Verallgemeinerte Schnittheuristiken in der periodischen Fahrplangestaltung

Diplomarbeit

vorgelegt von

Marc Goerigk

aus

Berlin

angefertigt
im Institut für Numerische und Angewandte Mathematik
der Georg-August-Universität zu Göttingen
2009

Erklärung

Ich versichere hiermit, dass ich die vorliegende Diplomarbeit mit dem Titel „Verallgemeinerte Schnittheuristiken in der periodischen Fahrplangestaltung“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Göttingen, den 17. August 2009 Marc Goerigk

Inhaltsverzeichnis

Teil I

Theoretischer Teil

„Es kann also niemand sich für praktisch bewandert in einer Wissenschaft ausgeben und doch die Theorie verachten, ohne sich bloß zu geben, daß er in seinem Fache ein Ignorant sei: indem er glaubt, durch Herumtappen in Versuchen und Erfahrungen, ohne sich gewisse Prinzipien (die eigentlich das ausmachen, was man Theorie nennt) zu sammeln, und ohne sich ein Ganzes (welches, wenn dabei methodisch verfahren wird, System heißt) über sein Geschäft gedacht zu haben, weiter kommen zu können, als ihn die Theorie zu bringen vermag.“

-Immanuel Kant,
Über den Gemeinspruch: Das mag in der Theorie richtig sein, taugt aber nicht für die Praxis

Kapitel 1

Einführung

1.1 Verkehrsoptimierung

Die in dieser Arbeit im Mittelpunkt stehende *Fahrplanoptimierung* ist Teil eines größeren Problemzyklus, der *Verkehrsoptimierung*. Es wird einleitend der Zusammenhang zu anderen Problemen dargestellt und die wirtschaftliche Bedeutung anhand von Beispielen illustiert.

Der öffentliche Verkehr ist für Deutschland von zentraler ökonomischer und sozialer Bedeutung. Insbesondere im öffentlichen Personennahverkehr sind täglich millionen Menschen auf Unternehmen angewiesen, die den gewaltigen Anforderungen gewachsen sein müssen. In Großstädten führt ein unbefriedigend bedientes öffentliches Verkehrsnetz zu Staus und hohen Umweltbelastungen, im ländlichen Gegenden zu Immobilität und Bevölkerungsschwund. Um von staatlicher Seite diesen Problemen beizukommen, werden hohe Subventionen ausgezahlt, die wiederum den Steuerzahler treffen.
Damit ein Verkehrsunternehmen erfolgreich geführt werden kann, müssen zuvor zahlreiche Probleme gelöst werden. Ein Bahnunternehmen zum Beispiel muss sich unter anderem fragen: Auf welchen Linien sollen Züge verkehren? Was für Züge werden dafür dafür ausgewählt? Wie kann der erwartete Kundenbedarf befriedigt werden? Wo sollten zusätzliche Schienen oder Bahnhöfe gebaut werden? Wo muss andererseits ein Bahnhof geschlossen werden? Reicht das vorhandene Personal aus? Wann fährt jeder einzelne Zug an und ab? Wie können möglichst viele Güterzüge fahren? Wie werden Gleise in Bahnhöfen kollisionsfrei belegt?
Wie diese Fragen im Detail beantwortet werden, ist entscheidend für die Qualität des Angebots und die Höhe der anfallenden Kosten und damit ausschlaggebend über Erfolg oder Misserfolg eines Unternehmens. Die über die letzten Jahrzente exponentiell gewachsenen Rechenkapazitäten moderner Computeranlagen öffnen einen mathematischen Zugang zu diesen Fragen, der erst seit kurzer Zeit Verbesserungen zu den bisher gewählten, durch Erfahrung gewachsenen Methoden erlaubt.
Die mathematische Formulierung eines Sachverhaltes erfordert immer zunächst ein passendes Modell. Dabei muss abgewogen werden zwischen *Realitätsnähe*, das heißt in diesem Fall die Übertragbarkeit der Rechenergebnisse auf den konkreten Unternehmensbedarf, und *Komplexität*, also die mathematisch-theoretische wie die praktisch-rechnerische Zugänglichkeit des Problems.

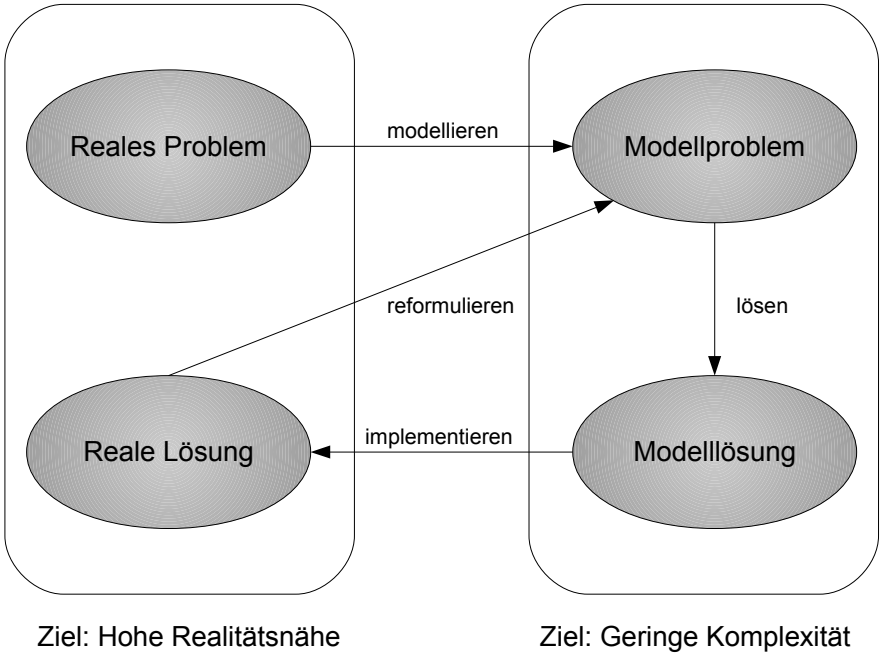


Abbildung 1.1: Ziele der Modellbildung

Eingangs wurde eine Auswahl der zahlreichen Probleme, die bei der Verkehrsoptimierung anfallen, bereits erwähnt. Um diese Fragen beantworten zu können, wählt die gängige Literatur als ersten Modellbildungsschritt eine Zerlegung in Einzelprobleme, die jeweils aus den bereits erarbeiteten Antworten die Eingangsdaten für darauf aufbauende Probleme finden sollen. Dies ist bereits eine Vereinfachung, da die so entstehenden Probleme offenbar nicht unabhängig voneinander sind und die simultane Bearbeitung eine bessere Lösung geben könnte. Am Beispiel eines Bahnunternehmens würden folgende Teilprobleme in Betracht gezogen werden müssen:

1. *Netzwerk-Design.* Wie muss die Infrastruktur verändert werden, um bei möglichst geringen Kosten genügend Kapazitäten gemäß meiner Bedarfsprognose gewähren zu können?

2. *Linienplanung.* Wie können auf der Infrastruktur derart Zuglinien angeboten werden, dass Kunden genügend Transportmöglichkeiten geboten werden bei gleichzeitig möglichst geringen Kosten?

3. *Fahrplangestaltung.* Wann müssen diese Züge an- und abfahren, so dass Kunden genügend Zeit zum Umsteigen haben, gleichzeitig aber nicht unnötig lange warten müssen?

4. *Umlaufplanung.* Wie kann dieser Fahrplan erfüllt und können gleichzeitig möglichst wenige Züge dafür verwendet werden?

5. *Personalzuweisung.* Wie kann gemäß den teilweise sehr komplexen Arbeitsgesetzen dem Personal ein konkreter Dienstplan gegeben werden, der

gleichzeitig für das Personal wie auch für die Finanzierung des Unternehmens nicht unnötig belastend ist?

6. *Tarifplanung.* Wie können Fahrkartenpreise für den Kunden gerecht gestaltet werden und gleichzeitig die Kosten des Unternehmens decken?

7. *Dispositionsplanung.* Was kann getan werden, um unvermeidlich entstehende Verspätungen möglichst auswirkungslos auf den restlichen Betrieb zu halten?

Abbildung 1.2 stellt diese Teilprobleme schematisch dar. Der lineare Verlauf deutet an, dass als Eingangsgrößen jeweils die Ausgangsgrößen der Vorgänger verwendet werden können; gleichzeitig wird es aber im Planungsprozess nötig sein, bereits bearbeitete Probleme noch einmal aufzugreifen, wenn die darauf aufbauenden Lösungen späterer Probleme nicht befriedigend sind. Daher sind die Teilprobleme als mit doppelseitigen Pfeilen verbunden dargestellt. Ferner unterscheiden sich die Probleme nach dem bei der Lösung zur Verfügung stehenden Zeitfenster. Ist die Standortplanung ein langfristiger Prozess, so muss die Dispositionsplanung auf operationeller Ebene agieren können. Dies ist wiederum ausschlaggebend für die Modellierung und Lösung des Teilproblems.
Die Fahrplangestaltung nimmt dabei eine herausragende Rolle ein, da sich einerseits ein großes Potential ergibt, unnötige Kosten zu reduzieren und gleichzeitig die Kundenzufriedenheit zu erhöhen, andererseits Änderungen im Vergleich zum Bau eines Bahnhofes oder auch zur Anschaffung eines neuen Zuges einfach umzusetzen sind. So kann die Forschung an der mathematischen Fahrplangestaltung zwei junge Erfolge vorweisen:

- *Der neue Niederländische Fahrplan.* (Siehe (KHA$^+$09)) Im Dezember 2006 konnte die größte Niederländische Bahngesellschaft, die *Nederlandse Spoorwegen*, den bereits 36 Jahre alten Fahrplan durch einen vollständig neuen ersetzen, der durch eine mathematische Herangehensweise gewonnen wurde. Die Pünktlichkeit der Züge, das heißt der Anteil der Züge, die höchstens drei Minuten verspätet ankamen, konnte somit von 84,8% in den Jahren 2005 und 2006 auf 87% im Jahr 2007 erhöht werden. Aufgrund von Abkommen mit Verbraucherorganisationen erlaubte diese hohe Pünktlichkeit eine Tariferhöhung oberhalb der Inflation, was dem Unternehmen zusätzliche 20 Millionen Euro Profit ergab. Insgesamt erwarten die Autoren, die für ihre Arbeit mit dem rennomierten Franz Edelman Award für Erfolge im Bereich des Operations Research ausgezeichnet wurden, einen jährlichen Gewinn von 70 Millionen Euro für Nederlandse Spoorwegen. Ihre Arbeit hebt zudem deutlich den genannten Aspekt der Fahrplangestaltung hervor, dass die verschiedenen Planungsschritte sich einander beeinflussen: Es wurden gleich 10 verschiedene Fahrpläne berechnet, die dann in Hinsicht auf ihre Brauchbarkeit im weiteren Planungsverlauf bewertet wurden (anders als im „klassischen“ Modell, das in dieser Arbeit untersucht wird). Die zwei besten Exemplare wurden dann zu dem endgültigen Resultat zusammengefügt.

- *Die Berliner U-Bahn.* (Siehe (Lie08) und (Hus05)) Laut den Autoren von (Lie08) trat am 12. Dezemer 2005 der erste mathematische gestaltete Fahrplan im öffentlichen Verkehr für die tägliche Anwendung im durch die BVG betriebenen Berliner U-Bahnnetz in Kraft. Aufgrund der Planungserfahrungen des Unternehmens wurden die verschiedenen Linien gemäß Prioritäten

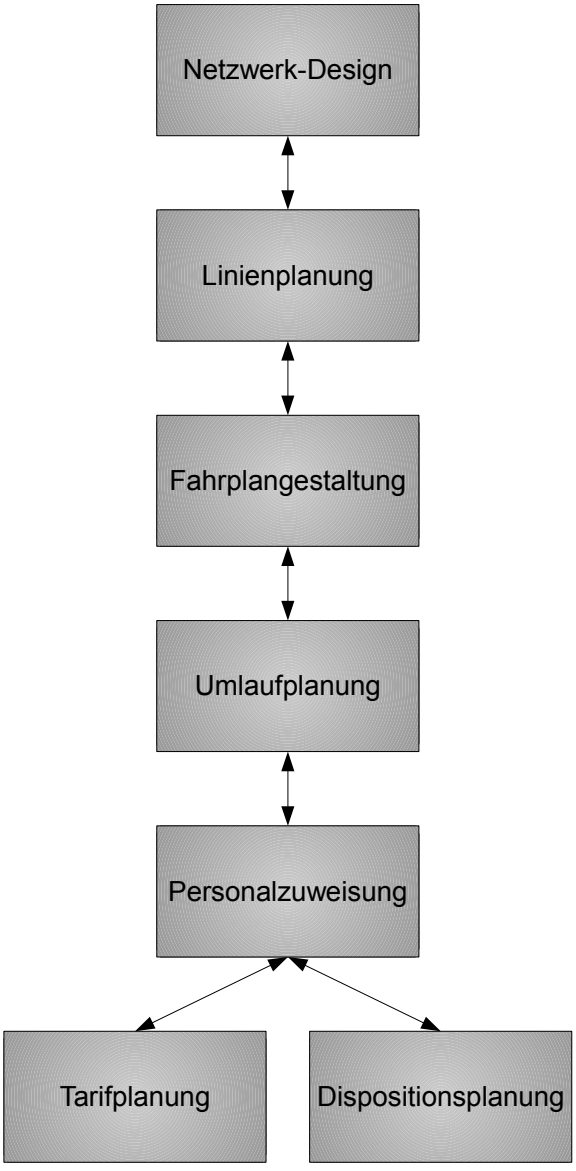


Abbildung 1.2: Teilprobleme der Verkehrsoptimierung

gestuft, um das Problem einerseits zu erleichtern und andererseits Kernstrecken zu bevorzugen. Dabei wurden die drei Ziele auf Unternehmensseite, nämlich einen Zug einzusparen, die maximale Wartezeit von Zügen in Bahnhöfen von 3,5 min auf 2,5 min zu reduzieren und die Umsteigezeiten von Fahrten nicht erster Priorität zu reduzieren (aufgrund der bisherigen Herangehensweise waren diejenigen erster Priorität bereits sehr gut) voll erfüllt.

