestimmte Datei aus, welche folgendes Format aufweist:

```
# compatibility-id; tail-trip-id; head-trip-id; lower-bound;
# upper-bound; cost
```

Für die unten beschriebene Heuristik kann mithilfe des Parameters `tim_vs_given_compatibilities` bestimmt werden, ob die Kompatibilitäten aus der oben beschriebenen Datei gelesen oder direkt berechnet werden sollen.

5. Für die oben beschriebene Berechnung der Kompatibilitäten sind die Abstände zwischen den Stationen entscheidend. Die Abstände zwischen jedem Paar von Stationen sind in der durch den Parameter `default_vs_station_distances_file` bestimmten Datei gespeichert, welche das Format

   ```
   # from-station-id; to-station-id; distance
   ```

   aufweist. Die Datei wurde bislang bei Ausführung des Targets `vehicle-schedules` miterzeugt. Um ausschließlich diese Datei zu erhalten, wurde das Target `distances` zu dem *Makefile* hinzugefügt.

6. Die Abstände von Paaren von Stationen im PTN werden mithilfe der durch die Parameter `default_stops_file` und `default_edges_file` bestimmten Dateien berechnet. Die erste hat die Form

   ```
   # stop-id; short-name; long-name; x-coordinate; y-coordinate
   ```

   und die zweite hat die Form

   ```
   # edge-id; left-stop-id; right-stop-id ; length ; lower-bound;
   # upper-bound
   ```

Die Wahl des Modells ((TTVS1) oder (TTVS2)) geschieht über den Parameter `vs_model`, die Fahrzeugkosten werden über `vs_vehicle_cost` angegeben und die maximale Anzahl von Fahrzeugen gibt der Parameter `vs_number_vehicles` an. Des Weiteren wird in der Implementierung von (TTVS2) allezeit davon ausgegangen, dass sich das Depot an einer Station des PTNs befindet, deren Nummer dem Parameter `vs_depot_index` zugeordnet wird.

Die Ausgabe der Heuristiken erfolgt folgendermaßen:

1. Der Fahrplan wird in die durch den Parameter `default_timetable_file` spezifizierte Datei geschrieben, welche im Format

   ```
   # event-id; time
   ```

   vorliegt.

2. Der Umlaufplan wird in die dem Parameter `default_vehicle_schedule_file` zugeordnete Datei ausgegeben. Die Spalten der Datei tragen die Bezeichnungen:

   ```
   # circulation-id; vehicle-id; trip-number of this vehicle; trip-id;
   # start-id; periodic-start-id; start-station; start-time; end-id;
   # periodic-end-id; end-station; end-time; line
   ```

Dabei werden anders – als bei den bisher entwickelten Methoden – keine Leerfahrten explizit aufgeführt.

3.4.3 Heuristik zur Lösung von (TTVS1) und (TTVS2) in LinTim

Im Rahmen dieser Bachelorarbeit wurden zwei Versionen der Heuristik programmiert. Die erste Version TimVS wandelt die Eingabe im oben beschriebenen LinTim-Format entsprechend dem Beweis zu Lemma 3.10 in eine Eingabe für Algorithmus 5 um, startet daraufhin das in Abschnitt 3.4.1 beschriebene Programm und übersetzt die Ausgabe zurück ins LinTim-Format. Da die Eingabedaten in der Form, wie Algorithmus 5 sie benötigt, allerdings viele Redundanzen enthalten, ist diese Version sehr speicheraufwändig. Außerdem benötigen die Matrix-Vektor-Multiplikationen viel Zeit. Deshalb wurde eine zweite Version TimVSb entwickelt, welche problemspezifische Optimierungen enthält und die Entwicklung der Zielfunktionswerte im Laufe der Iterationen in der Datei obj.res protokolliert. Wir stellen zuerst die erste Version vor und gehen in der Folge auf die Anpassungen in der zweiten Version ein.

Die erste Version besteht im Wesentlichen aus folgenden fünf Methodenaufrufen.

1. `convertInput()`: Übersetzt die LinTim-Eingabedateien in die in Abschnitt 3.4.1 beschriebenen Eingabedateien für das Programm `Algo5`.

2. `generatePi()`: Erzeugt zulässige Startlösung $\pi^{(0)}$ und speichert diese in die Datei pi.dat.

3. `Algo5.run()`: Oben beschriebene Methode, führt Algorithmus 5 durch.

4. `ArrayList<Integer> pi = piToTimetable()`: Übersetzt die in der Datei pi.dat stehende Ausgabe in einen Fahrplan im LinTim-Format.

5. `zToVehicleSchedule(pi)`: Übersetzt die in der Datei z.dat stehende Ausgabe in einen Umlaufplan im LinTim-Format.

Zu `convertInput()` Die meisten der oben beschriebenen Dateien werden in LinTim bereits an anderer Stelle (z. B. in der gewöhnlichen Umlaufplanung) als Eingabe verwendet, weshalb für diese schon Methoden zum Einlesen implementiert sind. Diese Methoden übersetzen den Inhalt der Dateien in Objekte in Java; so wird z. B. beim Einlesen der Datei, welche die Fahrten enthält, für jede Zeile ein Objekt der Klasse Trip erzeugt, welches die Informationen aus den verschiedenen Spalten der Datei enthält. Diese Einlesemethoden sowie die dazugehörigen Klassen zur Repräsentation der einzulesenden Entitäten, wie z. B. Trip oder NonPeriodicAvtivity, konnten daher übernommen werden. Einzig für die Kompatibilitäten und die Abstände mussten neue Einlesemethoden implementiert werden. Nach dem Einlesen liegen sämtliche Daten in Form von Java-Objekten vor. Daraus werden anschließend die Eingabedaten wie im Beweis zu Lemma 3.10 erzeugt und in die in Abschnitt 3.4.1 vorgegebenen Dateien geschrieben.

Zu `generatePi()` Ein zulässiger Umlaufplan wird generiert, indem das Problem (TT1) mit geänderter Zielfunktion gelöst wird. Die Zielfunktionskoeffizienten `zf` werden auf drei unterschiedliche Arten bestimmt:

(IS1) Mit folgendem Programmfragment:

```
1  double[] zf = new double[numberEvents];
2  double average = 0.0;
3  for(NonPeriodicActivity a : Act)
4     average += a.getWeight();
5  average /= Act.size();
6  System.out.println("average:␣"+average);
7  for(NonPeriodicActivity a : Act) {
8     double factor = Math.random();
9     zf[a.getSource().getID()-1]-=a.getWeight()*factor-average
         /2;
10    zf[a.getTarget().getID()-1]+=a.getWeight()*factor-average
         /2;
11 }
```

Die Koeffizienten c_a, $a \in \mathcal{A}$ werden darin mit einer zufälligen Zahl aus dem Intervall $[0, 1]$ multipliziert und anschließend um den halben Mittelwert verringert. Die so erhaltenen Werte haben den erwarteten Mittelwert 0. Es wird auf diese Weise erreicht, dass die Dauer einiger Aktivitäten minimiert, die anderer maximiert wird, wobei die Wahrscheinlichkeit, dass die Dauer einer Aktivität minimiert wird, mit steigender Anzahl der Passagiere, die diese Aktivität nutzen, zunimmt.

(IS2) Es wurde genauso wie in (IS1) verfahren, allerdings werden alle Einträge mit -1 multipliziert. Dadurch nimmt die Wahrscheinlichkeit, dass die Zeit einer Aktivität minimiert wird, mit steigender Anzahl der Fahrgäste, die diese nutzen, ab.

(IS3) Es wird die originale Zielfunktion von (TT1) verwendet.

Zu `zToVehicleSchedule()` Um aus dem binären Vektor z einen Umlaufplan in dem in Abschnitt 3.4.2 beschriebenen Format zu erzeugen, muss eine Variante von Algorithmus 1 implementiert werden. Diese speichert für jede Fahrt in einem Feld `tripCovered[]`, ob diese bereits durch eine Fahrzeugroute abgedeckt wurde. Es wird über alle Fahrten iteriert. Falls eine Fahrt noch nicht abgedeckt ist, wird in Zeile 17 die Methode `coverTrip()` aufgerufen, welche die Route erzeugt, die diese Fahrt abdeckt. Um beim Erzeugen der Route Vorgänger- und Nachfolgerfahrten zu bestimmen, wird die Liste `chosenCompatibilities` der stattfindenden Kompatibilitäten durchsucht. Die Methode `coverTrip()` bestimmt zu einer gegebenen Fahrt zuerst den Teil der Fahrzeugroute bis zu dieser Fahrt (Zeilen 28-46) und anschließend den Teil der Fahrzeugroute nach dieser Fahrt (Zeilen 50-67). Dabei verwendete Kompatibilitäten werden später nicht mehr benötigt und deshalb zur Beschleunigung späterer Suchen aus der Liste textttchosenCompatibilities entfernt. Der Umlaufplan wird zunächst als Array von Listen mit Trips gespeichert. Dieser wird dann von der Methode `IO.outputVehicleSchedule()` in eine Datei übersetzt. Um die Abfahrts- und Ankunftszeiten korrekt auszugeben, greift diese auch auf den Fahrplan `pi` zu.

```
1  private static void zToVehicleSchedule(List<Integer> pi) throws
       Exception {
2     ArrayList<Integer> z = readVector("z");
3     ArrayList<Compatibility> chosenCompatibilities = new ArrayList<
         Compatibility>();
4
5     for(Compatibility c : Comp)
6        if(z.get(c.getID()-1) == 1)
7           chosenCompatibilities.add(c);
8
9     int numberVehicles = T.size() - chosenCompatibilities.size();
10    ArrayList<Trip>[] vs = new ArrayList[numberVehicles];
11    boolean tripCovered[] = new boolean[T.size()];
12    int vehicleID = 0;
13
14    try{
15       for(Trip t : T)
16          if(!tripCovered[t.getID()-1])
17             vs[vehicleID++] = coverTrip(t, chosenCompatibilities,
                  tripCovered);
18
19       IO.outputVehicleSchedule(vs, pi);
20    } catch(IndexOutOfBoundsException e) {
21       System.out.println("Fehler:␣Vermutlich␣gibt␣es␣im␣Trip-
            Graphen␣einen␣Kreis␣mit␣Fahrzeit␣0.");
22       throw e;
23    }
24 }
25
```

```
26 private static ArrayList<Trip> coverTrip(Trip t, ArrayList<
      Compatibility> chosenCompatibilities, boolean[] tripCovered) {
27
28    // find first trip
29    Trip start = t;
30    boolean noPredecessor = false;
31    ArrayList<Trip> route = new ArrayList<Trip>();
32    while(!noPredecessor) {
33       tripCovered[start.getID()-1] = true;
34       route.add(start);
35       noPredecessor = true;
36       Iterator<Compatibility> cCIt = chosenCompatibilities.
            iterator();
37       while(cCIt.hasNext()) {
38          Compatibility c = cCIt.next();
39          if(c.getTarget() == start) {
40             start = c.getSource();
41             noPredecessor = false;
42             cCIt.remove();
43             break;
44          }
45       }
46    }
47
48    Collections.reverse(route);
49
50    // find last trip
51    Trip end = t;
52    boolean noSuccessor = false;
53    while(!noSuccessor) {
54       noSuccessor = true;
55       Iterator<Compatibility> cCIt = chosenCompatibilities.
            iterator();
56       while(cCIt.hasNext()) {
57          Compatibility c = cCIt.next();
58          if(c.getSource() == end) {
59             end = c.getTarget();
60             noSuccessor = false;
61             cCIt.remove();
62             tripCovered[end.getID()-1] = true;
63             route.add(end);
64             break;
65          }
66       }
67    }
```

```
68
69      return route;
70  }
```