Diese Beispiele verdeutlichen, dass die mathematische Fahrplangestaltung bereits so weit entwickelt ist, dass in der Praxis einsetzbare Lösungen erreicht werden können. In dieser Arbeit werden zunächst die Grundlagen erläutert, die für das darauf folgende mathematische Modell notwendig sind. Im Mittelpunkt steht ein Lösungsverfahren, das von Nachtigall und Opitz in (NO08) entwickelt wurde. Es werden eine Verallgemeinerung und mehrere Verbesserungen vorgeschlagen und in einem zweiten, experimentellen Teil an Beispielen der Größe realer Instanzen untersucht. Die dabei erzielten Ergebnisse können das etablierte Verfahren teilweise in Geschwindigkeit und Qualität übertreffen.

1.2 Graphentheorie

Die grundlegenden Konzepte der Verkehrsoptimierung, auf die im Verlaufe dieser Arbeit immer wieder zurückgegriffen wird, stammen aus der Graphentheorie. Dieser Abschnitt gibt eine kurze Übersicht zu den wichtigsten Definitionen.

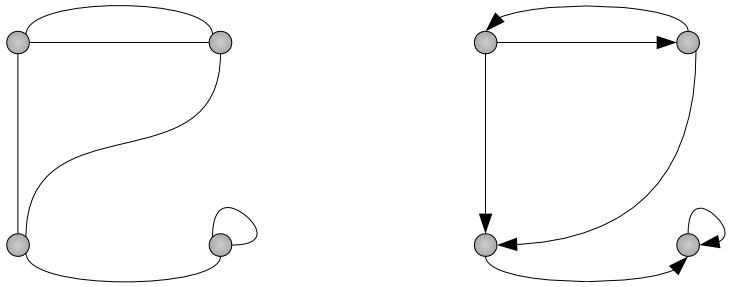
Die Ursprünge der Graphentheorie reichen bis in das 18. Jahrhundert zurück, als Leonhard Euler das Königsberger Brückenproblem formulierte, doch sind gerade im Laufe des 20. Jahrhunderts durch die zunehmende Verwendung in der Informatik und Komplexitätstheorie Graphen zu einem zentralen mathematischen Baustein avanciert. Die intuitive Verwendung eines Graphen als Modell eines geographischen Zusammenhangs, wie urpsrünglich von Euler eingeführt, ist gerade in der Verkehrsoptimierung von Relevanz. Da sich der Großteil dieser Arbeit daher auf Graphen stützt, werden diese grundlegenden Konzepte kurz eingeführt.

Definition 1.1. Graph (Che76)
Ein (ungerichteter) *Graph* $G(V, E)$, auch kurz G geschrieben, wenn der Zusammenhang offenbar ist, besteht aus einer Menge V, den *Knoten*, und einer Menge von ungeordneten Knotenpaaren $\{i, j\}$, $i, j \in V$, den *Kanten*.
Sind die Kanten nicht Mengen, sondern Tupel, so spricht man von einem gerichteten Graphen. Der Einheitlichkeit halber werden Kanten daher im Folgenden immer in der Form (i, j) notiert, unabhängig davon, ob der Graph gerichtet ist oder nicht.

Definition 1.2. Teilgraph (Che76)
Ein *Teilgraph* $G'(V', E')$ eines Graphen $G(V, E)$ ist ein Graph mit $V' \subset V$ und $E' \subset E$. Gilt sogar $V' = V$, so ist der Teilgraph G' ein *spannender Teilgraph*.

Definition 1.3. Der durch eine Knotenmenge induzierte Teilgraph
Sei eine Teilmenge $V' \subset V$ der Knoten eines Graphen $G(V, E)$ gegeben. Dann heißt

$$E(V') := \{(i, j) \in E : i, j \in V'\}$$



(a) Ein ungerichteter Graph. (b) Ein gerichteter Graph.

Abbildung 1.3: Beispiele für Graphen.

die durch V' induzierte Kantenmenge und $G'(V', E(V'))$ der induzierte Teilgraph.

Definition 1.4. Kantenfolge (Che76)
Eine *Kantenfolge* der Länge $k-1$, $k \geq 2$, in einem Graphen G ist eine endliche Folge von Kanten der Form

$$(i_1, i_2), (i_2, i_3), \ldots, (i_{k-1}, i_k).$$

Ist $i_k = i_1$, so heißt die Kantenfolge *geschlossen*, ansonsten *offen*.

Definition 1.5. Pfad, Kreis (Che76)
Sind die Kanten und die Knoten einer offenen Kantenfolge

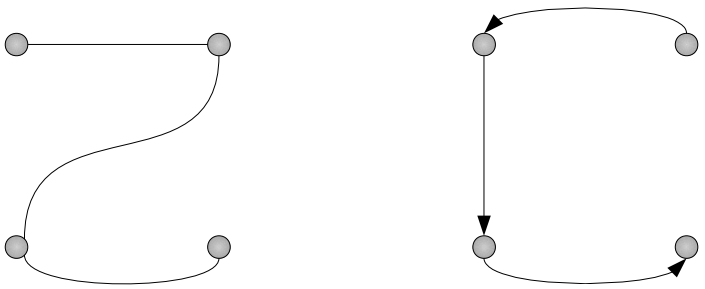
$$p = (i_1, i_2), (i_2, i_3), \ldots, (i_{k-1}, i_k)$$

paarweise verschieden, so heißt p auch *Pfad* der Länge $k-1$.
Ist eine solche Kantenfolge dagegen geschlossen, so wird sie *Kreis* oder *Zykel* genannt.

Definition 1.6. Zusammenhängender Graph (Che76)
Ein Graph heißt *zusammenhängend*, wenn jedes Knotenpaar durch einen Pfad verbunden ist.

Definition 1.7. Baum (Che76)
Ein zusammenhängender spannender Teilgraph, der keinen Kreis enthält, wird *Baum* genannt.
Im Falle eines gerichteten Graphen gilt ein Teilgraph genau dann als Baum, wenn er im entsprechenden ungerichteten Graphen ein Baum ist.

Satz 1.8. *Zusammenhangskomponenten (NW88)*
Es gibt genau eine Knotenpartition eines ungerichteten Graphen $G(V, E)$ in Teilmengen $V_1, \ldots, V_p$ mit der Eigenschaft, dass zwei Knoten genau dann in derselben Teilmenge liegen, wenn sie durch einen Pfad verbunden sind. Der Teilgraph $G_k(V_k, E(V_k))$ heißt Zusammenhangskomponente *von G.*



(a) Ein Baum im ungerichteten Graphen aus 1.3.

(b) Ein Baum im gerichteten Graphen aus 1.3.

Abbildung 1.4: Beispiele für Bäume.

Satz 1.9. *Eigenschaften von Bäumen (NW88)*
Sei $G(V, E)$ ein ungerichteter Graph. Dann ist äquivalent:

1. *G ist ein Baum.*
2. *Es gibt genau einen Pfad zwischen allen Knotenpaaren in G.*
3. *G ist zusammenhängend und besitzt $|V| - 1$ Kanten.*
4. *G ist zusammenhängend und kreisfrei.*

1.3 Lineare Optimierung

Die meisten der in dieser Arbeit beschriebenen Probleme sind *lineare Programme*. Hier wird dargestellt, worum es sich dabei handelt und welche allgemeinen Eigenschaften sich herleiten lassen.

Das Ziel der linearen Optimierung ist die Suche nach dem Minimum einer linearen Funktion, deren Definitionsbereich durch endlich viele lineare Bedinungen beschränkt ist. Ein solches Problem ist also von der Form

$$\min f(x) = c^T x \tag{1.10}$$
$$\text{so dass} \quad Ax \leq p \tag{1.11}$$
$$Bx = q \tag{1.12}$$
$$Cx \geq r \tag{1.13}$$
$$x, c \in \mathbb{R}^n \tag{1.14}$$
$$p \in \mathbb{R}^k, q \in \mathbb{R}^l, r \in \mathbb{R}^m \tag{1.15}$$
$$A \in \mathbb{R}^{k \times n} \tag{1.16}$$
$$B \in \mathbb{R}^{l \times n} \tag{1.17}$$
$$C \in \mathbb{R}^{m \times n}. \tag{1.18}$$

Durch elementare Umformungen lässt sich ein lineares Programm immer in die sogenannte $\leq$-Form bringen.

Definition 1.19. $\leq$-Form eines linearen Programms (BS80)
Ein lineares Programm (LP) der Form

$$\min f(x) = c^T x \tag{1.20}$$
$$\text{so dass} \quad Ax \leq b \tag{1.21}$$
$$x \in \mathbb{R}^n_+ \tag{1.22}$$

heißt *in $\leq$-Form*.

Wird die Gleichung $Ax = b$ anstelle von (1.21) verwendet, so befindet sich das lineare Programm in der *Standardform*. Da beide Formulierungen ineinander überführbar sind, reicht es im Folgenden aus, lineare Programme in $\leq$-Form zu untersuchen.
Mathematisch besonders leicht zu fassen ist die Menge konvexer Probleme, zu denen auch die linearen Programme gehören.

Definition 1.23. Konvexität (Sch07)
Eine *Menge* $\mathcal{M} \subset \mathbb{R}^n$ heißt *konvex*, wenn für alle $x, y \in \mathcal{M}$ und alle $\lambda \in [0, 1]$ gilt:

$$\lambda x + (1 - \lambda)y \in \mathcal{M}.$$

Eine *Funktion* f auf einer konvexen Menge $\mathcal{M}$ heißt *konvex*, wenn für alle $x, y \in \mathcal{M}$ und alle $\lambda \in [0, 1]$ gilt:

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y).$$

Satz 1.24. *Minima in konvexen Programmen (Sch07)*
Sei $X \subset \mathbb{R}^n$ konvex, $X \neq \emptyset$ und $f : X \to \mathbb{R}$ konvex. Dann gilt:

1. *Jedes lokales Minimum des Programms $\{\min f(x) : x \in X\}$ ist zugleich ein globales Minimum.*
2. *Die Menge der Minima*

   $$M^* = \{x \in X : f(x) \leq f(y)\ \forall y \in X\}$$

   ist konvex.

In diesem Sinne darf im Zusammenhang mit linearen Programmen von *dem* Optimum gesprochen werden, denn jedes lokale Optimum ist zugleich ein globales Optimum.
Häufig ist es von Nutzen, anstelle des ursprünglichen LPs ein dazu *duales* Problem zu untersuchen.

Definition 1.25. Dualität (NW88)
Sei ein lineares Programm, das *primale* (P) genannt, der Form

$$\min f(x) = c^T x \tag{1.26}$$
$$\text{so dass} \quad Ax \leq b \tag{1.27}$$
$$x \in \mathbb{R}^n_+ \tag{1.28}$$

gegeben. Dann ist das dazu *duale* Problem (D) gegeben durch

$$\max g(x) = -u^T b \tag{1.29}$$
$$\text{so dass} \quad u^T A \geq -c \tag{1.30}$$
$$u \in \mathbb{R}^m_+. \tag{1.31}$$

Welches Problem als primal, welches als dual angesehen wird, spielt tatsächlich keine Rolle, wie der folgende Satz besagt:

Satz 1.32. *(NW88)*
Das Duale eines dualen Problems ist das primale Problem.

Duale Probleme sind gerade deshalb für die Untersuchung von linearen Programmen von Interesse, da sie dieselben Optimalwerte besitzen und sich gegenseitig Schranken liefern. Das wird mit den Sätzen der *schwachen* und der *starken* Dualität festgehalten.

Satz 1.33. *Schwache Dualität (NW88)*
Sei x^* optimale Lösung des primalen Problems, u^* optimale Lösung des Dualen. Dann gilt für alle zu (P) zulässigen Lösungen x und für alle zu (D) zulässigen Lösungen u

$$u^T b \leq u^{*T} b \leq cx^* \leq cx.$$

Satz 1.34. *Starke Dualität (NW88)*
Gilt für die optimalen Lösungen x^* und u^*, dass $c^T x$ oder $u^T b$ endlich ist, so sind beide Zielfunktionswerte endlich und gleich.

Korollar 1.35. *(NW88)*
Ist das Problem (P) unbeschränkt, so ist (D) unzulässig.

Korollar 1.36. *(NW88)*
Es gilt immer nur genau eine der vier Möglichkeiten:

1. *Die optimalen Zielfunktionswerte von (P) und (D) sind endlich und gleich.*
2. *Der optimale Zielfunktionswert von (P) ist $-\infty$ und (D) ist unzulässig.*
3. *Der optimale Zielfunktionswert von (D) ist ∞ und (P) ist unzulässig.*
4. *(P) und (D) sind unzulässig.*

1.4 Komplexitätstheorie

Im Laufe dieser Arbeit ist es nötig, Algorithmen bezüglich ihrer Laufzeit und Probleme bezüglich ihrer Schwierigkeit zu bewerten. Hier wird kurz dargestellt, wie dies mit Hilfe der Landau-Symbole und Komplexitätsklassen möglich ist.

1.4.1 Landau-Symbole

Angenommen, ein Algorithmus ist gegeben und es soll nun untersucht werden, wie schnell er laufen wird. Offenbar ist das entscheidend abhängig davon, wie er implementiert wurde und auf welcher Hardware er läuft. Um dennoch eine Vergleichbarkeit zu ermöglichen, wird von der konkreten Ausführung abstrahiert und es werden statt dessen *Schritte* gezählt, die der Algorithmus vollführen muss. Dafür muss überhaupt festgelegt werden, *welche* Schritte als die Laufzeit beeinflussend gezählt werden und wie sehr sie ins Gewicht fallen sollen.

Definition 1.37. Kostenfunktion
Eine *Kostenfunktion* ist eine Abbildung

$$\{+, -, \cdot, /, :=, \leq\} \to \mathbb{N},$$

die einer Rechenoperation, einer Variablenzuweisung oder einem Vergleich Kosten zuordnet.