Es werden kurz die Optimierungen vorgestellt, die in der zweiten Version TimVSb vorgenommen wurden.

1. Die untere Hälfte der für den Algorithmus 5 konstruierten Matrix A unterscheidet sich von der oberen genau um den Faktor -1 (siehe Beweis von Lemma 3.10). Außerdem enthält jede Zeile neben Nullen nur genau eine 1 und eine -1. Es reicht also, die Hälfte der Zeilen und für jede dieser Zeilen lediglich die beiden Spaltenindizes der 1 und der -1 zu speichern, d. h. genauso wie in der Aktivitäten-Datei werden für jede Aktivität die Indizes von Anfangs- und Endereignis gespeichert.

2. Anstatt des im Beweis von Lemma 3.10 zusammengesetzten Vektors U werden die oberen und die unteren Schranken der Aktivitäten in zwei getrennten Vektoren L und U gespeichert.

3. Auch Matrix B enthält redundante Information. Anstelle der Matrix werden in TimVSb für jede Kompatibilität die Indizes der vorherigen und der nachfolgenden Fahrt gespeichert. Die Indizes der Kanten im Trip-Graphen zwischen Depot und einer Fahrt und umgekehrt müssen nicht gespeichert werden, da die Variablen stets so sortiert sind, dass es sich dabei um die letzten m_{II} Indizes handelt.

4. Im Fall von (TTVS2) wird statt des Vektors V nur die maximale Anzahl an Fahrzeugen gespeichert.

5. Für die Matrix D wird analog zur Matrix A verfahren: Für jede Kompatibilität werden die Indizes des Ankunftsereignisses der Vorgängerfahrt und des Abfahrtsereignisses der Folgefahrt gespeichert.

6. Für de Vektor W wird in gleicher Weise wie für U vorgegangen: Die unteren Schranken werden in einem Vektor M, die oberen in einem Vektor W gespeichert.

7. Die Bestimmung der Indizes für ($IP1_\pi$), an denen Fahrzeugübergänge möglich sind, wurde aufgrund der hohen Laufzeit aus *Mosel* in das *Java*-Programm ausgelagert.

Diese Änderungen des Eingabeformats machen eine Anpassung der *Mosel*-Programme erforderlich. In diesem Zusammenhang wurde auch das *Mosel*-Programm zur Lösung des Umlaufplanproblems durch zwei *Mosel*-Programme – eines für (TTVS1) und eines für (TTVS2) – ersetzt. Es folgt ein Auszug aus dem Quellcode für (TTVS2). Das Programm startet mit `chosenIndices` als Eingabe und gibt neben `zSol` auch `objvalue` aus.

```
forall(j in chosenIndices) do
  z(j) is_binary
end-do

forall(i in 1..numberTrips) do
  sum(j in chosenIndices | B(0,j) = i) z(j) = 1
  sum(j in chosenIndices | B(1,j) = i) z(j) = 1
end-do

sum(j in rowsD+1 .. dimZ) z(j) <= 2*numberVehicles

minimize(sum(j in chosenIndices) C(j) * z(j))

forall(j in 1..dimZ) do
  if(j in chosenIndices) then
    zSol(j) := getsol(z(j))
  else
    zsol(j) := 0
  end-if
end-do

objvalue := getobjval
```

Ähnliche Anpassungen für den Fahrplanoptimierungsschritt führen zu folgendem Code:

```
forall(j in 1..dimPi) pi(j) is_integer

forall(i in 1..rowsA) do
  pi(A(i,1)) - pi(A(i,0)) <= U(i)
  pi(A(i,1)) - pi(A(i,0)) >= L(i)
end-do

forall(i in 1..rowsD)
  if(z(i) = 1) then
    pi(D(i,1))  - pi(D(i,0)) <= W(i)
    pi(D(i,1))  - pi(D(i,0)) >= M(i)
  end-if

minimize(sum(j in 1..dimPi) c(j) * pi(j))

objvalue := getobjval
```

Neben den oben beschriebenen Anpassungen zur besseren Nutzung von Speicher und Zeit wurde das Programm um eine Funktion zum Protokollieren der Zielfunktionswerte erwei-

tert. Dadurch ergibt sich für die Methode zur Durchführung von Algorithmus 5 folgender Programmcode:

```
1  ArrayList<Double> objZ = new ArrayList<Double>();
2  ArrayList<Double> objPi = new ArrayList<Double>();
3  int iterations = 0;
4
5  readObj(objPi);
6  do {
7     System.out.println("Iteration␣"+(++iterations));
8     calcChosenIndices(D,M,W,dimZ);
9     switch (model) {
10       case 1: Mosel.exec(SOURCEPATH+"IP1piVS1.mos");
11               break;
12       case 2: Mosel.exec(SOURCEPATH+"IP1piVS2.mos");
13               break;
14    }
15    readObj(objZ);
16    Mosel.exec(SOURCEPATH+"IP1zb.mos");
17    readObj(objPi);
18 } while (solutionChanged() && MAX_ITERATIONS > iterations);
19 outputTable(objPi, objZ, "obj.res");
```

Die Methode `readObj()` fügt den Zielfunktionswert der aktuellen Iteration in die übergebene Liste ein; die Methode `outputTable` erzeugt aus den beiden übergebenen Listen die Datei obj.res, welche für jeden Zeitpunkt (d. h. nach jedem gelösten IP) die beiden Zielfunktionswerte enthält. Anhand der Datei pi.dat berechnet die Methode `calcChosenIndices` die Menge der Indizes, an denen Übergänge zwischen Fahrten stattfinden können und speichert diese in der Datei chosenCompatibilities.dat. TTVSb wird mit dem *make*-Target `tim-vs` gestartet.

3.4.4 Auswertungsroutine

Um die Implementation zu testen und die Ergebnisse der oben beschriebenen Heuristik mit denen der bisher in LinTim vorhandenen Algorithmen zu vergleichen, wurde eine Methode zum Auswerten von Fahr- und Umlaufplan geschrieben, welche mit dem *make*-Target `tim-vs-evaluate` gestartet werden kann. Sie überprüft den in der Datei `default_timetable_file` gespeicherten Fahrplan auf Zulässigkeit und berechnet die zugehörige Gesamtreisezeit. Ferner überprüft die Methode den in der Datei `default_vehicle_schedule_file` gespeicherten Umlaufplan unter der Annahme, dass alle Paare von Fahrten prinzipiell kompatibel sind, auf Zulässigkeit und berechnet sowohl die Fahrzeuganzahl als auch die Kosten des Umlaufplans.

3.4.5 Ergebnisse

Die oben beschriebene Heuristik wurde auf Eingabeinstanzen, welche mit LinTim aus den vorgegebenen PTNs bahn-01 und bahn-02 erzeugt wurden, getestet. Für diese wurden mit LinTim nacheinander Linienkonzept, aperiodisches EAN und Fahrtenmenge mittels der *make*-Targets `lc-line-concept`, `ptn2ean`, `tim-timetable` und dem oben beschriebenen selbst entwickelten Target `tim-vs-trips`. Dabei wurden die folgenden standardmäßig eingestellten Parameter verwendet:

```
lc_model;                                   "cost"
lc_minimal_global_frequency;                0
lc_maximal_global_frequency;                6
period_length;                              60
time_units_per_minute;                      1
ean_model_frequency;                        "FREQUENCY_AS_ATTRIBUTE"
ean_model_weight_drive ;                    "AVERAGE_DRIVING_TIME"
ean_model_weight_change;                    "FORMULA_1"
ean_model_weight_wait;                      "ZERO_COST"
ean_initial_duration_assumption_model;      "AUTOMATIC"
tim_concept;                                 aperiodic
tim_solver;                                  "xpress"
tim_model;                                   "csp_ns"
tim_use_old_solution;                        false
tim_fix_old_modulo;                          false
tim_nws_loc_search;                          "CONNECTED_CUT"
tim_nws_tab_search;                          "TAB_PERCENTAGE"
tim_nws_ts_memory;                           40
tim_nws_ts_max_iterations;                   150
tim_nws_sa_init;                             50000
tim_nws_sa_cooldown;                         0.95
tim_nws_percentage;                          80
tim_nws_min_pivot;                           0.1
tim_nws_dyn_pivot;                           0.5
tim_nws_seed;                                0
tim_nws_limit;                               0
tim_nws_timelimit;                           0
tim_nws_use_robustness;                      false
tim_nws_min_robustness;                      0
rollout_whole_trips;                         false
DM_earliest_time;                            28800
DM_latest_time;                              43200
```

Die letzten beiden Werte geben an, dass für den Zeitraum von 8 bis 12 Uhr geplant wird (die Angabe erfolgt in LinTim in Sekunden), d. h. $D = 240$ (in Minuten). Weiterhin wurden die zu den Instanzen bahn-01 und bahn-02 gehörenden Standardeinstellungen verwendet.