Mithilfe einer gegebenen Kostenfunktion kann dann ein Algorithmus darauf untersucht werden, welche Kosten er verursacht.

Beispiel 1.38.
Sei das Kostenmaß T gegeben, das elementare Rechenoperationen mit Kosten 1 belegt und Variablenzuweisungen und Vergleiche mit Kosten 0. Betrachte den Algorithmus 1.38 zur Suche nach Teilern.

Algorithmus 1 Beispielsalgorithmus zum Finden aller Teiler einer Zahl

Eingabe: Eine natürliche Zahl n.
Ausgabe: Die Menge L aller Teiler dieser Zahl.

```
1: Setze L := ∅.
2: for (i = 1 to n) do
3:    if (n/i ∈ ℕ) then
4:       L := L ∪ {i}.
5:    end if
6: end for
```

Offenbar ist hier die Laufzeit davon abhängig, wie groß die Eingabezahl ist. In diesem Falle schreiben wir $T(n)$, um die summierten Kosten des Algorithmus in Abhängigkeit von der Eingangsgröße darzustellen. Da Zuweisungen in diesem Beispiel keine Kosten verursachen, verursacht einzig die Division in Zeile 3 eine einzelne Kosteneinheit pro Durchlauf. Da alle Zahlen von 1 bis n durchlaufen werden, entstehen also Gesamtkosten von $T(n) = n$.

Häufig ist allerdings weniger die konkrete Anzahl an Schritten von Interesse, da diese wiederum stark von der Implementation abhängen und sich durch verschiedene Größenordnungen gegenseitig „überdecken“. Die Tabelle 1.1 vergleicht verschiedene Laufzeiten miteinander in Abhängigkeit von der Eingangsgröße n.

n	n^2	$n^2 + n$	$2n^2$	n^3
10	100	110	200	1000
20	400	420	800	8000
30	900	930	1800	27000
40	1600	1640	3200	64000
50	2500	2550	5000	125000

Tabelle 1.1: Verschiedene Laufzeiten

Offenbar befinden sich die Spalten mit quadratischen Termen trotz der Unterschiede in derselben *Größenordnung*, der kubische Teil dagegen *entfernt* sich immer weiter. Dies wird durch die das asymptotische Verhalten beschreibenden *Landau*-Symbole ausgedrückt.

Definition 1.39. Landau-Symbole (BS80)
Seien zwei Funktionen $f, g : \mathbb{R} \to \mathbb{R}_+$ gegeben.

1. Gilt
$$\lim_{x \to x_0} \frac{f(x)}{g(x)} = 0 \text{ bzw. } \lim_{x \to \pm\infty} \frac{f(x)}{g(x)} = 0,$$
so schreibt man $f(x) = o(g(x))$ für $x \to x_0$ bzw. $x \to \pm\infty$.

2. Gilt
$$\left|\frac{f(x)}{g(x)}\right| < M \text{ für } x \to x_0 \text{ bzw. } x \to \pm\infty$$
für ein $M \in \mathbb{R}$, so schreibt man $f(x) = \mathcal{O}(g(x))$ für $x \to x_0$ bzw. $x \to \pm\infty$.

Die Symbole o und $\mathcal{O}$ werden als *Landau-Symbole* bezeichnet.

Als „Faustregel“ gilt, dass es durch die Landau-Symbole ausreicht, auf den am schnellsten wachsenden Teil einer Funktion zu achten. Da es für unsere Zwecke nur von Interesse ist, das Verhalten der Variablen im Unendlichen zu untersuchen, wird im folgenden stets auf die Schreibweise „für $x \to \infty$“ verzichtet.

Beispiel 1.40.
$$5n^2 + n + \log(n) = \mathcal{O}(n^2),$$
denn
$$\left|\frac{5n^2 + n + \log(n)}{n^2}\right| = \left|5 + \frac{1}{n} + \frac{\log(n)}{n^2}\right| < 5 + \frac{2}{n} < 6 \text{ für } x \to \infty.$$

1.4.2 Komplexitätsklassen

Wie erwähnt, lassen sich die Landau-Symbole nur auf eine konkrete Funktion anwenden, die zum Beispiel die Kosten eines Algorithmus angibt, können aber nicht vom Algorithmus abstrahierte Aussagen über die Schwierigkeit eines Problems geben. Ist *ein* Algorithmus zur Lösung eines Problems gefunden, der in Laufzeit $\mathcal{O}(2^n)$ läuft, heißt das nicht, dass es keinen besseren Algorithmus geben kann. Da jeder Algorithmus polynomieller Laufzeit asymptotisch kleiner ist als jeder Algorithmus exponentieller Laufzeit, liegt es nahe, zu unterscheiden, ob es zu einem Problem einen polynomiellen Algorithmus gibt oder nicht. Das ist die wesentliche Idee der *Komplexitätsklassen*. Dazu wird eine *Komplexitätsrelation* $\propto$ eingeführt, die die Rückführbarkeit aufeinander abbildet:

Definition 1.41. Polynomielle Reduzierbarkeit (Sch09)
Seien zwei Probleme A und B gegeben. Man sagt, A sei auf B *polynomiell reduzierbar* und schreibt $A \propto B$, wenn es einen in polynomieller Zeit laufenden Algorithmus gibt, der eine Probleminstanz von A in eine Instanz von B überführt.

Definition 1.42. Zulässigkeitsproblem (NW88)
Sei eine Menge D endlicher binärer Zahlenfolgen, die *Menge der Instanzen*, und eine Menge $F \subset D$, die *Menge der zulässigen Instanzen*, gegeben. Das *Zulässigkeitsproblem* $X(D, F)$ lautet dann, zu einer gegebenen Instanz $d \in D$ zu entscheiden, ob $d \in F$ oder nicht.

Definition 1.43. Komplexitätsklassen (NW88),(Sch09)

1. Gibt es zu einem Problem P einen Algorithmus mit Laufzeit $f(n) = \mathcal{O}(n^k)$ für ein $k \in \mathbb{N}$, so heißt P in polynomieller Zeit lösbar. Die Menge $\mathcal{P}$ bezeichne alle in polynomieller Zeit lösbaren Probleme.

2. Es sei $\mathcal{NP}$ die Menge aller Zulässigkeitsprobleme, für die die Zulässigkeit einer zulässigen Lösung in polynomieller Zeit durch einen nichtdeterministischen Algorithmus festgestellt werden kann. (Es wird keine Aussage über den Fall, dass eine Lösung unzulässig ist, getroffen.)

3. Sei $\mathcal{NPC} := \{P \in \mathcal{NP} : P' \propto P \ \forall P' \in \mathcal{NP}\}$. Probleme dieser Klasse werden *NP-schwer* genannt.

Beispiel 1.44.

1. $\mathcal{P}$.(Com09) Der Test, ob eine Zahl prim ist, und das Finden eines maximalen Matchings in einem allgemeinen Graphen sind in polynomieller Zeit lösbar.
2. $\mathcal{NP}$. Das *Graphenisomorphieproblem* lautet, zu zwei gegebenen Graphen eine Bijektion zwischen den Knoten zu finden, so dass Nachbarschaften erhalten bleiben. Das Problem ist in NP, doch ist es zu diesem Zeitpunkt nicht bekannt, ob es auch NP-schwer ist. Es lässt sich praktisch allerdings sehr schnell lösen dank rekursiver Algorithmen wie NAUTY (McK81).
3. $\mathcal{NPC}$.(Com09) Typische Vertreter für NP-schwere Probleme sind das Problem des Handlungsreisenden, die Dreifärbbarkeit eines Graphen oder die Suche einer maximalen Clique. Auch das in dieser Arbeit behandelte periodische Fahrplanproblem gehört in diese Klasse.

Bemerkung 1.45. (NW88)
Ein noch immer offenes Problem ist, ob $\mathcal{P} = \mathcal{NP}$ gilt, wobei die gängige Vermutung lautet, dass dies nicht der Fall sei. Würde es wider Erwarten doch gelten, wäre insbesondere das Problem der periodischen Fahrplangestaltung sehr viel effektiver lösbar, als alle bisher gefunden Algorithmen es ermöglichen.

Das Schema 1.5 nach (NW88) veranschaulicht den Zusammenhang der Mengen $\mathcal{NP}$, $\mathcal{NPC}$ und $\mathcal{P}$ für den Fall, dass $\mathcal{P} \neq \mathcal{NP}$. Es kann gezeigt werden, dass dann $\mathcal{P} \cap \mathcal{NPC} = \emptyset$ und $\mathcal{P} \cup \mathcal{NPC} \neq \mathcal{NP}$ gilt.

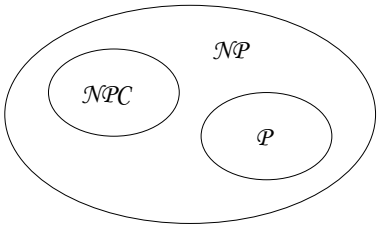


Abbildung 1.5: Zusammenhang zwischen $\mathcal{P}$, $\mathcal{NP}$ und $\mathcal{NPC}$, falls $\mathcal{NP} \neq \mathcal{P}$.

Kapitel 2

Aperiodische Fahrplangestaltung

2.1 Problemstellung

Das Ziel der aperiodischen Fahrplangestaltung ist die Erzeugung eines bezüglich einer gegebenen Zielfunktion optimalen Fahrplanes für ein gegebenes Linienkonzept auf einem sich nicht wiederholenden Zeitabschnitt. Gemäß Abschnitt 1.1 baut sie also auf einem Linienkonzept auf. Es wird daher zunächst die Linienplanung genauer untersucht (2.1.1) und anschließend, wie daraus ein *Ereignis-Aktivitäts-Netzwerk* gewonnen wird (2.1.2). Schließlich kann das Problem der aperiodischen Fahrplangestaltung formuliert werden (2.1.3).

2.1.1 Linienplanung

Zunächst werden die Eingangsdaten der Linienplanung formuliert. Angenommen, dass das Netzwerk-Design bereits abgeschlossen wurde und somit eine feste Menge Haltestellen oder Bahnhöfe existieren, die durch Strecken fester Länge (also zum Beispiel Bahnschienen oder kürzeste Wege im Straßenverkehr einer Stadt) verbunden sind. Der dazugehörige Graph wird *Personentransport-Netzwerk* (PTN) genannt:

Definition 2.1. PTN (Sch09)
Ein *Personentransport-Netzwerk* ist ein ungerichteter Graph $G = (V, E)$, dessen Knoten V als Haltestellen und Kanten E als direkte Verbindungen aufgefasst werden.

Die Fahrt eines Fahrzeuges kann somit als *Pfad* im PTN aufgefasst werden. Da es in der Praxis üblich ist, mehrfach in einem Zeitraum dieselbe Fahrt anzubieten, wird auf Seite des Modells die *Linie* eingeführt:

Definition 2.2. Linie (Sch09)
Eine *Linie* ist ein Pfad in einem PTN.
Sei eine Periode $T \in \mathbb{N}$ gegeben. Dann stellt die *Frequenz* $f_l \in \mathbb{N}$ einer Linie l dar, wie häufig sie innerhalb dieser Zeiteinheit bedient wird.
Ein *Linienkonzept* $(\mathcal{L}, f)$ ist eine Menge Linien mit den dazugehörigen Frequenzen f_l.

Bemerkung 2.3.
Die in 1.5 gegebene Definition eines Pfades schließt Linien aus, die eine Haltestelle mehrfach bedienen. Sollte dieser Fall im Modell erwünscht sein, müsste die Linie entsprechend anders definiert werden.

Für gewöhnlich verfügt der Anbieter dieser Fahrten bereits über einen historisch gewachsenen und den Kunden bekannten Vorrat an möglichen Linien, den *Linienpool*.

Definition 2.4. Linienpool (Sch09)
Ein *Linienpool* $\mathcal{L}^0$ ist eine Menge von Linien zu einem PTN.

Eine Menge von Linien lässt sich übersichtlich in Form einer Matrix schreiben, indem jede Spalte eine Linie darstellt, jede Zeile eine Kante:

Definition 2.5. Kanten-Linien-Matrix
Sei eine Menge Linien $\mathcal{L}$ auf einem PTN $G(V, E)$ gegeben. Dann bezeichnet man die Matrix $A = (a_{i,j}) \in \mathbb{B}^{|E|\times|L|}$ mit

$$a_{el} = \begin{cases} 1, & \text{wenn } e \in l, \\ 0 & \text{sonst} \end{cases}$$

als *Kanten-Linien-Matrix*.

Einzelne Strecken können unter Umständen nicht beliebig häufig befahrbar sein; Züge zum Beispiel müssen Sicherheitsabstände einhalten, wodurch ein Schienenabschnitt nur eine bestimmte Kapazität für das Linienkonzept bereitstellen kann. Andererseits gehen wir davon aus, dass der Bedarf der Kunden bereits aus Erfahrung und Umfragen bekannt ist. Dieser muss gedeckt werden, damit ein Linienkonzept als zulässig gilt. Die Grundlage für die Bedarfsbestimmung bildet die *origin-destination matrix* (Abfahrts-Ziel-Matrix), auch kurz *OD-Matrix* gennant:

Definition 2.6. OD-Matrix (Sch09)
Sei ein PTN $G(V, E)$ gegeben. Eine Matrix $OD = (od_{i,j}) \in \mathbb{N}_+^{|V|\times|V|}$ heißt dann *OD-Matrix*.