Auf den beiden so erzeugten Eingaben wurde die verbesserte Heuristik TimVSb, welche intern Algorithmus 5 verwendet, sowohl für (TTVS1) als auch für (TTVS2) getestet. Dabei wurden alle drei oben beschriebenen Verfahren (IS1), (IS2) und (IS3) zur Bestimmung einer Startlösung ausprobiert. Sofern nicht anders angegeben, wurden folgende Parameterwerte verwendet.

```
vs_vehicle_costs;          1000
vs_min_distance;           1
vs_max_distance;           10000
vs_number_vehicles;        100000
vs_cost_factor;            1.0
vs_depot_index;            1
tim_vs_given_compatibilities; false
tim_vs_all_compatibilities;   true
```

Sofern nicht anders angegeben, wurde der Algorithmus abgebrochen, falls eine bereits erhaltene Lösung erneut als Ergebnis berechnet wird, spätestens jedoch nach 200 Schritten (d. h. 100 vollständigen Schleifendurchläufen).

Entwicklung der Zielfunktionswerte

Abbildung 3.9 zeigt beispielhaft die ersten 50 Schritte der Entwicklung der Zielfunktionswerte einer Ausführung der Heuristik für (TTVS2) auf bahn-02. Für den Iterationswert 0 werden die Zielfunktionswerte für die Startlösung $\pi^{(0)}$ und für den zugehörigen Umlaufplan $z^{(1)}$ aufgetragen, für den Iterationswert 1 die Zielfunktionswerte für $\pi^{(1)}$ und $z^{(1)}$, usw. Die Zielfunktionswerte an jeder Stelle gehören also zu einem Paar von zueinander passendem Fahrplan und Umlaufplan.

Wie in Lemma 3.13 gezeigt, ist es nicht möglich, dass sich einer der Zielfunktionswerte im Laufe des Algorithmus verschlechtert. Es wird aber deutlich, dass sich in den meisten Schritten beide Zielfunktionswerte nicht ändern. Dies kommt dadurch zustande, dass es mehrere Fahr- oder Umlaufpläne mit gleichem Zielfunktionswert gibt. Nachdem beispielsweise zu einem Fahrplan ein Umlaufplan berechnet wurde, ist es möglich, dass der Solver aufgrund der geänderten Nebenbedingungen eine andere Optimallösung mit gleichem Zielfunktionswert findet. Da sich nun der Fahrplan geändert hat, kann sich im nächsten Schritt wieder der Umlaufplan ändern, usw. Dabei kann nach mehreren Schritten ohne Änderung eine Verbesserung der Zielfunktionswerte eintreten. In dem in Abbildung 3.9 visualisierten Rechengang wurde keine Beschränkung der Iterationenzahl gemacht. Die Rechnung endete

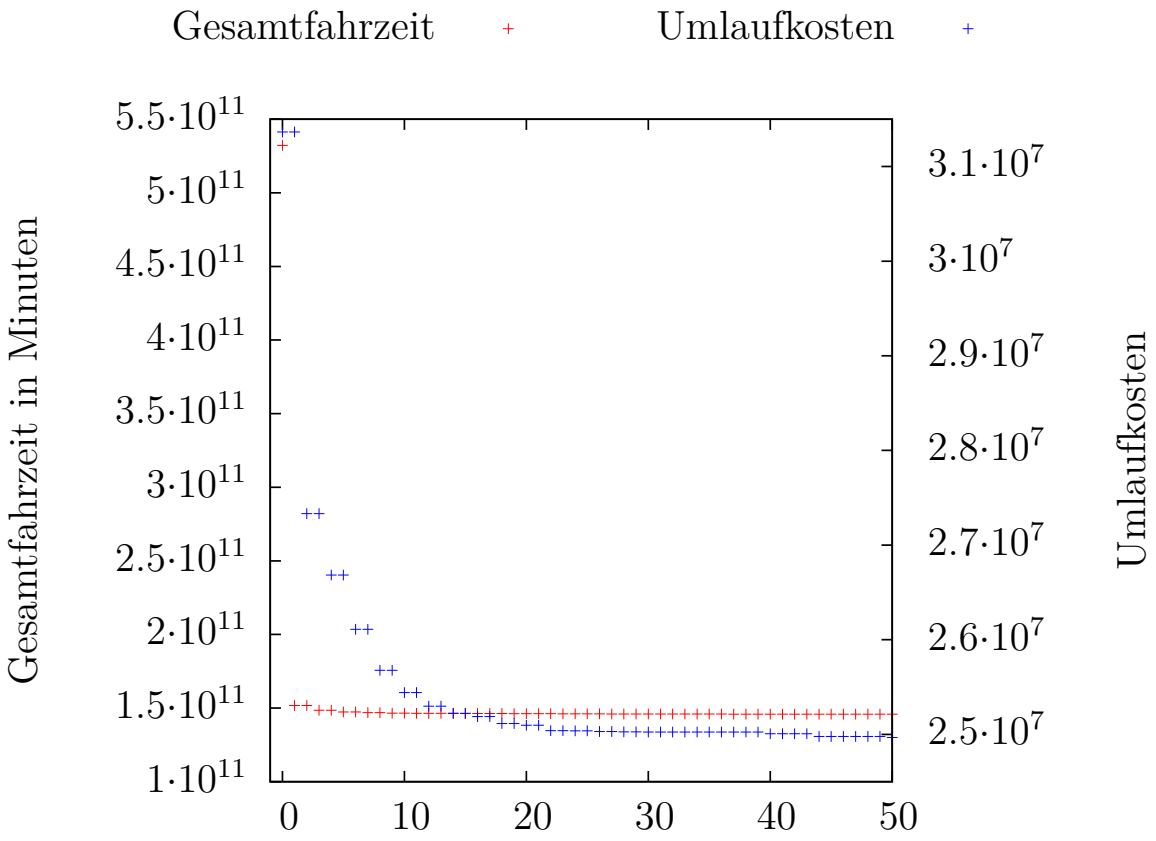

Abb. 3.9: Entwicklung der Zielfunktionswerte während der ersten 50 Iterationen auf bahn-02 für Modell (TTVS2) und Startlösung nach (IS1)

nach 11098 Schritten, d. h. nach 5549 vollständigen Schleifendurchläufen. Die letzte Änderung eines Zielfunktionswertes fand im 2110. Schritt statt, in dem sich die Gesamtfahrzeit von $1,45818 \cdot 10^{11}$ auf $1,45817 \cdot 10^{11}$ verringerte; die Umlaufkosten änderten sich zum letzten Mal im 111. Schritt von $2,4966 \cdot 10^7$ auf $2,49648 \cdot 10^7$.

Es fällt auf, dass sich die Fahrzeiten im ersten Schritt sehr viel stärker (Faktor $\approx 72\%$) verbessern als die Umlaufkosten (Faktor $\approx 13\%$). Dies ist dadurch begründet, dass der Startfahrplan $\pi^{(0)}$ bei dem Verfahren (IS1) mit einer von der eigentlichen Zielfunktion abweichenden Zielfunktion gewonnen wurde, wohingegen der erste Umlaufplan $z^{(1)}$ bereits als Lösung von (IP1$_\pi$) mit der richtigen Zielfunktion berechnet wurde.

In Abbildung 3.10 werden die beiden Zielfunktionswerte Fahrzeuganzahl und Gesamtfahrzeit für jeden der ersten 200 Schritte der Heuristik für (TTVS1) auf bahn-01 gegeneinander aufgetragen. Der Punkt, an dem Gesamtfahrzeit und Fahrzeuganzahl am größten sind, gehört zu dem Startfahrplan $\pi^{(0)}$ und einem besten dazu passenden Umlaufplan $z^{(1)}$. Auch in diesem Fall verbessert sich der Fahrplan im ersten Schritt stark. Auffällig ist, dass sich danach der Umlaufplan ebenfalls deutlich verbessert, d. h. dass es für einen besseren Fahr-

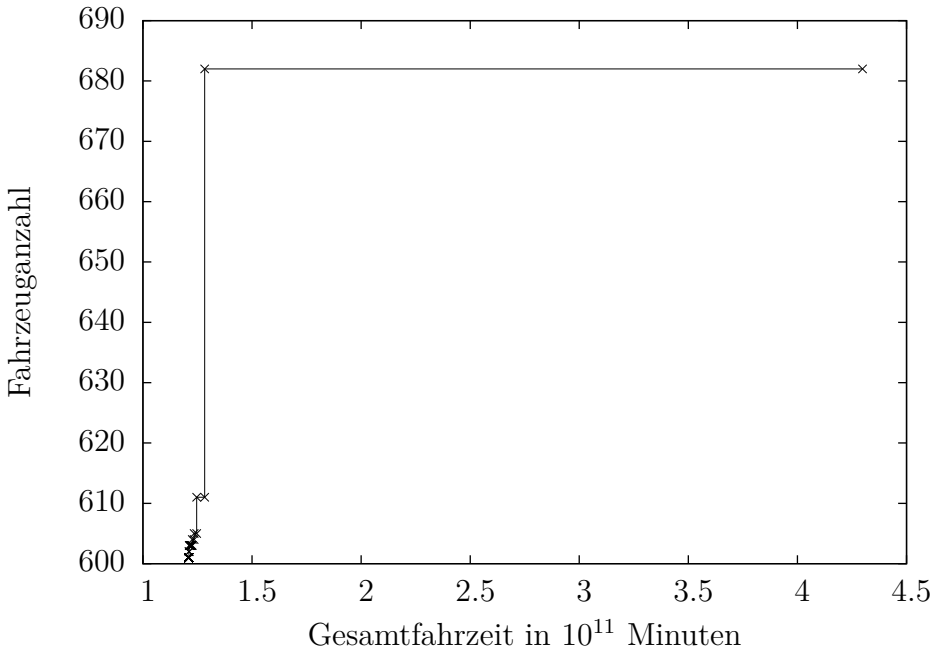

Abb. 3.10: Entwicklung der Zielfunktionswerte auf bahn-01 für Modell (TTVS1) und Startlösung nach (IS1)

plan auch einen besseren Umlaufplan gibt. Dies ist auch in allen anderen Testinstanzen zu beobachten, woraus sich schließen lässt, dass zu Beginn des Algorithmus verbesserte Fahrpläne auch bessere Umlaufpläne ermöglichen. Aus diesem Grund ist es in jedem Testlauf nach Berechnung von $(\pi^{(1)}, z^{(1)})$ zu einer weiteren Verbesserung der Zielfunktionswerte gekommen.