Sie ist so zu verstehen, dass der Eintrag $od_{i,j}$ die Anzahl der Kunden darstellt, die von Haltestelle i zu Haltestelle j fahren möchten. Mittels kürzeste-Wege-Verfahren lässt sich dieser Bedarf annäherungsweise auf die einzelnen Kanten verteilen.
Das Problem, ein zulässiges Linienkonzept zu finden, lässt sich nun formulieren:

(LP0) Finden eines zulässigen Linienkonzeptes (Sch09).
Seien ein PTN G, ein Linienpool $\mathcal{L}^0$, sowie untere und obere Schranken $f_e^{\min} \leq f_e^{\max}$ für jede Kante $e \in E$ gegeben. Es bezeichne A die zum Pool gehörige Kanten-Linien-Matrix. Finde ein Linienkonzept $(\mathcal{L}, f)$ mit $\mathcal{L} \subset \mathcal{L}^0$ und

$$f^{\min} \leq Af \leq f^{\max}.$$

Beispiel 2.7. Sei ein PTN $G(V, E)$ gegeben wie durch Abbildung 2.1 dargestellt. Die Ziffern an den Kanten bezeichnen die jeweiligen Entfernungen. Angenommen,

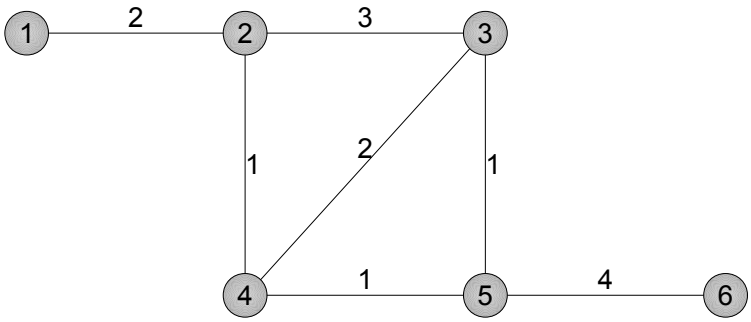


Abbildung 2.1: Ein Beispiels-PTN

durch Kundenumfragen wird nun folgende Bedarfsmatrix ermittelt:

$$OD = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & 50 & 100 & 50 & 0 & 50 \\ 100 & 0 & 0 & 0 & 100 & 40 \\ 0 & 70 & 0 & 0 & 30 & 80 \\ 20 & 0 & 50 & 0 & 0 & 100 \\ 40 & 50 & 10 & 30 & 0 & 40 \\ 0 & 100 & 0 & 150 & 0 & 0 \end{pmatrix} \end{array},$$

wobei eine Einheit ein ganzes Fahrzeug darstellen soll.
Ferner haben sich folgende Linien bereits im Einsatz bewährt und sollen wieder eingesetzt werden, wenn der entsprechende Bedarf besteht:

$$\begin{aligned} \mathcal{L}^0 = \{&(1,2,3),\\ &(2,4,5),\\ &(4,3,5,6),\\ &(2,3,4,5)\}. \end{aligned}$$

Durch Verteilung der OD-Matrix auf die einzelnen Kanten mittels kürzester Wege ermittelt man einen Bedarfsvektor

$$b = \begin{pmatrix} 410 \\ 170 \\ 450 \\ 50 \\ 120 \\ 660 \\ 560 \end{pmatrix},$$

wobei die Kanten in der Reihenfolge $(1,2),(2,3),(2,4),(3,4),(3,5),(4,5),(5,6)$ aufgelistet sind. Angenommen, ein Fahrzeug ist in der Lage, 100 Fahrgäste zu transportieren. Dann lässt sich die Anzahl der Kunden in benötigte Fahrzeuge umrechnen und der die untere Schranke für das Linienkonzept repräsentierende

Vektor ist gewonnen:

$$f^{\min} = \begin{pmatrix} 5 \\ 2 \\ 5 \\ 1 \\ 2 \\ 7 \\ 6 \end{pmatrix}.$$

Falls keine oberen Kapazitäten bekannt sind, lautet das Programm: Finde $f \in \mathbb{N}^4$ mit

$$f^{\min} \leq \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} f.$$

Es sei an dieser Stelle noch angemerkt, dass (LP0) *NP-schwer* ist.

Satz 2.8. *Komplexität der Suche eines zulässigen Linienkonzeptes (Sch09) (LP0) ist ein NP-schweres Problem.*

Eine natürliche Erweiterung dieses Problems besteht nun darin, nicht nur eine beliebige zulässige Lösung zu suchen, sondern eine *kostenoptimale*. Dazu gehen wir davon aus, dass die Bedienung jeder Linie Kosten verursacht, die zum Beispiel die Längen der Strecken oder Schwierigkeiten bei der Befahrung widerspiegeln.

(LP1) Finden eines kostenminimalen Linienkonzeptes (Sch09).
Seien ein PTN G, ein Linienpool $\mathcal{L}^0$, sowie untere und obere Schranken $f_e^{\min} \leq f_e^{\max}$ für jede Kante $e \in E$ und *Kosten* $c_l \in \mathbb{R}$ für jede Linie $l \in \mathcal{L}^0$ gegeben. Es bezeichne A die zum Pool gehörige Kanten-Linien-Matrix. Finde ein Linienkonzept $(\mathcal{L}, f)$ mit $\mathcal{L} \subset \mathcal{L}^0$ und

$$f^{\min} \leq Af \leq f^{\max},$$

das die Gesamtkosten

$$\sum_{l \in \mathcal{L}} c_l f_l$$

minimiert.

Dieses Problem ist selbstverständlich wiederum NP-schwer, lässt sich aber erfahrungsgemäß mit gängigen Solvern leicht lösen, wie im experimentellen Teil gezeigt werden wird. Hier seien noch zwei Greedy-Heuristiken aus (Sch09) dargestellt:

Der einfachste Ansatz, dargestellt in Algorithmus 2, besteht darin, in jedem Schritt diejenige Linie zu wählen, die am günstigsten ist und den Bedarf einer Kante deckt. So werden nach und nach immer teurere Kanten gewählt, bis schließlich jeder Bedarf gedeckt wurde oder aber festgestellt wird, dass die Probleminstanz unzulässig ist.

Algorithmus 3 dagegen wählt auch in jedem Schritt die gemäß einem Kriterium beste Linie aus, mit dem Unterschied, dass nun beachtet wird, wie gut die Kosten in Relation zu der Anzahl der überdeckten Kanten sind.

Algorithmus 2 Greedy-Heuristik für (LP1) ohne obere Schranken nach (Sch09)

Eingabe: Ein PTN $G(V, E)$, ein Linienpool $\mathcal{L}^0$, Kosten c_l und untere Schranken $f_l^{\min}$ für alle $l \in \mathcal{L}^0$.
Ausgabe: Ein zulässiges Linienkonzept, falls eines existiert.
1: Setze $\mathcal{L} = \emptyset$, $f = 0$.
2: Sortiere $\mathcal{L}^0$ nach den gegebenen Kosten.
3: **while** $(\exists e : f_e^{\min} \neq 0)$ **do**
4: Wähle die erste Linie l mit $f_e^{\min} \neq 0$ für ein $e \in l$. Existiert keine solche Linie, STOP: Es exisiert kein zulässiges Linienkonzept.
5: Setze $\mathcal{L} := \mathcal{L} \cup \{l\}$, $f_l := \max_{e \in l} f_e^{\min}$, $f_e := 0 \ \forall e \in l$.
6: **end while**

Algorithmus 3 Verbesserte Greedy-Heuristik für (LP1) ohne obere Schranken nach (Sch09)

Eingabe: Ein PTN $G(V, E)$, ein Linienpool $\mathcal{L}^0$, Kosten c_l und untere Schranken $f_l^{\min}$ für alle $l \in \mathcal{L}^0$.
Ausgabe: Ein zulässiges Linienkonzept, falls eines existiert.
1: Setze $\mathcal{L} = \emptyset$, $f = 0$.
2: **while** $(\exists e : f_e^{\min} \neq 0)$ **do**
3: Wähle $l \in \mathcal{L}^0$ mit $e \in l$, so dass

$$g(l) = \frac{c_l}{|\{e \in l : f_e^{\min} > 0\}|}$$

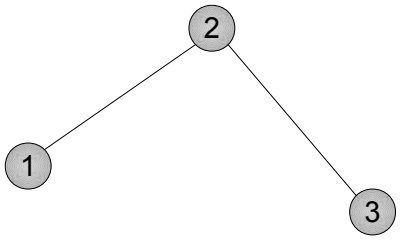
maximal ist. Wenn es kein solches l gibt, STOP: Es existiert kein zulässiges Linienkonzept.
4: Setze $\mathcal{L} := \mathcal{L} \cup \{l\}$, $f_l := f_l + \min_{e \in l : f_e^{\min} > 0} f_e^{\min}$, $f_e^{\min} := \max\{f_e^{\min} - f_l, 0\} \ \forall e \in l$.
5: **end while**

2.1.2 Ereignis-Aktivitäts-Netzwerke

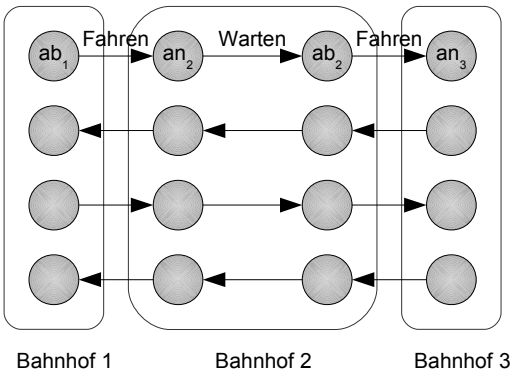
Im vorherigen Abschnitt wurde beschrieben, wie aus einem PTN und einem Linienpool ein Linienkonzept gewonnen wird. Nun ist also bereits bekannt, wieviele Fahrten innerhalb eines Zeitraumes entlang welchen Pfaden angeboten werden müssen. Zu welchen konkreten Zeitpunkten dies geschehen soll, ist Aufgabe der Fahrplangestaltung. Dazu wird das PTN erweitert zu einem sogenannten Ereignis-Aktivitäts-Netzwerk (*event-activity-network*, kurz EAN), das den zeitlichen Zusammenhang anstelle des geographischen betrachtet.
EANs beziehen ihren Namen daher, dass Knoten für *Ereignisse* und Kanten für *Aktivitäten* stehen; ein Knoten also einen Zeit*punkt* bedeutet, eine Kante eine Zeit*dauer*. Die Orientierung einer Kante impliziert damit eine Reihenfolge, in der zwei Ereignisse statt finden sollen. Um von einem PTN aus ein EAN zu erzeugen, muss daher neben dem Linienkonzept als zusätzliches Datum gegeben sein, wie lange jede Fahrt zwischen zwei Haltestellen dauert und wie lange dort jeweils die Verweilzeit ist.
Formal lässt sich zunächst definieren:

Definition 2.9. EAN
Ein *Ereignis-Aktivitäts-Netzwerk* ist ein gerichteter Graph $G(\mathcal{E}, \mathcal{A})$ mit *Kantengewichten* $\omega_{ij} \in \mathbb{N}$ und unteren und oberen Schranken $l_{ij}, u_{ij} \in \mathbb{N}$ für jede Kante $(i, j) \in \mathcal{A}$.

Auf welche Weise genau das PTN mit dem gewonnen Linienkonzept in ein EAN umgewandelt werden soll, ist letztlich Teil des Modells und nicht a priori zwingend. Betrachten wir eine einzelne Linie l mit Frequenz 2:



Da diese Linie zwei Mal im gegebenen Zeitabschnitt bedient werden soll, werden je zwei *Fahrten*, also Verkettungen von Fahrt- mit Warteaktivitäten, in beide Richtungen erzeugt:



Anschließend können *Umlaufkanten* eingefügt werden, die die Fahrt des Fahrzeugs vom Ende einer Linie zum Beginn der nächsten simulieren sollen. Je mehr solcher Kanten eingefügt werden, desto weniger Fahrzeuge sind nötig, um den später berechneten Fahrplan zu bedienen. Zu viele der Kanten können allerdings zu einer Unzulässigkeit des Problems führen.
Das Hauptaugenmerk der klassischen Fahrplangestaltung liegt dagegen auf möglichst kurzen Fahrzeiten der Passagiere. Da nicht alle Verkehrsteilnehmer für ihre geplante Fahrt eine passende Linie finden werden, die sie direkt bedient, müssen *Umsteigeaktivitäten* mit berücksichtigt werden. Befinden sich zwei Fahrzeuge verschiedener Linien am selben Bahnhof und gibt es umsteigewillige Passagiere, so werden Kanten eingefügt, die die zeitliche Dauer des Umsteigens simulieren, wie in Abbildung 2.2 dargestellt.

Ein solcherart erzeugtes EAN ignoriert alle möglicherweise vorhandenen Kapazitäts- und Sicherheitsbeschränkungen und wäre in der Praxis noch nicht einsetzbar. Ein Bahnbetreiber zum Beispiel muss darauf achten, dass Züge auf ihren Fahrten einen Sicherheitsabstand einhalten und nur dann in einen Bahnhof einfahren, wenn auch ein freies Gleis zur Verfügung steht. Für einen Busbetreiber dagegen kann die Modellierung von Sicherheitsabständen vernachlässigt werden,

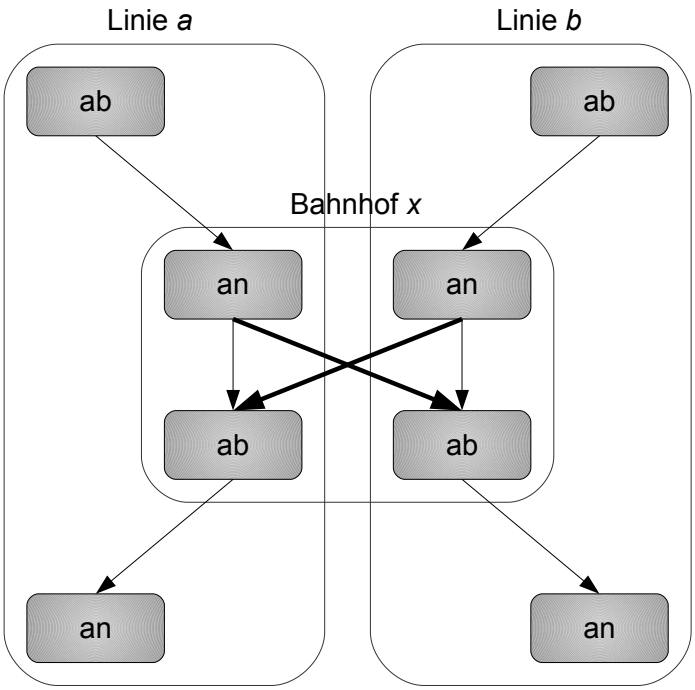


Abbildung 2.2: Einfügen von Umsteigekanten. Die neu hinzugefügten Kanten sind dick dargestellt.

doch müsste auch hier darauf geachtet werden, dass an einer Haltestelle nicht mehr Busse gleichzeitig halten, als es die Größe der Haltestelle erlaubt.
Ist die zeitliche Reihenfolge zweier Ereignisse vorgegeben, lassen sich Sicherheitsabstände innerhalb dieses Modells leicht durch *Headway-Aktivitäten* darstellen, indem an ein Ereignis i, das mindestens x Zeiteinheiten vor einem Ereignis j stattfinden muss, eine Kante (i, j) mit unterer Schranke x angefügt wird. Da im aperiodischen Fall nur ein fixer Zeitabschnitt geplant wird, ist eine Trennung in beide Zeitrichtungen hier nicht möglich, wohl aber im periodischen Fall, wie später offenbar sein wird. Kapazitätsbeschränkungen innerhalb von Haltestellen lassen sich leider nicht so leicht in das Modell integrieren und werden daher für gewöhnlich noch von Hand nach Erzeugung des Fahrplans geplant.

Insgesamt erhalten wir im auf diese Weise gewonnen EAN also folgende Ereignisse und Aktivitäten:

- Ereignisse
    1. Ankunftsereignisse
    2. Abfahrtsereignisse
- Aktivitäten
    1. Fahraktivitäten
    2. Warteaktivitäten
    3. Umlaufaktivitäten

4. Umsteigeaktivitäten
5. Headway-Aktivitäten

Abschließend müssen nun die Kantengewichte modelliert werden. Der gewöhnliche Ansatz lautet, dass die Gewichte die Anzahl der Passagiere repräsentieren sollen, die eine Fahrt verwenden. Wie ein Fahrgast allerdings im Einzelfall zu fahren plant, hängt wiederum entscheidend vom Fahrplan ab, der noch gar nicht bekannt ist - ein Kreisschluss, aus dem nur durch eine Approximation entkommen werden kann. Es müssen Fahrzeiten *geschätzt werden* (zum Beispiel durch Mittelwerte von unteren und oberen Schranken) und als Grundlage für die Fahrgastverteilung verwendet werden.
Dies ist eine der Modellierungsstellen, an denen die in der Einführung erwähnte Vereinfachung des Problems durch Unterteilung in Teilprobleme klar zutage tritt. Die einzelnen Probleme beeinflussen sich gegenseitig, werden aber für eine zugängliche Behandlung sequentiell gelöst. Wird auf Grundlage des durch das approximierte EAN gewonnenen Fahrplans ein *neues* EAN erstellt, so wird dies das Fahrgastverhalten *besser* widerspiegeln.

2.1.3 Vom EAN zum Fahrplan

Indem aus einem Linienkonzept ein EAN gewonnen wurde, sind nun die nötigen Mittel beisammen, um das Problem der *aperiodischen Fahrplangestaltung* zu formulieren.

(AF) Finden eines kostenminimalen Fahrplans, erste Formulierung (Sch09). Sei ein EAN $G(\mathcal{E}, \mathcal{A})$ mit unteren Schranken $l_{ij} \in \mathbb{N}$, oberen Schranken $u_{ij} \in \mathbb{N}$ und Gewichten $\omega_{ij} \in \mathbb{N}$ für jede Kante $(i,j) \in \mathcal{A}$ gegeben. Gesucht sind im *aperiodischen Fahrplanproblem* (AF) kostenminimale Knotenpotentiale π_i für jeden Knoten $i \in \mathcal{E}$, so dass die Fahrzeitschranken eingehalten werden, also

$$\min \quad \sum_{(i,j)\in\mathcal{A}} w_{ij}(\pi_j - \pi_i) \tag{2.10}$$

$$\text{so dass} \quad l_{ij} \leq \pi_j - \pi_i \leq u_{ij} \qquad \forall (i,j) \in \mathcal{A} \tag{2.11}$$

$$\pi_i \in \mathbb{R}_+ \qquad \forall i \in \mathcal{E}. \tag{2.12}$$

Unter einem *Fahrplan* wird also eine Abbildung $\mathcal{E} \to \mathbb{R}_+$ verstanden. Auffälligerweise werden die Knotenpotentiale nur als Differenzen auf den Kanten verwendet. Das motiviert eine neue Formulierung, in der statt der Knotenpotentiale eine Variable für jede Kante verwendet wird, die *Spannung*. Dann muss allerding darauf geachtet werden, dass die Spannung entlang jeden Kreises sich auf 0 summiert, denn nur so ist gesichert, dass der relative zeitliche Abstand zweier Ereignisse unabhängig vom Pfad ist, über den sie sich verbinden.

Definition 2.13. Kreismatrix (Che76)
Die *Kreis-Kanten-Inzidenzmatrix* (oder kurz *Kreismatrix*) $B \in \{0, 1, -1\}^{p \times |E|}$ eines gerichteten Graphen $G(V, E)$ mit p (ungerichteten) Kreisen ist gegeben durch

$$b_{ce} = \begin{cases} 1, & \text{wenn } e \text{ im Kreis } c \text{ mit gleicher Orientierung enthalten ist,} \\ -1, & \text{wenn } e \text{ im Kreis } c \text{ mit umgekehrter Orientierung enthalten ist,} \\ 0 & \text{sonst.} \end{cases}$$

Durch die Kreismatrix lässt sich nun (AF) reformulieren:

(AF) Finden eines kostenminimalen Fahrplans, zweite Formulierung (Sch09).
Sei ein EAN $G(\mathcal{E}, \mathcal{A})$ mit unteren Schranken $l_{ij} \in \mathbb{N}$, oberen Schranken $u_{ij} \in \mathbb{N}$ und Gewichten $\omega_{ij} \in \mathbb{N}$ für jede Kante $(i,j) \in \mathcal{A}$ gegeben. Sei B die zu G gehörende Kreismatrix. Gesucht sind nun Spannungen x_e für jede Kante $e \in \mathcal{A}$, so dass gilt

$$\begin{aligned}
&\min && \sum_{(i,j)\in\mathcal{A}} w_{ij}x_{ij} && && (2.14)\\
&\text{so dass} && l_{ij} \le x_{ij} \le u_{ij} && \forall (i,j) \in \mathcal{A} && (2.15)\\
& && Bx = 0 && && (2.16)\\
& && x_{ij} \in \mathbb{R}_+ && \forall (i,j) \in \mathcal{E}. && (2.17)
\end{aligned}$$

Satz 2.18. *(Sch09) Sei ein gerichteter, zusammenhängender Graph $G(V,E)$ mit Daten $\pi_i \in \mathbb{R}\ \forall i \in V$ gegeben. Dann gilt*

$$\pi_j - \pi_i = x_{ij} \Rightarrow Bx = 0,$$

wobei B die zu G gehörende Kreismatrix bezeichne.

Beweis. (Sch09)
Sei $x_{ij} = \pi_j - \pi_i$ und $p = (i_1, i_2), \ldots, (i_k, i_1)$ ein beliebiger Kreis in G. Dann ist

$$\begin{aligned}
0 &= \pi_{i_2} - \pi_{i_1} + \pi_{i_3} - \pi_{i_2} + \cdots + \pi_{i_1} - \pi_{i_k}\\
&= x_{i_1 i_2} + \cdots + x_{i_k i_1}.
\end{aligned}$$

Da dies für jeden beliebigen Kreis gilt, ist $Bx = 0$. □

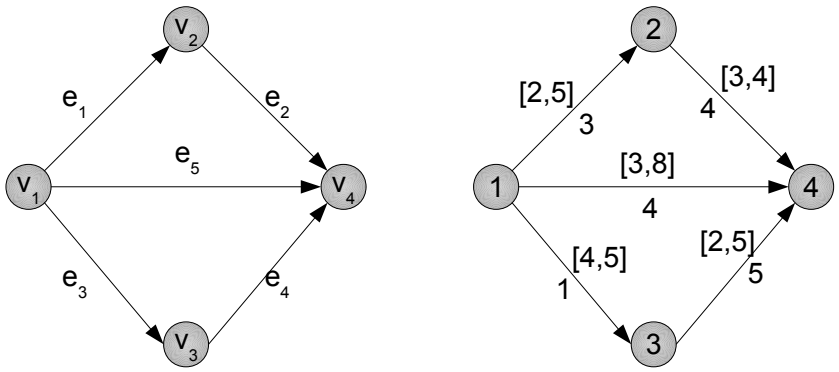
Satz 2.19. *Sei ein gerichteter, zusammenhängender Graph $G(V,E)$ mit Daten $x_{ij} \in \mathbb{R}\ \forall (i,j) \in E$ gegeben. Dann gilt*

$$\pi_j - \pi_i = x_{ij} \Leftarrow Bx = 0,$$

wobei B die zu G gehörende Kreismatrix bezeichne.

Beweis.
Es gelte $Bx = 0$. Wähle einen beliebigen Knoten i und setze $\pi_i = 0$. Defininiere nun rekursiv $\pi_j := x_{ij} + \pi_i$ für alle Knoten j, die durch Kante x_{ij} mit einem Knoten i verbunden sind, für den bereits π definiert wurde. Es lassen sich so alle Knoten erreichen, da der Graph zusammenhängend ist. Ferner ist wie gefordert $x_{ij} = \pi_j - \pi_i$. Es bleibt die Wohldefiniertheit zu zeigen. Angenommen, es gibt zwei Pfade $p_1 = (i, v_1), \ldots, (v_k, j)$ und $p_2 = (i, v'_1), \ldots, (v'_l, j)$ von i nach j mit $x_{iv_1} + \ldots x_{v_k j} \neq x_{iv'_1} + \ldots x_{v'_l j}$. Die Aneinanderkettung von p_1 und p_2 ergibt dann einen Kreis, für den nach Voraussetzung gilt, dass die Summe aller x-Werte der Kanten 0 ergibt. Wir erhalten einen Widerspruch. □

Beispiel 2.20.
Sei ein EAN wie in Abbildung 2.3 (a) mit unteren und oberen Schranken und Gewichten wie in (b) gegeben.



(a) EAN. (b) Gewichte und Schranken.

Abbildung 2.3: Beispielsinstanz für (AF).

(AF) in der ersten Formulierung lautet dann

$$\begin{aligned}
\min \quad & 3(\pi_2 - \pi_1) + 4(\pi_4 - \pi_2) + 1(\pi_3 - \pi_1) + 5(\pi_4 - \pi_2) + 4(\pi_4 - \pi_1) \\
\text{so dass} \quad & 2 \leq \pi_2 - \pi_1 \leq 5 \\
& 3 \leq \pi_4 - \pi_2 \leq 4 \\
& 4 \leq \pi_3 - \pi_1 \leq 5 \\
& 2 \leq \pi_4 - \pi_3 \leq 5 \\
& 3 \leq \pi_4 - \pi_1 \leq 8 \\
& \pi_i \in \mathbb{R}_+ \text{ für } i = 1, 2, 3, 4.
\end{aligned}$$

Werden Spannungen anstelle von Potentialen verwendet, erhält man statt dessen die Formulierung

$$\begin{aligned}
\min \quad & 3x_1 + 4x_2 + x_3 + 5x_4 + 4x_5 \\
\text{so dass} \quad & 3 \leq x_1 \leq 5 \\
& 3 \leq x_2 \leq 4 \\
& 4 \leq x_3 \leq 5 \\
& 2 \leq x_4 \leq 5 \\
& 3 \leq x_5 \leq 8 \\
& x_1 + x_2 = x_3 + x_4 \\
& x_1 + x_2 = x_5 \\
& x_3 + x_4 = x_5 \\
& x_i \in \mathbb{R}_+ \text{ für } i = 1, 2, 3, 4, 5.
\end{aligned}$$

Es ist allerdings gar nicht nötig, *alle* Kreise des Netzwerks als Bedingung zu verwenden, da diese offenbar redundant werden. Tatsächlich reicht es bereits, eine *Kreisbasis* zu verwenden.

Definition 2.21. Fundamentalkreise (Che76)
Sei ein Graph $G(V, E)$ mit spannendem Baum $\mathcal{T}$ gegeben. Ein *Fundamentalkreis* bezüglich $\mathcal{T}$ ist ein Kreis, der genau eine Nichtbaumkante beinhaltet und in Richtung dieser Kante durchlaufen wird.

Da Bäume kreisfrei sind und es daher nach Satz 1.9 immer nur genau einen Pfad zwischen zwei Knoten innerhalb eines Baumes gibt, gibt es zu jeder Nichtbaumkante genau einen Fundamentalkreis. Die zu den Fundamentalkreisen korrespondierende Teilmatrix B_f der Kreismatrix B lässt sich dann immer so sortieren, dass sie die Form $B_f = (Id, B_{\text{Rest}})$ besitzt, indem die Nichtbaumkanten auf die linke Seite gezogen werden.

Beispiel 2.22. Fundamentalkreismatrix nach (Che76)
Sei ein Netzwerk wie in Abbildung 2.4 gegeben mit einem Baum, der aus den Kanten e_5, e_6, e_7, e_8 besteht.

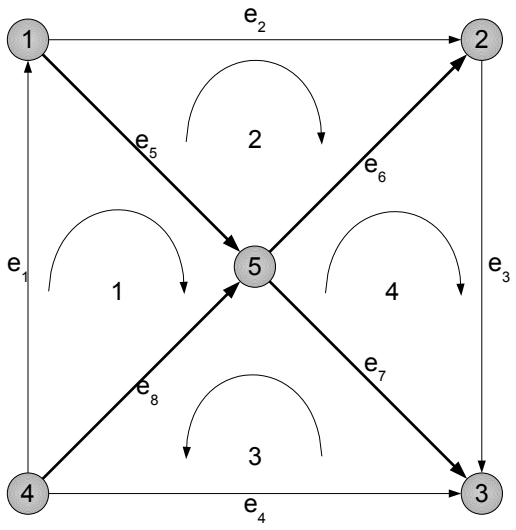


Abbildung 2.4: Ein Beispiel für Fundamentalkreise.