Abhängigkeit von der Startlösung bei (IS1)

Sowohl bei bahn-02 als auch bei bahn-01 unterschieden sich für unterschiedliche Startlösungen nicht nur die Endlösungen, sondern auch deren Zielfunktionswerte. Abbildung 3.11 zeigt für fünf Ausführungen des Algorithmus für (TTVS1) auf bahn-01 mit unterschiedlichen mittels (IS1) bestimmten Startlösungen die Zielfunktionswerte der Startlösung und der Endlösung. Es zeigt sich, dass Startlösungen mit geringeren Fahrzeugbedarf nicht unbedingt zu Endlösungen mit geringerem Fahrzeugbedarf führen.

Für die Gesamtfahrzeit lässt sich auf der Abbildung aufgrund der verhältnismäßig großen Verbesserung im ersten Schritt nicht viel erkennen, weshalb in Abbildung 3.12 für die gleichen Ausführungen des Algorithmus die Zielfunktionswerte aller Schritte, beginnend mit den Lösungen $(\pi^{(1)}, z^{(1)})$, welche nach dem ersten Fahrplanoptimierungsschritt feststehen, aufgetragen sind.

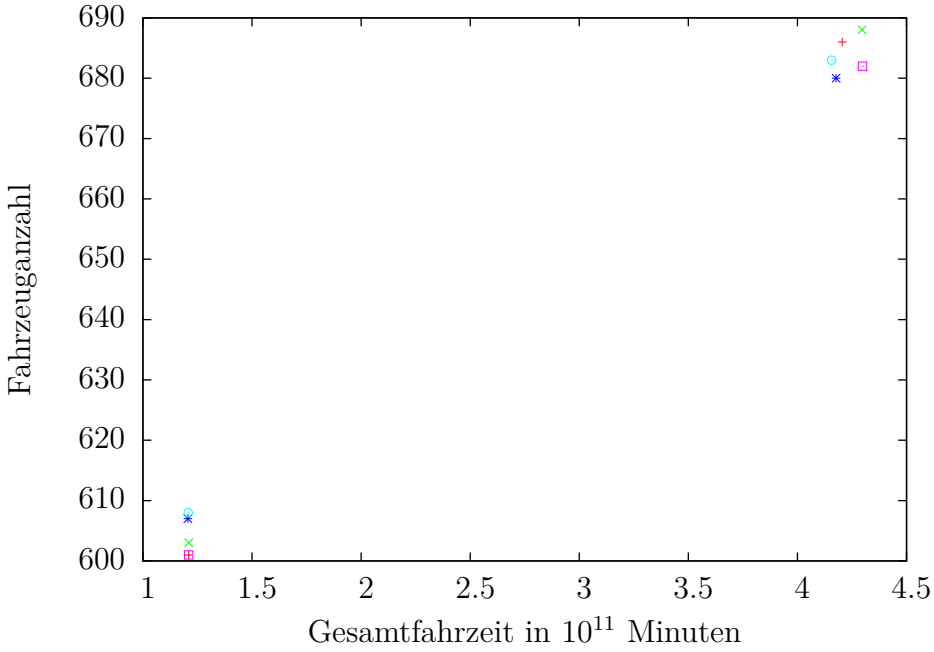

Abb. 3.11: Zielfunktionswerte der Startlösung $(\pi^{(0)}, z^{(1)})$ und der Endlösung für fünf Ausführungen der Heuristik für (TTVS1) mit unterschiedlichen mit (IS1) bestimmten Startlösungen

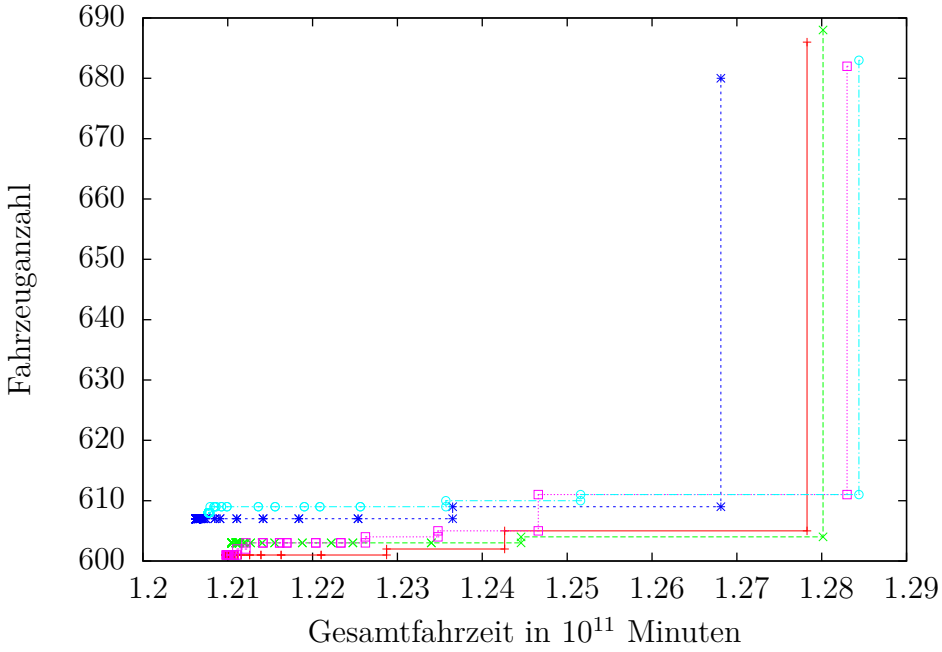

Abb. 3.12: Zielfunktionswerte für fünf Ausführungen der Heuristik für (TTVS1) mit unterschiedlichen mit (IS1) bestimmten Startlösungen beginnend mit $(\pi^{(1)}, z^{(1)})$

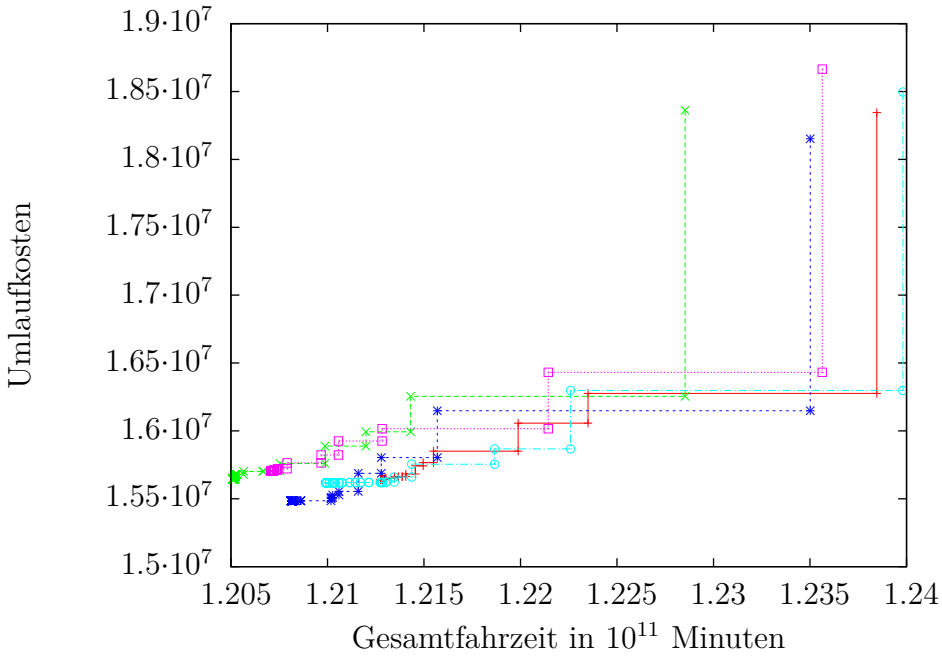

Abb. 3.13: Zielfunktionswerte für fünf Ausführungen der Heuristik für (TTVS2) mit unterschiedlichen mit (IS1) bestimmten Startlösungen beginnend mit $(\pi^{(1)}, z^{(1)})$

Für (TTVS2) wird die Entwicklung der Zielfunktionswerte für fünf Ausführungen, beginnend mit den Lösungen $(\pi^{(1)}, z^{(1)})$, in Abbildung 3.13 dargestellt. Es ist kein direkter Zusammenhang zwischen der Gesamtfahrzeit von $\pi^{(1)}$ und der Endlösung erkennbar, d. h. Ausführungen, deren Startlösung eine höhere Gesamtfahrzeit haben, führen nicht unbedingt zu Endlösungen mit höheren Gesamtfahrzeiten.

Vergleich von (TTVS1) mit (TTVS2)

Beim Vergleich der beiden Abbildungen 3.12 und 3.13 fällt auf, dass in jeder Ausführung die Verbesserung des Zielfunktionswertes Fahrzeuganzahl bzw. Umlaufkosten bei der Berechnung des zweiten Umlaufplans im Vergleich zur Gesamtverbesserung bei (TTVS1) größer als bei (TTVS2) ist. Außerdem finden in (TTVS1) in späteren Iterationen häufig nur noch Verbesserungen der Reisezeit statt, wohingegen sich in (TTVS2) auch später noch beide Zielfunktionswerte verringern. Erklärt werden können diese Unterschiede damit, dass die Fahrzeuganzahl relativ klein ist und ganzzahlig sein muss. Es wäre also nur eine absolute Verbesserung um mindestens 1 möglich, was einer prozentualen Verbesserung um ca. $0,17\%$ entspräche. Die relativen Verbesserungen der Umlaufkosten bei (TTVS2) in den späteren Iterationen liegen hingegen unter $0,1\%$.

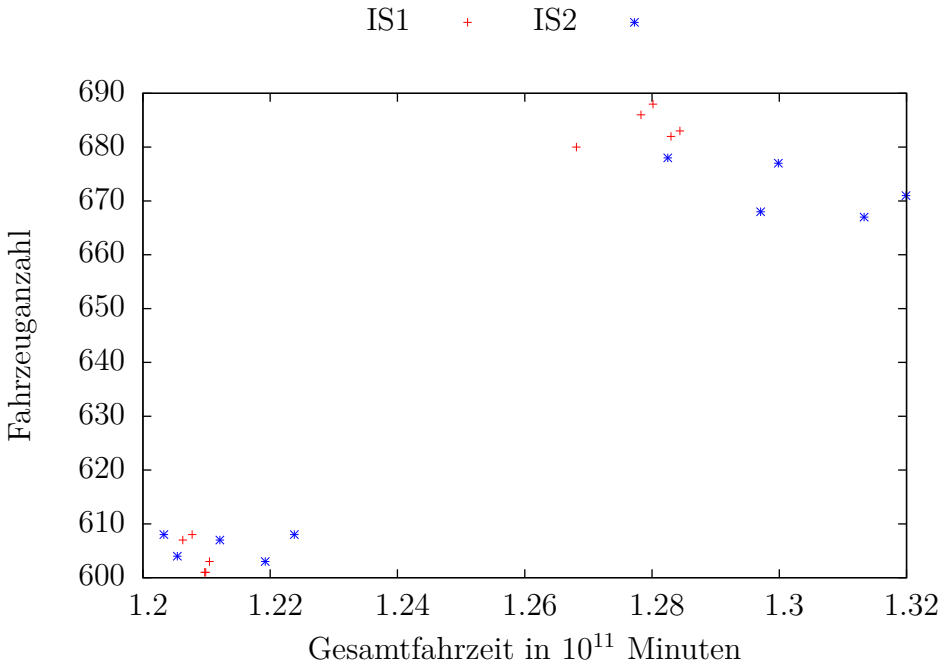

Abb. 3.14: Zielfunktionswerte der Lösung $(\pi^{(1)}, z^{(1)})$ und der Endlösung der Heuristik für (TTVS1) mit (IS1) und (IS2)

Vergleich von (IS1) mit (IS2)

Da die in (IS2) zur Bestimmung einer Startlösung $\pi^{(0)}$ verwendete Zielfunktion stärker von der eigentlichen Zielfunktion abweicht als bei (IS1), haben die mit (IS2) bestimmten Startlösungen einen deutlich schlechteren Zielfunktionswert $c^T\pi^{(0)}$ als die mit (IS1) bestimmten: Während dieser für bahn-01 bei (IS1) meist im Bereich von $4,25 \cdot 10^{11}$ Minuten liegt, liegt er bei (IS2) bei ungefähr $6 \cdot 10^{11}$ Minuten. In Abbildung 3.14 sind die Zielfunktionswerte der Lösungen $(\pi^{(1)}, z^{(1)})$ und der Endlösungen nach Abbruch des Algorithmus (maximal 200 Schritte) für (TTVS1) für mit (IS1) und mit (IS2) generierte Startlösungen dargestellt. Es fällt auf, dass sich bereits nach dem ersten Fahrplanoptimierungsschritt der Unterschied deutlich verringert hat. Nach Ablauf des Algorithmus sind keine Unterschiede mehr festzustellen.

Analog zur Heuristik für (TTVS1) zeigt Abbildung 3.15 dies für die Heuristik für (TTVS2).

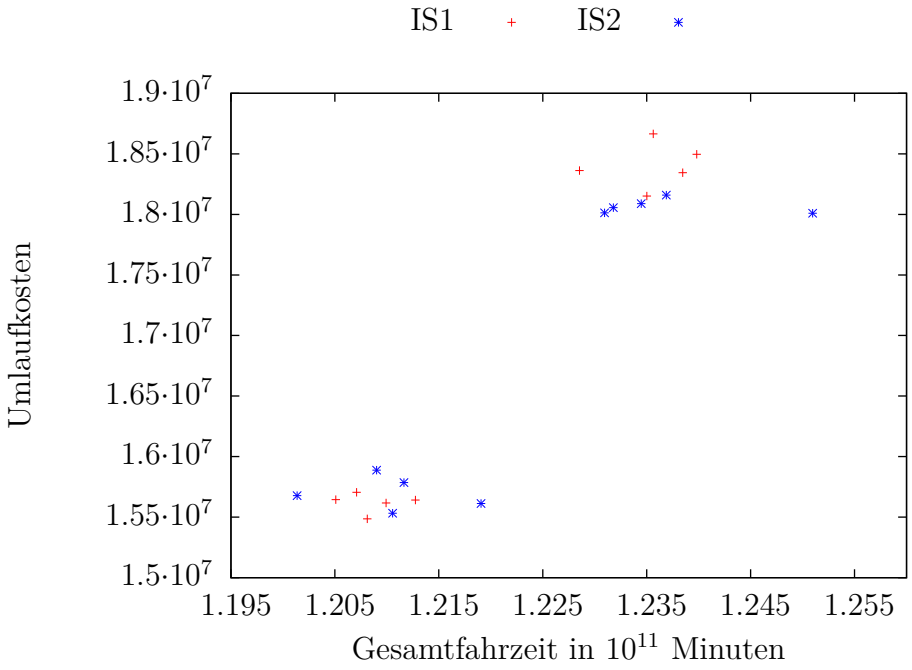

Abb. 3.15: Zielfunktionswerte der Lösung $(\pi^{(1)}, z^{(1)})$ und der Endlösung der Heuristik für (TTVS2) mit (IS1) und (IS2)

Vergleich von $\mathcal{C} = \mathcal{C}_\alpha := \mathcal{T}^2$ mit $\mathcal{C} = \mathcal{C}_\beta := \{(t_1, t_2) \mid \text{end}(t_1) = \text{start}(t_2)\}$

Wie in Abschnitt 3.4.2 beschrieben, wird der Parameter `tim_vs_all_compatibilities` dazu verwendet, einzustellen, ob potenziell alle Paare von Fahrten kompatibel sind, oder nur diejenigen, für welche die Endstation der Vorgängerfahrt mit der Anfangsstation der Nachfolgerfahrt übereinstimmt. In Abbildung 3.16 sind die Lösung $(\pi^{(1)}, z^{(1)})$ und die Endlösung für beide Kompatibilitätsrelationen für jeweils 5 Ausführungen des Algorithmus für (TTVS1) auf bahn-01 mit (IS1) dargestellt; Abbildung 3.17 zeigt Entsprechendes für (TTVS2). Es stellt sich heraus, dass die Anzahl der benötigten Fahrzeuge bzw. die Umlaufkosten wie zu erwarten für $\mathcal{C} = \mathcal{C}_\beta$ höher sind als für $\mathcal{C} = \mathcal{C}_\alpha$, da es weniger Freiheiten bei der Optimierung des Umlaufplans gibt. Aus dem gleichen Grund terminiert Algorithmus 5 für $\mathcal{C}_\beta$ nach sehr viel weniger Schritten. Obwohl der Zielfunktionswert der Fahrpläne nach Ablauf des Algorithmus für beide Kompatibilitätsrelationen ungefähr gleich ist, ist der Zielfunktionswert der Lösung $\pi^{(1)}$ für $\mathcal{C}_\beta$ sehr viel niedriger als für $\mathcal{C}_\alpha$. Dies kann damit erklärt werden, dass der erste Umlaufplan für $\mathcal{C}_\beta$ sehr viel weniger Einschränkungen an den Fahrplan stellt als der für $\mathcal{C}_\alpha$.