Dann lautet die Fundamentalkreismatrix

$$B_f = \begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \end{array} \begin{array}{c} \begin{array}{cccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 \end{pmatrix} \end{array}.$$

Beispiel 2.23. Fortsetzung von Beispiel 2.20.
Die dick gezeichneten Linien in Abbildung 2.5 stellen einen spannenden Baum im EAN aus Beispiel 2.20 dar.

Durch Verwendung der dadurch erzeugten Fundamentalkreise verkleinert sich

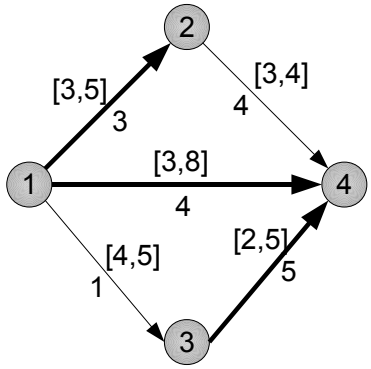


Abbildung 2.5: Spannender Baum zum Beispiel 2.20.

die Problemformulierung zu

$$\begin{aligned}
\min \quad & 3x_1 + 4x_2 + x_3 + 5x_4 + 4x_5 \\
\text{so dass} \quad & 3 \leq x_1 \leq 5 \\
& 3 \leq x_2 \leq 4 \\
& 4 \leq x_3 \leq 5 \\
& 2 \leq x_4 \leq 5 \\
& 3 \leq x_5 \leq 8 \\
& x_2 = -x_1 + x_5 \\
& x_3 = x_5 - x_4 \\
& x_i \in \mathbb{R}_+ \text{ für } i = 1, 2, 3, 4, 5.
\end{aligned}$$

Definition 2.24.
Zu einem gerichteten Graphen $G(V, E)$ heißt die Matrix $A \in \{-1, 0, 1\}^{|V| \times |E|}$ mit

$$a_{ie} = \begin{cases} 1, & \text{wenn } e = (i, j) \text{ für ein } j \in V, \\ -1, & \text{wenn } e = (j, i) \text{ für ein } j \in V, \\ 0, & \text{sonst.} \end{cases}$$

die *Knoten-Kanten-Inzidenzmatrix*. Ihre Transponierte wird entsprechend die Kanten-Knoten-Inzidenzmatrix genannt.

Es folgen einige Ergebnisse aus (Che76), die die Verwendung der Fundamentalkreismatrix rechtfertigen.

Satz 2.25. *(Che76)*
Für die Knoten-Kanten-Inzidenzmatrix A und die Kreismatrix B gilt, sofern die Kanten in der gleichen Reihenfolge angeordnet sind,

$$AB^T = 0.$$

Beispiel 2.26.
Im Graphen aus den Beispielen 2.20 und 2.23 lautet die Knoten-Kanten-Inzidenzmatrix

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & -1 & -1 \end{pmatrix}$$

und die Kreismatrix

$$B = \begin{pmatrix} 1 & 1 & 0 & 0 & -1 \\ 0 & 0 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 & 0 \end{pmatrix}.$$

Nachrechnen ergibt

$$AB^T = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Korollar 2.27. *(Che76)*
Der Rang der Kreismatrix eines gerichteten Graphens $G(V, E)$ ist $|E| - |V| + 1$.

Das heißt, dass die Spannung entlang eines beliebigen Kreises in einem Graphen darstellbar ist als Spannung entlang von Fundamentalkreisen. Wir erhalten eine neue Problemformulierung für (AF):

(AF) Finden eines kostenminimalen Fahrplans, dritte Formulierung (Sch09). Sei ein EAN $G(\mathcal{E}, \mathcal{A})$ mit unteren Schranken $l_{ij} \in \mathbb{N}$, oberen Schranken $u_{ij} \in \mathbb{N}$ und Gewichten $\omega_{ij} \in \mathbb{N}$ für jede Kante $(i, j) \in \mathcal{A}$ gegeben. Sei $\mathcal{T}$ ein spannender Baum und B_f die zu $\mathcal{T}$ gehörende Fundamentalkreismatrix. Gesucht sind nun Spannungen x_e für jede Kante $e \in \mathcal{A}$, so dass gilt

$$\begin{aligned} \min \quad & \sum_{(i,j)\in\mathcal{A}} w_{ij}x_{ij} && && (2.28) \\ \text{so dass} \quad & l_{ij} \le x_{ij} \le u_{ij} && \forall (i,j) \in \mathcal{A} && (2.29) \\ & B_f x = 0 && && (2.30) \\ & x_{ij} \in \mathbb{R}_+ && \forall (i,j) \in \mathcal{A}. && (2.31) \end{aligned}$$

2.2 Der Netzwerk-Simplex Algorithmus

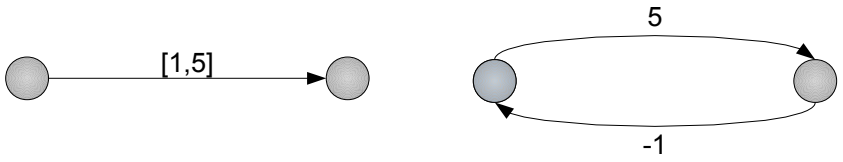
Karl Nachtigall und Jens Opitz wiesen in (NO08) darauf hin, dass das aperiodische Fahrplanproblem (AF) dual sei zu einem *minimum cost flow* Problem, welches wiederum durch den sehr effektiven Netzwerk-Simplex Algorithmus lösbar ist. In diesem Abschnitt wird zunächst der Dualitätszusammenhang untersucht und anschließend der Algorithmus nach (AMO93) vorgestellt.

2.2.1 Das Duale zum aperiodischen Fahrplanproblem

Um das duale Problem zu untersuchen, wird zunächst die Formulierung über die *Potentiale* verwendet:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in\mathcal{A}} \omega_{ij}(\pi_j - \pi_i) & \\ \text{so dass} \quad & l_{ij} \le \pi_j - \pi_i \le u_{ij} & \forall (i,j) \in \mathcal{A} \\ & \pi_i \in \mathbb{R}_+ & \forall i \in \mathcal{E}. \end{aligned}$$

Um die unteren und oberen Schranken besser handhaben zu können, wird nun jede Kante des ursprünglichen Problems ersetzt durch zwei antiparallele Kanten. Die neu erhaltene Kantenmenge werde mit $\mathcal{A}'$ bezeichnet. Die obere Schranke der neu hinzugefügten Kante ist nun das Negative der unteren Schranke der ursprünglichen Kante. War das ursprüngliche Kantengewicht ω, so wird der Kante, die die alte Orientierung behalten hat, das neue Gewicht $\omega/2$, und der Kante, die in verkehrter Orientierung steht, das Gewicht $-\omega/2$ zugeordnet. Dadurch reicht es, sich auf obere Schranken zu beschränken.



(a) Ursprüngliche Kante. (b) Daraus erzeugte Kanten.

Abbildung 2.6: Erzeugung zweier Kanten aus einer.

Es entsteht die neue Problemformulierung

$$\begin{aligned} \min \quad & \sum_{(i,j)\in\mathcal{A}'} \omega'_{ij}(\pi_j - \pi_i) & \\ = \min \quad & \sum_{(i,j)\in\mathcal{A}} \omega_{ij}(\pi_j - \pi_i) & \\ \text{so dass} \quad & \pi_j - \pi_i \le u'_{ij} & \forall (i,j) \in \mathcal{A}' \\ & \pi_i \in \mathbb{R}_+ & \forall i \in \mathcal{E}. \end{aligned}$$

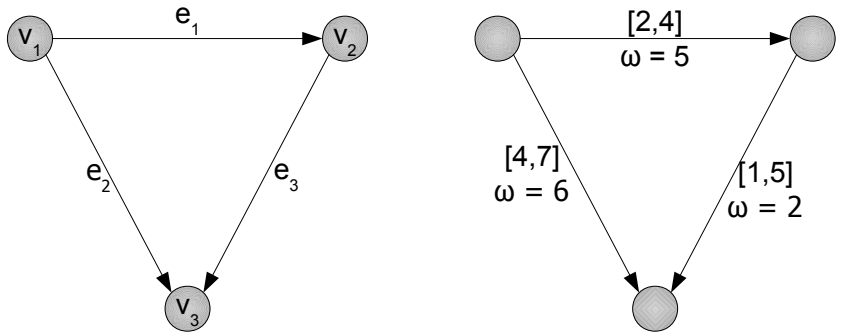
Sei nun $A \in \{-1,0,1\}^{|\mathcal{A}'|\times|\mathcal{E}|}$ die Kanten-Knoten-Inzidenzmatrix. Dann kann die Zielfunktion $\min \omega'_{ij}(\pi_j - \pi_i)$ umgeschrieben werden zu $\min -\omega'^T A\pi$. Das hinzugekommene Minuszeichen ist nötig, da die *ausgehenden* Kanten in der Inzidenzmatrix positive Einträge darstellen, in der Zielfunktion dagegen negativ sein müssen. Indem das Minimum durch ein Maximum ersetzt wird, kann wiederum auf das Minuszeichen verzichtet werden und das Problem wird zu

$$\begin{aligned} \max \quad & \omega'^T A\pi & \\ \text{so dass} \quad & A\pi \ge -u' & \\ & \pi_i \in \mathbb{R}_+ & \forall i \in \mathcal{E}. \end{aligned}$$

Das Problem ist nun in einer Form, zu der sich das duale Problem direkt notieren lässt nach Abschnitt 1.3:

$$\begin{array}{rll} \min & {u'}^T x & \\ \text{so dass} & x^T A \leq A^T \omega' & \\ & x_i \in \mathbb{R}_+ & \forall i \in \mathcal{A}'. \end{array}$$

Beispiel 2.32.



(a) Ursprüngliches EAN. (b) Gewichte und Schranken.

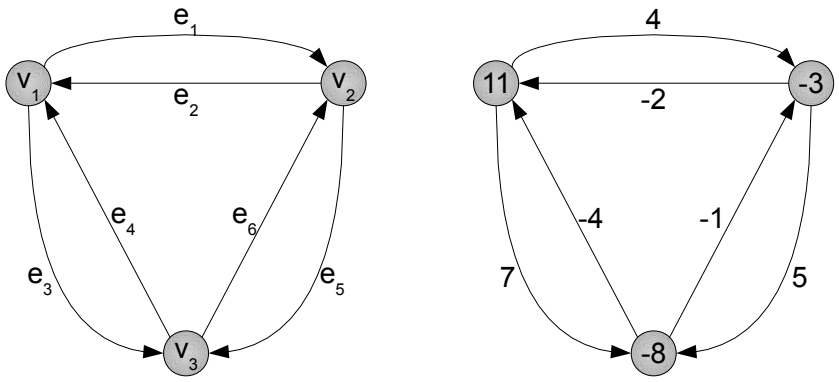
Abbildung 2.7: Ein Beispiels-EAN.

Sei ein EAN wie in Abbildung 2.7(a) dargestellt gegeben mit Schranken und Kosten wie in Abbildung 2.7(b). Zunächst werden die Kanten verdoppelt, um untere Schranken durch obere zu ersetzen. Die zu der neu enstandenen Kantenmenge $\mathcal{A}'$ gehörige Inzidenzmatrix A lautet dann

$$A = \begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 1 & 0 & -1 \\ -1 & 0 & 1 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix} \end{array}$$

und das primale Problem lautet

$$\begin{array}{rl} \min & 5(\pi_2 - \pi_1) + 6(\pi_3 - \pi_1) + 2(\pi_3 - \pi_2) \\ \text{so dass} & \pi_2 - \pi_1 \leq 4 \\ & \pi_1 - \pi_2 \leq -2 \\ & \pi_3 - \pi_1 \leq 7 \\ & \pi_1 - \pi_3 \leq -4 \\ & \pi_3 - \pi_2 \leq 5 \\ & \pi_2 - \pi_3 \leq -1 \\ & \pi_i \in \mathbb{R}_+ \text{ für } i = 1, 2, 3. \end{array}$$



(a) Das erweiterte EAN. (b) Bedarf und Kosten.

Abbildung 2.8: Der Dualitätszusammenhang an einem Beispiel.

Die dazu duale Probleminstanz lässt sich nun bilden, indem für jeden Knoten die Gewichte der ausgehenden Kanten addiert und die Gewichte der eingehenden Kanten subtrahiert werden. Dies ist der *Vorrat* des Knotens. Gesucht ist nun ein Wert für jede Kante, so dass der Vorrat beachtet wird, mit minimalen Kosten.

$$\begin{aligned} \min \quad & 4x_1 - 2x_2 + 7x_3 - 4x_4 + 5x_5 - 1x_6 \\ \text{so dass} \quad & x_1 - x_2 + x_3 - x_4 \leq 11 \\ & x_5 - x_6 + x_2 - x_1 \leq -3 \\ & x_4 - x_3 + x_6 - x_5 \leq -8 \\ & x_i \in \mathbb{R}_+ \text{ für } i = 1, \ldots, 6. \end{aligned}$$

Die Bedingung $x^T A \leq A^T \omega'$ kann sogar verschärft werden zu $x^T A = A^T \omega'$.

Satz 2.33.
Sei ein Problem

$$\begin{aligned} \textit{(P)} \quad \min \quad & c^T x \\ \textit{so dass} \quad & Ax \leq b \\ & x_i \in \mathbb{R}_+ \qquad \forall i \in E \end{aligned}$$

gegeben. Angenommen, jede Spalte der Matrix A und der Vektor b summieren sich auf 0*, das heißt*

$$\vec{1}^T A = \vec{1}^T b = 0.$$

Dann ist die Lösungsmenge des Problems (P) identisch zu der des Problems

$$\begin{aligned} \textit{(P')} \quad \min \quad & c^T x \\ \textit{so dass} \quad & Ax = b \\ & x_i \in \mathbb{R}_+ \qquad \forall i \in E. \end{aligned}$$

Beweis. Offenbar ist die Lösungsmenge von (P') in der von (P) enthalten. Es bleibt nur die andere Richtung zu zeigen.