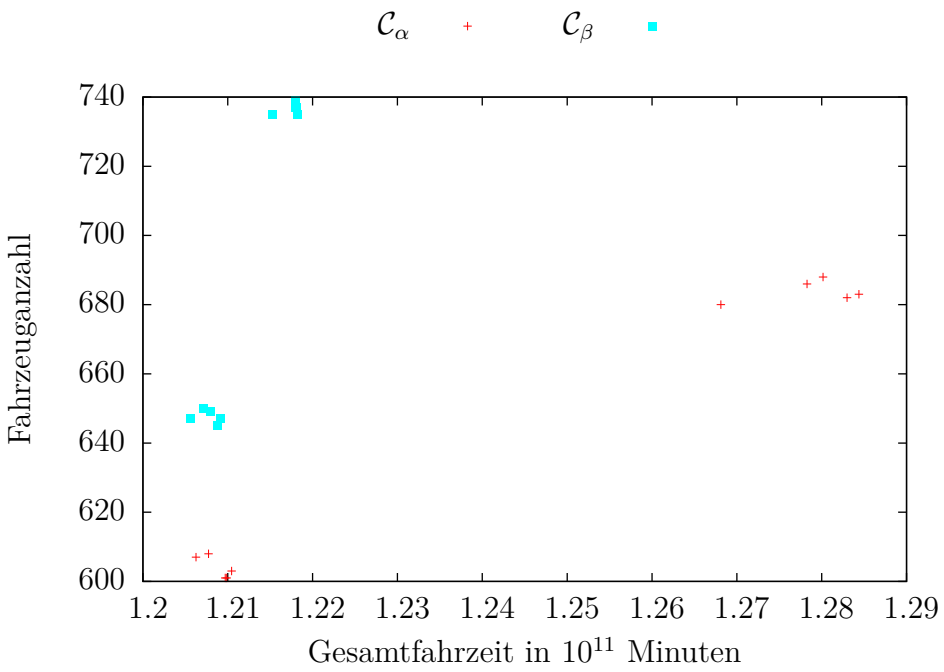

Abb. 3.16: Zielfunktionswerte der Lösung $(\pi^{(1)}, z^{(1)})$ und der Endlösung der Heuristik für (TTVS1) mit (IS1) und $\mathcal{C}_\alpha$ und $\mathcal{C}_\beta$

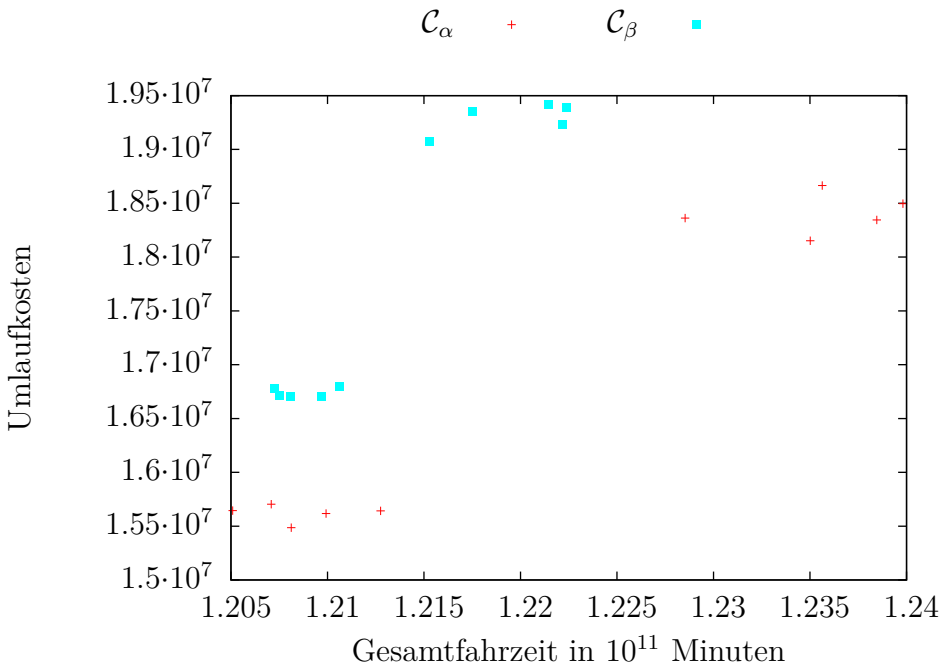

Abb. 3.17: Zielfunktionswerte der Lösung $(\pi^{(1)}, z^{(1)})$ und der Endlösung der Heuristik für (TTVS2) mit (IS1)

Startlösung mit (IS3)

Im Laufe dieser Arbeit hat sich herausgestellt, dass der in Lemma 3.7 beschriebene Fall, dass nach Änderung des Umlaufplans wieder der gleiche Fahrplan wie vorher gefunden wird, selten eintritt. Selbst wenn sich der Fahrplan durch Änderung des Umlaufplans nicht verbessern lässt, wird meist aufgrund der anderen Nebenbedingungen doch ein anderer Fahrplan gefunden. Deshalb wurde auch Algorithmus 2 getestet, welchen man erhält, wenn als Startlösung ein optimaler Fahrplan verwendet wird. Dieses Vorgehen wird, wie oben beschrieben, in diesem Abschnitt als (IS3) bezeichnet.

Vergleich mit bisher implementierten Methoden

Die bisherige Methode, einen Fahrplan und einen Umlaufplan zu erstellen, besteht darin, die drei *make*-Targets `tim-timetable`, `ro-trips` und `vs-vehicle-schedules` nacheinander auszuführen. Dabei wird durch das Target `tim-timetable` auch das aperiodische EAN erzeugt, indem ein periodischer Fahrplan bestimmt wird, anhand dessen das EAN ausgerollt wird. Wird dies mit obigen Parameterwerten durchgeführt, können mehrere Ausführungen mit gleichen Eingabewerten zu unterschiedlichen periodischen Fahrplänen und somit zu unterschiedlichen aperiodischen EANs führen.

Die klassische Methode zur Minimierung der Fahrzeuganzahl (Minimal Decomposition Model 2) wurde mit der entwickelten Heuristik für TTVS1 verglichen, indem für bahn-01 zuerst mit `tim-timetable` ein aperiodisches EAN mit zugehörigem Fahrplan bestimmt wurde, als zweites mit `tim-vs-trips` und `vs-vehicle-schedules` der zugehörige Umlaufplan berechnet und mit `tim-vs-evaluate` ausgewertet wurde. Anschließend wurden mehrfach, ohne vorher das aperiodische EAN zu verändern, mit `tim-vs-trips` und `tim-vs` unter Verwendung unterschiedlicher Verfahren zur Bestimmung einer Startlösung weitere Paare von Fahrplan und Umlaufplan erzeugt und mit `tim-vs-evaluate` ausgewertet. Das Minimal Decomposition Model verwendet dabei stets die Kompatibilitätsrelation $\mathcal{C}_\beta$. Wird in der Heuristik die gleiche Kompatibilitätsrelation verwendet, so sind die Lösungen schlechter, als mit dem klassischen Verfahren, wie Abbildung 3.18 zeigt. Verwendet man hingegen die Kompatibilitätsrelation $\mathcal{C}_\alpha$, so sind die Lösungen deutlich besser. Dies ist in Abbildung 3.19 dargestellt. Die Verbesserung ergibt sich also hauptsächlich aus der Erweiterung der Kompatibilitätsrelation. Außerdem fällt auf, dass für die Kompatibilitätsrelation $\mathcal{C}_\beta$ die Ausführungen, deren Startlösungen mit (IS1) erzeugt wurden, zu Endlösungen mit geringerer Fahrzeugzahl führen, als die, deren Startlösungen mit (IS2) erzeugt wurden.

Das oben beschriebene Prozedere wurde auch zum Vergleich der beiden Methoden verwendet, bei denen neben der Gesamtfahrzeit die Umlaufkosten minimiert werden. Dazu wurde in der klassischen Vorgehensweise das Netzwerkflussmodell verwendet und die Heuristik für (TTVS2) verwendet. Da das Netzwerkflussmodell mit der Kompatibilitätsrelation $\mathcal{C}_\alpha$ arbeitet, wurde diese auch für (TTVS2) verwendet. Die Zielfunktionswerte der Ergebnisse sind in Abbildung 3.20 dargestellt.

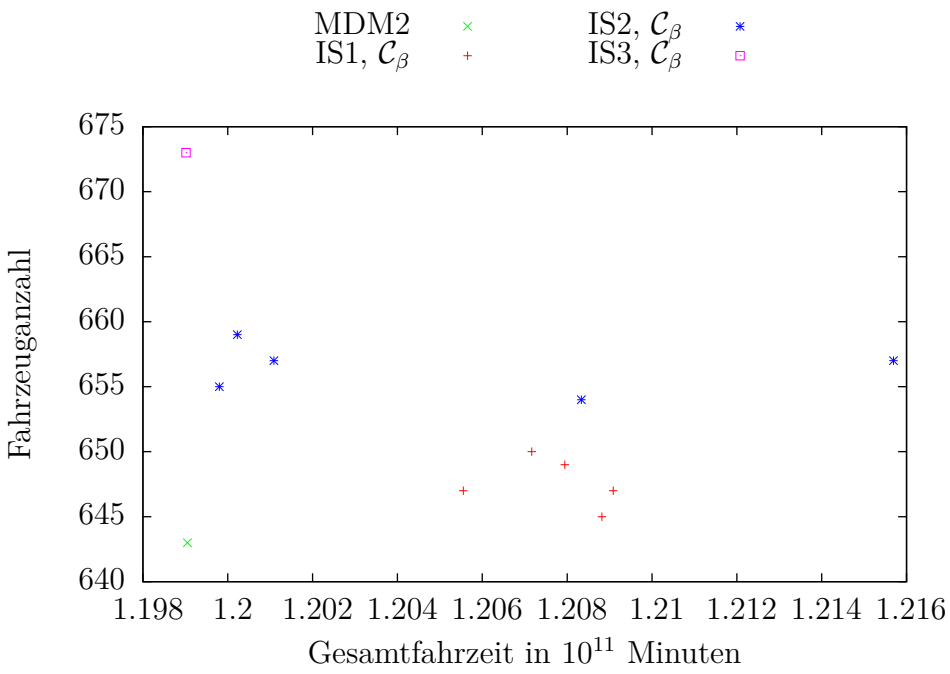

Abb. 3.18: Übersicht über mit unterschiedlichen Verfahren erzeugte Lösungen für (TTVS1) mit $\mathcal{C}_\beta$