Sei x eine zulässige Lösung von (P) mit $Ax = b^*$ für ein b^*. Würde $b_i^* < b_i$ für ein $i \in E$ gelten, wäre dennoch $\vec{1}^T b^* = \vec{1}^T Ax = \vec{1}^T b = 0$ - da kein b_i^* größer sein darf als das entsprechende b_i, erhalten wir einen Widerspruch. Folglich ist $b^* = b$ und beide Lösungsmengen sind gleich. □

Satz 2.34.
Sei

$$\begin{array}{lll} (D) \quad \min & c^T x & \\ so\ dass & Ax \leq b & \\ & x_i \in \mathbb{R}_+ & \forall i \in \mathcal{A}' \end{array}$$

das duale Problem zu einem aperiodischen Fahrplanproblem. Dann gilt $\vec{1}^T A = \vec{1}^T b = 0$.

Beweis. Da A die Inzidenzmatrix eines Graphen ist und jede Kante genau einen Ursprung und ein Ziel besitzt, ist $\vec{1}^T Ax = 0$. Andererseits ist auch $\vec{1}^T b = \vec{1}^T \omega' A^T = 0$, da es nach Konstruktion zu jeder Kante (i, j) mit Bedarf ω'_{ij} es eine Kante (j, i) mit Bedarf $\omega_{ji} = -\omega'_{ij}$ gibt. □

Die verschärfte Form entspricht nun einem sogenannten *minimum cost flow*-Problem („Minimaler-Kosten-Fluss“).

(MCF) Minimum Cost Flow (AMO93)
Sei ein gerichteter Graph $G(V, E)$ mit Kosten c_{ij} und Kapazitäten u_{ij} für jede Kante (i, j), sowie ein Vorrat b_i für jeden Knoten i gegeben. Das *minimum cost flow*-Problem lautet dann

$$\min \quad f(x) = \sum_{(i,j)\in E} c_{ij} x_{ij} \tag{2.35}$$

$$\text{so dass} \quad \sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = b_i \qquad \forall i \in V \tag{2.36}$$

$$0 \leq x_{ij} \leq u_{ij} \qquad \forall (i, j) \in E. \tag{2.37}$$

Zusammenfassend ist also in diesem Abschnitt gezeigt worden:

Korollar 2.38. *Dualität des aperiodischen Fahrplanproblems*
Das aperiodische Fahrplanproblem ist dual zu einem minimum cost flow Problem mit unendlich großen Kapazitäten.

Bemerkung 2.39.
Wird im Problem (AF) anstelle von

$$\pi_i \in \mathbb{R}^+$$

der Raum aller reellen Zahlen verwendet, das heißt

$$\pi_i \in \mathbb{R},$$

so ist das dazugehörige duale Problem bereits der Form

$$Ax = b,$$

ohne dass Satz 2.33 benötigt wird.

2.2.2 Netzwerk-Simplex für Minimum Cost Flow Probleme

Das (MCF)-Problem in seiner Formulierung als lineares Programm lässt sich mit dem bekannten Simplex-Verfahren lösen, doch würde damit nicht die zugrunde liegende Struktur des Netzwerkes ausgenutzt. Durch die verlorene Strukturinformation ist das Verfahren nicht so effektiv, wie es sein könnte. Daher wurde das *Netzwerk-Simplex* Verfahren entwickelt als Adaption des Simplex-Verfahrens an die spezifische Problemstruktur, was den Algorithmus hoch effektiv werden lässt. Da sich dieser Abschnitt ausschließlich mit dem Problem (MCF) beschäftigt, ist mit einer „zulässigen Lösung“ immer eine zulässige Lösung für eine (MCF)-Instanz gemeint.
Die grundlegende Idee des Verfahrens ist die sukzessive Verbesserung einer zulässigen Lösung durch Ausnutzung von Kreisen mit negativen Kosten. Um diese zu identifizieren, folgen einige Definitionen.

Definition 2.40. Freie und beschränkte Kanten (AMO93)
Sei x eine zulässige Lösung. Eine Kante (i, j) heißt *freie Kante*, wenn $0 < x_{ij} < u_{ij}$ gilt, ansonsten wird sie *beschränkte Kante* genannt.

Definition 2.41. Kreisfreie Lösung (AMO93)
Eine Lösung x heißt *kreisfrei*, wenn im Graphen kein Kreis aus freien Kanten existiert.

Wie bereits gezeigt wurde, lässt sich anstatt der Verwendung von *allen* Kreisen häufig ein Problem auf *Fundamentalkreise* reduzieren. Diese wiederum entstehen durch spannende Bäume.

Definition 2.42. Spannender-Baum-Lösung (AMO93)
Eine Lösung x heißt eine *Spannender-Baum-Lösung*, wenn es einen spannenden Baum gibt, so dass jede Nichtbaumkante beschränkt ist.

Beispiel 2.43.
Sei ein Graph wie in Abbildung 2.9 gegeben mit einem Bedarfvektor

$$b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} 7 \\ -2 \\ -3 \\ -2 \end{pmatrix}.$$

Die Kapazität jeder Kante ist jeweils durch den zweiten Wert in der dazugehörigen Klammer gegeben. Mit der zulässigen Lösung x, die dem ersten Eintrag abzulesen ist, bilden die fett gedruckten Kanten eine Spannender-Baum-Lösung.

Durch einen Kreis, dessen Kostensumme negativ ist, lässt sich der Zielfunktionswert verringern, indem der Fluss jeder Kante erhöht wird, und gleichzeitig die Bedarfsbedingung jedes Knotens erhalten. Das ist immer so lange möglich, bis eine Kante an ihre untere oder obere Schranke stößt. Analog gilt das auch für Kreise mit positiven Kosten. Daher reicht es aus, nur die Lösungen zu betrachten, die keinen freien Kreis mehr beinhalten.

Satz 2.44. *Kreisfreiheit optimaler Lösungen (AMO93)*
Ist der Zielfunktionswert einer (MCF)-Instanz von unten beschränkt, so gibt es immer eine optimale Lösung, die kreisfrei ist.

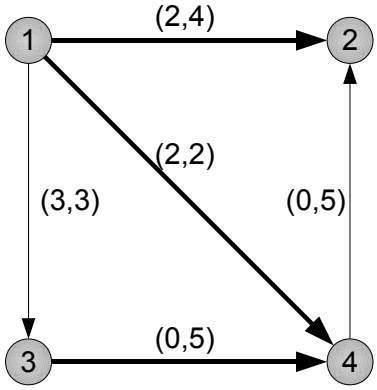


Abbildung 2.9: Beispiel für eine Spannender-Baum-Lösung.

Freie Kanten in einer kreisfreien Lösungen bilden immer einen *Wald*. Durch Hinzufügen von beschränkten Kanten lässt sich dieser Wald daher zu einem spannenden Baum ergänzen mit der Eigenschaft, dass Nichtbaumkanten beschränkt sind. Auf diese Weise lässt sich eine kreisfreie Lösung in eine Spannender-Baum-Lösung transformieren und es gilt:

Korollar 2.45. *Spannender-Baum-Optimalität (AMO93)*
Ist der Zielfunktionswert einer (MCF)-Instanz von unten beschränkt, so gibt es immer eine optimale Spannender-Baum-Lösung.

In einer Spannender-Baum-Lösung ist die Information, ob der Fluss einer Nichtbaumkante (i, j) bei 0 oder bei u_{ij} liegt, nur indirekt enthalten. Da dies für den Netzwerk-Simplex Algorithmus von Interesse ist, wird sie expliziet durch Spannender-Baum-Strukturen angegeben.

Definition 2.46. Spannender-Baum-Struktur (AMO93)
Sei $\mathcal{T}$ ein spannender Baum, L die Menge der Nichtbaumkanten, deren Fluss bei 0 liegt, und U die Menge der Nichtbaumkanten, deren Fluss gleich der jeweils oberen Schranke ist. Dann wird das Tripel $(\mathcal{T}, L, U)$ eine Spannender-Baum-Struktur (im Sinne des Netzwerk-Simplex Verfahrens) genannt.

Jede Spannender-Baum-Lösung entspricht somit eine Spannender-Baum-Struktur und andersherum lässt sich aus einer Spannender-Baum-Struktur eine eindeutige Spannender-Baum-Lösung gewinnen, indem die Gleichungen für den Knotenbedarf gelöst werden.
Die Dualität zu Knotenpotentialen lässt sich verwenden, um ein Optimalitätskriterium zu formulieren:

Satz 2.47. *Optimalitätsbedingungen (AMO93)*
Eine Spannender-Baum-Struktur $(\mathcal{T}, L, U)$ ist eine optimale Spannder-Baum-Struktur, wenn sie zulässig ist und es Knotenpotentiale π gibt, so dass für die reduzierten Kosten $c^{\pi}_{ij} := c_{ij} - \pi_i + \pi_j$ gilt:

$$c^{\pi}_{ij} = 0 \quad \forall (i,j) \in \mathcal{T} \tag{2.48}$$

$$c^{\pi}_{ij} \geq 0 \quad \forall (i,j) \in L \tag{2.49}$$

$$c^{\pi}_{ij} \leq 0 \quad \forall (i,j) \in U \tag{2.50}$$

Aus einer Spannender-Baum-Struktur lassen sich die Knotenpotentiale berechnen, indem das Potential eines beliebigen Knotens auf 0 gesetzt wird, dann entlang des spannenden Baumes jeder weitere Knoten besucht wird und entsprechend den Kantenorientierungen jeweils die Kosten auf die Potentiale hinzuaddiert und subtrahiert werden. Damit ist unter anderem die Ganzzahligkeit von aperiodischen Fahrplanproblemen gewonnen.

Satz 2.51. *Ganzzahligkeit (AMO93)*
Sind alle Kosten einer (MCF)-Instanz ganzzahlig, so gibt es immer optimale Knotenpotentiale, die ganzzahlig sind.

Korollar 2.52.
Sind die Gewichte ω einer (AF)-Instanz ganzzahlig, so gibt es immer eine ganzzahlige optimale Lösung.

Es wird nun eine Übersicht über den gesamten Algorithmus gegeben.

Algorithmus 4 Der Netzwerk-Simplex Algorithmus nach (AMO93).

Eingabe: Eine (MCF)-Probleminstanz und eine zulässige Baumstruktur (T, L, U).
Ausgabe: Ein optimaler Fluss x und optimale Knotenpotentiale π.
1: Bestimme den Fluss x und die Knotenpotentiale π, die der Baumstruktur (T, L, U) zugeordnet sind.
2: **while** (Eine Nichtbaumkante verletzt die Optimalitätsbedingungen) **do**
3: Wähle eine Nichtbaumkante (i, j), die die Optimalitätsbedingungen verletzt.
4: Füge die Kante (i, j) dem Baum hinzu und bestimme eine zu entfernende Kante (p, q).
5: Erneuere den Baum und die Lösungen x und π.
6: **end while**

Eine zulässige Baumstruktur als Anfangslösung lässt sich sogar in linearer Zeit berechnen (AMO93). Die Wahl, welche Pivotoperation durchgeführt werden soll, trägt wesentlich zur Geschwindigkeit dieses Algorithmus bei und ist offen für eine Reihe von Ansätzen.

Beispiel 2.53.

1. „Dantzigs Pivotregel“. Es werden alle Nichtbaumkanten betrachtet und diejenige mit der größten Verletzung der Optimalitätsbedingungen wird in die Spannender-Baum-Struktur aufgenommen. Auf diese Weise werden ändert sich der Zielfunktionswert in Hinsicht auf die Anzahl der verwendeten Iterationen besonders schnell, jede einzelne Iteration braucht dafür verhältnismäßig lange.

2. „Erste zulässige Kante“-Regel. Es werden sukzessiv alle Nichtbaumkanten durchsucht und die *erste* zulässige Kante, die den Zielfunktionswert verringert, wird aufgenommen. Die Iterationen sind dadurch schnell, doch verändern sie den Zielfunktionswert durchschnittlich weniger als bei Verwendung von Dantzigs Regel.

3. „Kandidatenliste“-Regel. Dieses Verfahren versucht, die Vorteile der ersten zwei Regeln zu kombinieren. Es wird zunächst eine Liste aller Kanten erstellt, deren Aufnahme in die Spannender-Baum-Struktur zulässig ist.

Dann wird diese Liste sequentiell verwendet und in jedem Schritt diejenigen Operationen, die nicht mehr zulässig sind, gelöscht.

Durch Aufnahme einer zusätzlichen Kante in den spannenden Baum entsteht genau ein Kreis. Entlang dieses Kreises wird der Fluss reduziert, bis eine Kante die weitere Reduktion blockiert. Diese Kante wird dann aus der Spannender-Baum-Struktur entfernt und der Pivot-Schritt ist vollständig.

2.3 Elektrische Netzwerke

Die Theorie der elektrischen Netzwerke gilt als eine der bedeutendsten Anwendungen der Graphentheorie in der Physik (siehe (Che76)). Gleichzeitig besteht ein enger Zusammenhang zu dem aperiodischen Fahrplanproblem, der in diesem Abschnitt näher beleuchtet wird. Zunächst werden elektrische Netzwerke definiert und grundlegende Eigenschaften festgehalten. Anschließend wird ein Weg dargestellt, Fahrplanprobleme in Probleme elektrischer Netzwerke umzuwandeln.

Der deutsche Physiker Gustav Kirchhoff veröffentlichte im Jahre 1847 die Schrift „Über die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Verteilung galvanischer Ströme geführt wird“ (siehe (Che76)), in der er zwei Gesetze darstellte, die ihm zu Ehren heute als die zwei *Kirchhoffschen Gestze* bekannt sind. Gemeinsam mit dem Ohmschen Gesetz beschreiben sie elektrische Netzwerke vollständig.

Definition 2.54. Lineares elektrisches Netzwerk (Che76)
Ein *lineares (zeitkonstantes) elektrisches Netzwerk* ist ein gerichteter Graph $G(V, E)$ mit zwei Vektoren $I, U \in \mathbb{R}^{|E|}$, der folgende Eigenschaften erfüllt:

1. Das erste Kirchhoffsche Gesetz:

$$AI = 0, \tag{2.55}$$

   wobei A die Knoten-Kanten-Inzidenzmatrix darstellt.