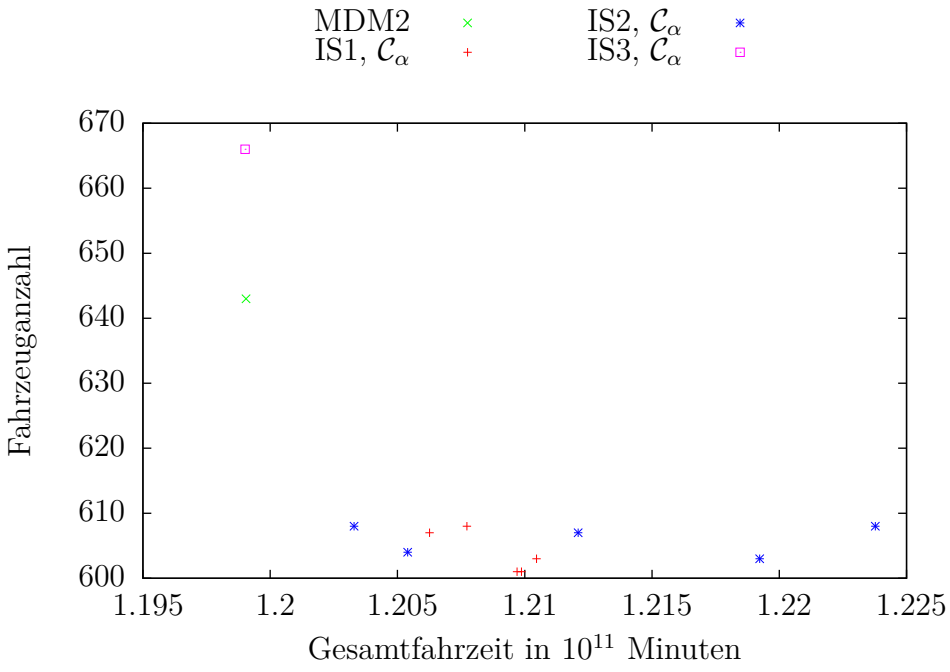

Abb. 3.19: Übersicht über mit unterschiedlichen Verfahren erzeugte Lösungen für (TTVS1) mit $\mathcal{C}_\alpha$

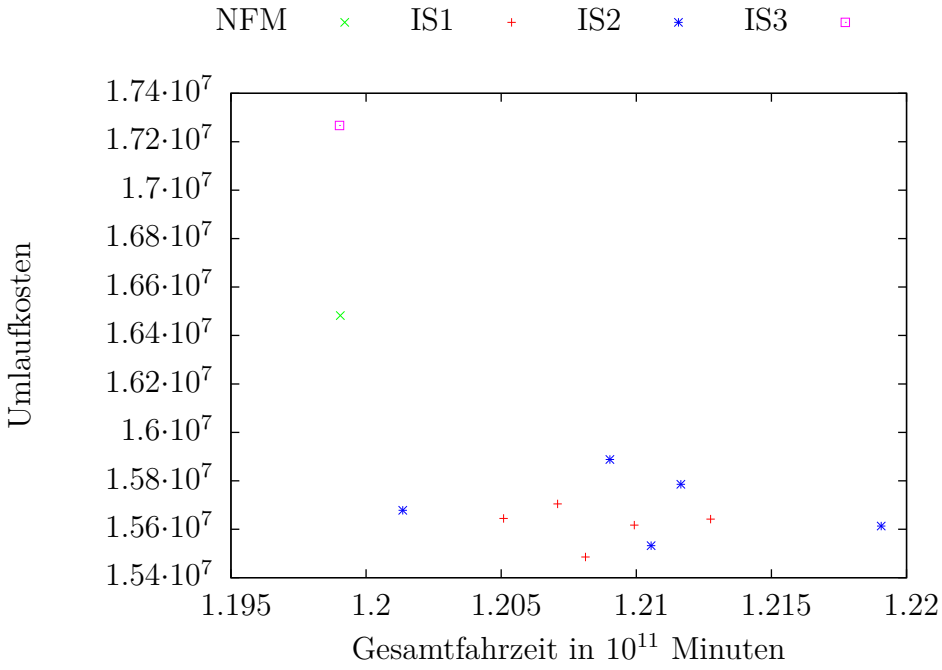

Abb. 3.20: Übersicht über mit unterschiedlichen Verfahren erzeugte Lösungen für (TTVS2) mit $\mathcal{C}_\alpha$

Es zeigt sich, dass die Heuristik für (TTVS2) eine deutliche Verringerung der Umlaufkosten bringt.

Zusammenfassung der Ergebnisse

Durch die numerischen Experimente wurden folgende Ergebnisse über das Verhalten der auf Algorithmus 5 beruhenden Heuristik gewonnen:

1. Die in einem Optimierungsschritt erzielte Verringerung der Gesamtfahrzeit sowie der Fahrzeuganzahl bzw. der Umlaufkosten nimmt im Verlauf des Algorithmus ab.

2. Die Gesamtfahrzeit verbessert sich nach dem ersten Optimierungsschritt prozentual viel stärker als die Fahrzeuganzahl bzw. die Umlaufkosten.

3. Es ist möglich, dass sich ein Zielfunktionswert verbessert, nachdem sich in einer Iteration beide Zielfunktionswerte nicht verändert haben.

4. Es ist kein direkter Zusammenhang zwischen den Zielfunktionswerten der Lösung $(\pi^{(1)}, z^{(1)})$ und der Endlösung erkennbar.

5. Die Gesamtfahrzeit der Endlösung ist bei (TTVS1) und (TTVS2) in etwa gleich.

6. Bei (TTVS1) ist der Anteil der Verbesserung des Umlaufplans im ersten Schritt an der Gesamtverbesserung höher als bei (TTVS1).
7. Die Endergebnisse von Ausführungen, bei denen (IS1) zur Bestimmung der Startlösung verwendet wurde, sind bei der Verwendung aller möglichen Fahrzeugübergänge in beiden Zielfunktionswerten in etwa genauso gut wie die von Ausführungen, bei denen (IS2) angewendet wurde.
8. Werden nur Fahrzeugübergänge ohne Leerfahrten zwischen verschiedenen Stationen zugelassen, so ergeben sich Lösungen mit höherer Fahrzeuganzahl bzw. höheren Umlaufkosten und ungefähr gleich hoher Gesamtreisezeit. Diese werden allerdings nach viel weniger Iterationen gefunden. In diesem Fall finden Durchläufe mit Startlösungen aus (IS1) Lösungen, welche mit weniger Fahrzeugen auskommen, als die mit (IS2) gefundenen.
9. Im Vergleich zu den bisher implementierten Verfahren können bei Minimierung der Fahrzeuganzahl Verbesserungen durch das Erlauben aller möglichen Übergänge und bei Minimierung der Umlaufkosten Verbesserungen durch die Anwendung der Heuristik erzielt werden.

4 Fazit

In dieser Arbeit wurden die Probleme, einen Fahrplan und einen Umlaufplan zu erstellen, vorgestellt und es wurde eine bikriterielle IP-Formulierung für deren gemeinsame Lösung aufgestellt. Dadurch werden Reisezeiten der Fahrgäste und Fahrzeuganzahl bzw. Kosten für das Verkehrsunternehmen als gleichwertige zu minimierende Zielfunktionen behandelt. Im Gegensatz zu bisherigen Ansätzen zur Integration von Fahrplanoptimierung und Umlaufplanung wurde der Schwerpunkt nicht auf die Modellierung möglichst vieler Details gelegt, sondern es wurde vom einfachst denkbaren Fall ausgegangen: Die Fahrplanoptimierung ist aperiodisch ohne Headway-Entscheidungen und in der Umlaufplanung wird von einem Fahrzeugtyp und einem Depot ausgegangen. Es wurde bewiesen, dass das integrierte Fahrplan-Umlaufplan-Problem selbst in diesem Fall, in dem beide Probleme für sich genommen polynomiell lösbar sind, und unter der weiteren Einschränkung, dass die Dauer aller Fahrten festgelegt ist, NP-schwer ist. Damit wurde gezeigt, dass dies auch in allen Szenarien, die sich auf den in dieser Arbeit betrachteten Fall zurückführen lassen (beliebig viele Depots, beliebig viele Fahrzeugtypen etc.), der Fall ist. Gilt $\mathbf{P} \neq \mathbf{NP}$, so ist es also nicht möglich, das integrierte Fahrplan-Umlaufplan-Problem in irgendeinem dieser Fälle in polynomieller Zeit exakt zu lösen.

Aus diesem Grund wurde eine Heuristik entwickelt, welche Fahrpläne und Umlaufpläne berechnet, die mit weniger Fahrzeugen bzw. Kosten auskommen, dafür aber höhere Reisezeiten der Kunden in Kauf nehmen. Die Heuristik nutzt aus, dass beide Teilprobleme in polynomieller Zeit lösbar sind und löst diese beginnend mit einem vorgegebenen zulässigen Fahrplan abwechselnd. Bei der Untersuchung der Heuristik wurde von diesem speziellen Problem abstrahiert und es wurden ganz allgemein bikriterielle ganzzahlige lineare Programme mit ähnlicher Struktur untersucht. Die Ergebnisse sowie die Heuristik können auf jedes dieser Programme übertragen werden. Ein Beispiel für ein anderes Problem aus der Verkehrsplanung, das sich als ganzzahliges Programm der untersuchten Form formulieren lässt, ist die Fahrplanoptimierung mit Headway-Entscheidungen, wobei als zweite Zielfunktion 0 verwendet wird. Auch das integrierte Fahrplan-Umlaufplan-Problem mit Headway-Entscheidungen kann als ganzzahliges Programm der beschriebenen Form formuliert werden.

Nachdem die Heuristik implementiert wurde, wurde sie an Probleminstanzen aus dem LinTim-Projekt der Universität Göttingen getestet und hat dort für die Minimierung der Umlaufkosten gute Ergebnisse geliefert. So konnte eine Verringerung der Umlaufkosten um über 4,8% bei einer Erhöhung der Reisezeiten um weniger als 0,2% erreicht werden.

Allerdings hat sich gezeigt, dass mit der Heuristik gefundene Fahrpläne häufig nicht so gute Umlaufpläne ermöglichen wie mit den bisherigen Methoden erstellte. Um die Vorteile der Heuristik und der bisher in LinTim implementierten unterschiedlichen Algorithmen zur Fahrplanoptimierung und Umlaufplanung zu verbinden, wäre ein möglicher nächster Schritt, die Heuristik so anzupassen, dass sie in jedem Schritt einen Fahrplan bzw. einen Umlaufplan im LinTim-Format erzeugt, so dass sie mit jeder erdenklichen Kombination der in LinTim vorhandenen und der neu implementierten Programmroutinen ausgeführt werden kann.

Eine mögliche Richtung zukünftiger Forschung ist, das beschriebene Modell zu erweitern und zu prüfen, für welche Fälle die implementierte Heuristik noch anwendbar ist bzw. angepasst werden kann. Mögliche Erweiterungen sind z. B. die Einbeziehung mehrerer Depots oder die Berücksichtigung zeitabhängiger Kosten für Fahrten und Übergänge. Das würde die Berücksichtigung der Variablen π bei der Berechnung der Umlaufkosten nötig machen.

Literaturverzeichnis

[AGKS06] Alfieri, Arianna ; Groot, Rutger ; Kroon, Leo ; Schrijver, Alexander: Efficient Circulation of Railway Rolling Stock. In: *TRANSPORTATION SCIENCE* 40 (2006), Nr. 3, 378–391.
http://homepages.cwi.nl/~lex/files/1526-5447-2006-40-03-0378.pdf

[Anh12] Anhalt, Jennifer: *Eine iterative Heuristik für aperiodische Fahrplangestaltung mit OD-Paaren*, Georg-August-Universität Göttingen, Diplomarbeit, 2012.
http://num.math.uni-goettingen.de/picap/pdf/E736.pdf

[BGAB83] Bodin, L. ; Golden, B. ; Assad, A. ; Ball, M.: Routing and scheduling of vehicles and crews: The state of the art. In: *Computers & Operations Research* 10 (1983), Nr. 2, 63 - 211.
http://dx.doi.org/10.1016/0305-0548(83)90030-8.

[BK09] Bunte, Stefan ; Kliewer, Natalia: An overview on vehicle scheduling models. In: *Public Transport* 1 (2009), Nr. 4, 299-317 – revised online-published version from 17 March 2010.
http://dx.doi.org/10.1007/s12469-010-0018-5.

[CLRS10] Cormen, T.H. ; Leiserson, C. E. ; Rivest, R. ; Stein, C.: *Algorithmen - Eine Einführung*. Oldenbourg Wissenschaftsverlag München, 2010.

[CM12] Cadarso, Luis ; Marín, Ángel: Integration of timetable planning and rolling stock in rapid transit networks. In: *Annals of Operations Research* 199 (2012), 113–135.
http://dx.doi.org/10.1007/s10479-011-0978-0.

[Coo71] Cook, Stephen A.: The complexity of theorem-proving procedures. In: *Proceedings of the third annual ACM symposium on Theory of computing*. New York, NY, USA : ACM, 1971 (STOC '71), 151–158.

[Dan02] Dantzig, George B.: Linear Programming. In: *Operations Research* 50 (2002), Nr. 1, 42–47.
http://ftp.cs.duke.edu/courses/spring07/cps296.2/papers/LinearProgramming_article.pdf.

[EMC+01] Ehrig, H. ; Mahr, B. ; Cornelius, F. ; Grobe-Rhode, M. ; Zeitz, P.: *Mathematisch-strukturelle Grundlagen der Informatik*. Springer-Verlag New York, 2001 (Springer-Lehrbuch).
http://books.google.de/books?id=lRxJ1PCHtZsC.

[GH10] Guihaire, Valérie ; Hao, Jin-Kao: Transit network timetabling and vehicle assignment for regulating authorities. In: *Computers & Industrial Engineering* 59 (2010), Nr. 1, 16–23.
http://dx.doi.org/10.1016/j.cie.2010.02.005.

[GJ79] Garey, Michael R. ; Johnson, David S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA : W. H. Freeman & Co., 1979.

[GSS12] Goerigk, M. ; Schachtebeck, M. ; Schöbel, A.: Dependencies between line planning, timetabling and delay management: Simulations with the LinTim toolbox. In: *Preprint-Serie des Instituts für Numerische und Angewandte Mathematik Göttingen* (2012).
http://num.math.uni-goettingen.de/preprints/files/2012-03.pdf

[IRRS11] Ibarra-Rojas, Omar J. ; Rios-Solis, Yasmin A.: Integrating synchronization bus timetabling and single-depot single-type vehicle scheduling. (2011).
http://orp3.uca.es/pdfs/ORP3_28_OIbarra.pdf

[JJ13] Jäger, Sven J. ; Jung, Tim A.: *Linienplanung plus Passagier-Routing*. 2013. – Präsentation zum Projekt des LinTim-Praktikums des Instituts für Numerische und Angewandte Mathematik der Georg-August-Universität Göttingen

[Kar72] Karp, Richard M.: Reducibility among Combinatorial Problems. Version: 1972. In: Miller, Raymond E. (Hrsg.) ; Thatcher, James W. (Hrsg.) ; Bohlinger, Jean D. (Hrsg.): *Complexity of Computer Computations*. Springer US, 1972 (The IBM Research Symposia Series).
http://dx.doi.org/10.1007/978-1-4684-2001-2_9.

[KKM94] Kohli, R. ; Krishnamurti, R. ; Mirchandani, P.: The Minimum Satisfiability Problem. In: *SIAM Journal on Discrete Mathematics* 7 (1994), Nr. 2, S. 275–283

[LM07] Liebchen, Christian ; Möhring, Rolf H.: The modeling power of the periodic event scheduling problem: railway timetables—and beyond. Version: 2007. In: *Algorithmic Methods for Railway Optimization*. Springer, 2007, 3–40
http://opus4.kobv.de/opus4-matheon/files/160/181.pdf.

[LP02] Liebchen, Christian ; Peeters, Leon: Some Practical Aspects of Periodic Timetabling. Version: 2002. In: Chamoni, Peter (Hrsg.) ; Leisten, Rainer (Hrsg.) ; Martin, Alexander (Hrsg.) ; Minnemann, Joachim (Hrsg.) ; Stadtler, Hartmut (Hrsg.): *Operations Research Proceedings 2001*. Springer Berlin

Heidelberg, 2002.
http://dx.doi.org/10.1007/978-3-642-50282-8_4.

[MS09] Michaelis, Mathias ; Schöbel, Anita: Integrating line planning, timetabling, and vehicle scheduling: a customer-oriented heuristic. In: *Public Transport* 1 (2009), Nr. 3, 211–232.
http://dx.doi.org/10.1007/s12469-009-0014-9.

[NSW11] Nickel, S. ; Stein, O. ; Waldmann, K. H.: *Operations Research*. Springer Berlin Heidelberg, 2011.

[Sah70] Saha, J. L.: An Algorithm for Bus Scheduling Problems. In: *Operational Research Quarterly (1970-1977)* 21 (1970), Nr. 4, pp. 463–474.
http://www.jstor.org/stable/3008423.

[Sch98] Schrijver, A.: *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998 (Wiley Series in Discrete Mathematics & Optimization).
http://books.google.de/books?id=zEzW5mhppB8C.

[Sch04] Schöbel, Anita: *Optimization Models in Public Transportation*. 2004. – Lecture notes

[Sch07] Scholz, Daniel: *Multikriterielle Optimierung*. Version: 2007. – Vorlesungsmitschrift im Sommersemester 2006 zur Vorlesung von S. Schwarze, Universität Göttingen
http://www.mehr-davon.de/content/multi.pdf.

[Sch09] Schachtebeck, Michael: *Delay Management in Public Transportation: Capacities, Robustness, and Integration*, Georg-August-Universität Göttingen, Diss., 2009.
http://hdl.handle.net/11858/00-1735-0000-0006-B3CE-4

[Sch11] Schmidt, Marie E.: *Integrating Routing Decisions in Network Problems*, Georg-August-Universität Göttingen, Diss., 2011

[Sch12] Schöbel, Anita: *Einführung in die Optimierung – Sommersemester 2012*. 2012. – Lecture notes

[Sch13] Schöbel, Anita: *Diskrete Optimierung in der Verkehrsplanung*. 2013. – Vorlesung im Wintersemester 2012/2013

[Sie11] Siebert, Michael: *Integration of Routing and Timetabling in Public Transportation*, Georg-August-Universität Göttingen, Diplomarbeit, 2011.
http://num.math.uni-goettingen.de/picap/pdf/E672.pdf

[SS09] Schachtebeck, Michael ; Schöbel, Anita: LinTim — A Toolbox for the Experimental Evaluation of the Interaction of Different Planning Stages in Public Transportation / ARRIVAL Project. Forschungsbericht – Version: 2009.
http://arrival.cti.gr/uploads/Documents.0206/ARRIVAL-TR-0206.pdf.

[SU89] Serafini, P. ; Ukovich, W.: A mathematical model for periodic scheduling problems. In: *SIAM Journal on Discrete Mathematics* 2 (1989), Nr. 4, S. 550–581

[UBA12] *Daten zum Verkehr*. Version: 2012. – Umweltbundesamt Deutschland `http://www.umweltdaten.de/publikationen/fpdf-l/4364.pdf`.

[Uff10] Uffmann, Anke: *Das Kanalmodell zur Effiziensteigerung in der Fahrzeugumlaufplanung*, Georg-August-Universität Göttingen, Diplomarbeit, 2010

[vvv08] van den Heuvel, A. P. R. ; van den Akker, J. M. ; van Kooten Niekerk, M. E.: Integrating timetabling and vehicle scheduling in public bus transportation. (2008). `http://www.cs.uu.nl/research/techreps/repo/CS-2008/2008-003.pdf`

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit mit dem Titel

„Integration der Fahrplanoptimierung und Umlaufplanung“

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Göttingen, den 30. August 2013