2. Das zweite Kirchhoffsche Gesetz:

$$BU = 0, \tag{2.56}$$

   wobei B die Kreis-Kanten-Inzidenzmatrix darstellt.

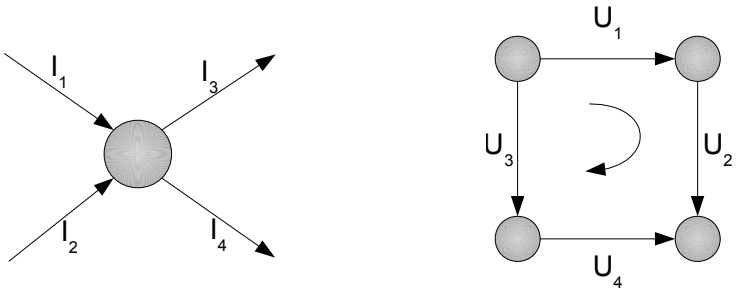
3. Das verallgemeinerte Ohmsche Gesetz:

$$U = E + ZI \tag{2.57}$$

   oder

$$I = J + YU \tag{2.58}$$

   wobei J, E, Z, Y gegebene Matrizen und Vektoren sind. Welche Formulierung verwendet wird, ist davon abhängig, ob Stromstärken oder Spannungen als Quellen vorgegeben sind.



(a) Erstes Kirchhoffsches Gesetz. (b) Zweites Kirchhoffsches Gesetz.

Abbildung 2.10: Veranschaulichung der beiden Kirchhoffschen Gesetze

Die erste Bedingung formuliert, dass der *Strom* in einen Knoten hinein immer gleich dem Strom hinaus sein muss, also $\sum_{j:(i,j)\in E} I_{ij} - \sum_{j:(j,i)\in E} I_{ji} = 0 \; \forall i \in V$. Die zweite Bedingung dagegen bedeutet, dass die *Spannung* in einem Kreis sich immer auf 0 summiert, also $\sum_{(i,j)\in c} U_{ij} = 0$ für alle Kreise c. Abbildung 2.10 veranschaulicht diese beiden Gesetze. In der dritten Bedingung schließlich sind die eigentlichen Bauteile des elektrischen Netzwerks wie zum Beispiel Widerstände in Form von linearen Verknüpfungen der Größen Spannung und Strom gegeben.

Definition 2.59.
Ein elektrisches Netzwerk, das keine oder mehr als eine Lösung besitzt, wird *entartet* genannt.

Beispiel 2.60.
Betrachte das elektrische Netzwerk, das in Abbildung 2.11 gegeben ist.
Die Eingangsspannung U_0 sei 5V groß und die Widerstände betragen

$$R_1 = 1\ \Omega, \qquad R_2 = 2\ \Omega,$$
$$R_3 = 2\ \Omega, \qquad R_4 = 1\ \Omega.$$

Wie groß sind die daraus resultierenden Spannungen und Stromstärken?
Um das beantworten zu können, werden zunächst die Knoten-Kanten-Inzidenzmatrix A und die Kreis-Kanten-Inzidenzmatrix B bestimmt.

$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{pmatrix} -1 & -1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & -1 \end{pmatrix} \end{array},$$

$$B = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{pmatrix} -1 & 1 & -1 & 1 & 0 \\ -1 & 0 & -1 & 0 & -1 \\ 0 & -1 & 0 & -1 & -1 \end{pmatrix} \end{array},$$

wobei $c_1 = (v_4, v_3, v_1, v_2, v_4)$, $c_2 = (v_4, v_1, v_2, v_4)$ und $c_3 = (v_4, v_1, v_3, v_4)$. Die

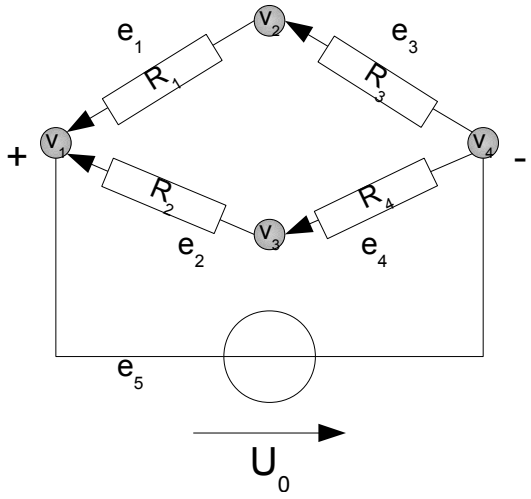


Abbildung 2.11: Ein elektrisches Netzwerk.

Kirchhoffschen Gesetze fordern dann

$$\begin{aligned} AI &= 0 \\ \text{und } BU &= 0. \end{aligned}$$

Die Spannungsquelle geht in die Formulierung des Ohmschen Gesetzes mit ein:

$$\begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -5 \end{pmatrix} + \begin{pmatrix} R_1 & 0 & 0 & 0 & 0 \\ 0 & R_2 & 0 & 0 & 0 \\ 0 & 0 & R_3 & 0 & 0 \\ 0 & 0 & 0 & R_4 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \end{pmatrix}.$$

Durch Umformung ergibt sich

$$\begin{aligned} 5 &= U_1 + U_3 \\ &= U_2 + U_4 \\ \text{und} \quad I_1 &= I_3 \\ I_2 &= I_4. \end{aligned}$$

Aus $U_i = R_i I_i$ für $i \in \{1, 2, 3, 4\}$ folgt die eindeutige Lösung

$$\begin{aligned} U_2 = U_3 &= 5/3 \\ U_1 = U_4 &= 10/3 \\ I_2 = I_4 &= 5/3 \\ I_1 = I_3 &= 10/3 \\ I_5 &= 5. \end{aligned}$$

An diesem Beispiel wird bereits ersichtlich, dass wie in der aperiodischen Fahrplangestaltung einige Kreisbedingungen überflüssig sind. Selbstverständlich kann auch hier auf die Fundamentalkreismatrix zurückgegriffen werden, um diese redundanten Bedinungen zu entfernen. Vielleicht nicht ebenso offensichtlich lassen sich aber auch die Knotenregeln reduzieren. Dafür wird das Konzept der *Schnitte* eingeführt.

Definition 2.61. Rang eines Graphen (Che76)
Sei ein Graph $G(V, E)$ mit k Zusammenhangskomponenten gegeben. Der *Rang* von G ist dann $r = |V| - k$.

Bemerkung 2.62.
Der Rang eines Graphen entspricht daher der Anzahl der Kanten eines spannenden Waldes.

Definition 2.63. Schnitt (Che76)
Sei ein Graph $G(V, E)$ und eine Knotenpartition $V = V_1 \dot\cup V_2$ gegeben. Dann nennt man die Kantenmenge $c = \{(i, j) \in E : i \in V_1 \wedge j \in V_2 \text{ oder } i \in V_2 \wedge j \in V_1\}$ einen *Schnitt*.
Die durch einen Schnitt induzierte Orientierung einer Schnittkante geht immer von V_1 nach V_2.

Bemerkung 2.64. (Che76)
Es gibt $2^r - 1$ nichtleere Schnitte in einem Graphen G mit Rang r.

Definition 2.65. Die Schnitt-Matrix (Che76)
Sei ein gerichteter Graph $G(V, E)$ gegeben und bezeichne C die Menge aller Schnitte in G, wobei bis auf die Orientierung identische Schnitte ausgelassen werden. Dann wird die Matrix $Q \in \{-1, 0, 1\}^{|C|\times|E|}$ mit

$$q_{ce} = \begin{cases} 1, & \text{wenn } e \text{ in Schnitt } c \text{ in der} \\ & \text{durch den Schnitt induzierten Orientierung enthalten ist,} \\ -1, & \text{wenn } e \text{ in Schnitt } c \text{ mit} \\ & \text{entgegengesetzter Orientierung enthalten ist,} \\ 0 & \text{sonst} \end{cases}$$

die Schnitt-Matrix zu G genannt.

Satz 2.66. *(Che76)*
Die Kreismatrix B und die Schnittmatrix Q eines Graphen G sind bei gleicher Kantenanordnung orthogonal zueinander:

$$QB^T = 0.$$

Wie im Fall der Fundamentalkreise sind viele Schnitte eines Graphen in dem Sinne redundant, dass sie durch eine Menge von Schnitten erzeugt werden. Dies sind die Fundamentalschnitte.

Definition 2.67. Fundamentalschnitte (Che76)
Sei ein zusammenhängender, gerichteter Graph G und ein spannender Baum $\mathcal{T}$ gegeben. Dann heißt ein Schnitt c ein *Fundamentalschnitt* bezüglich $\mathcal{T}$, wenn er genau eine Baumkante enthält.

Die Abbildung 2.12 veranschaulicht im ungerichteten Fall, wie Fundamentalschnitte aus einem spannenden Baum gewonnen werden: Da Bäume kreisfrei sind, zerfällt der spannende Baum durch Entfernen einer Kante in zwei Zusammenhangskomponenten. Die Knoten dieser Komponenten werden nun in unterschiedliche Partitionen gesetzt.

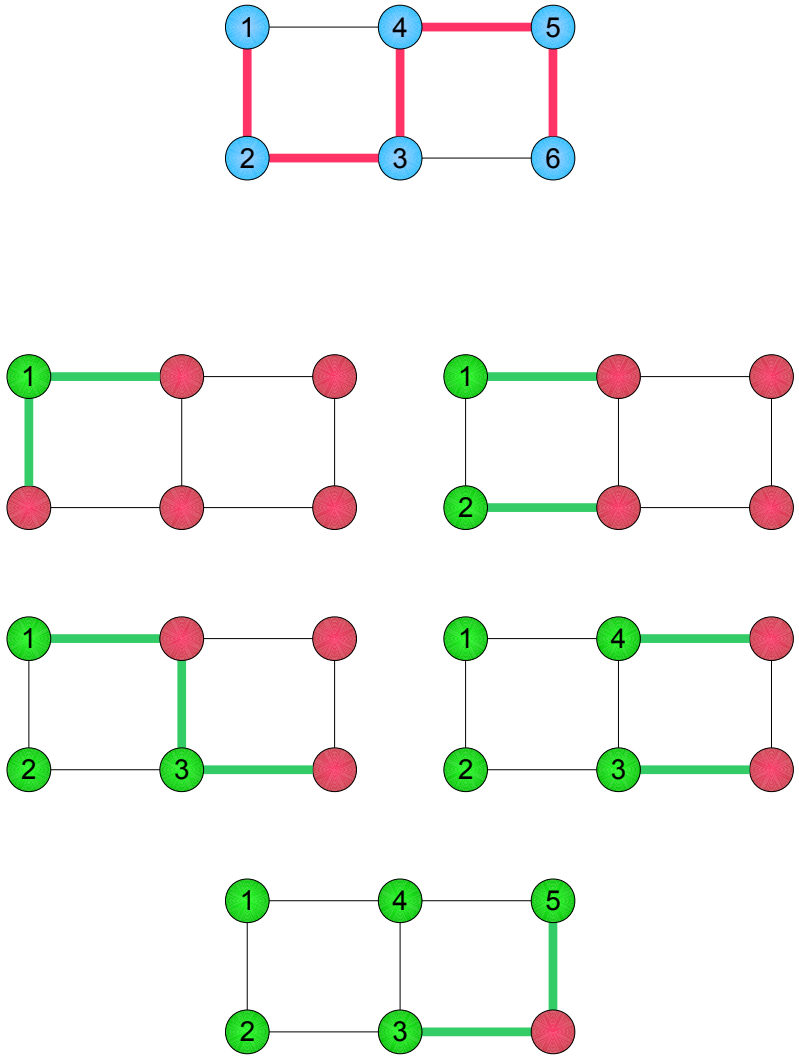


Abbildung 2.12: Veranschaulichung von Fundamentalschnitten. Im Graphen ganz oben ist in Rot der spannende Baum eingezeichnet. Darunter sind in grün die 5 Fundamentalschnitte gefärbt. Rote und grüne Knotenfärbungen stehen für die Wahl der entsprechenden Knotenpartition.

Definition 2.68. Fundamentalschnittmatrix (Che76)
Die Teilmatrix $Q_f \in \{-1, 0, 1\}^{m \times |E|}$ einer Schnittmatrix Q eines zusammenhängenden, gerichteten Graphens $G(V, E)$ mit spannendem Baum $\mathcal{T}$ heißt *Fundamentalschnittmatrix*, wenn jede Zeile einem Fundamentalschnitt bezüglich $\mathcal{T}$ entspricht.

Korollar 2.69. *(Che76)*
Die Fundamentalkreismatrix B_f und die Fundamentalschnittmatrix Q_f eines Graphen G mit spannendem Baum $\mathcal{T}$ sind bei gleicher Kantenanordnung orthogonal zueinander:

$$Q_f B_f^T = 0.$$

Beispiel 2.70. Sei ein Graph wie in Abbildung 2.13 auf Seite 44 gegeben mit in fetten Kanten gedrucktem spannenden Baum.

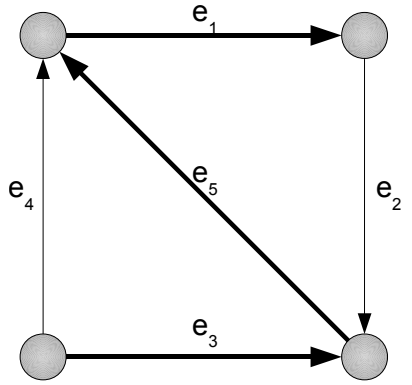


Abbildung 2.13: Ein Graph mit spannendem Baum.

Dann lautet die Fundamentalkreismatrix

$$B_f = \begin{array}{c} \\ e_2 \\ e_4 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 & -1 \end{pmatrix} \end{array}$$

und die Fundamentalschnittmatrix

$$Q_f = \begin{array}{c} \\ e_1 \\ e_3 \\ e_5 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & 1 & 1 \end{pmatrix} \end{array}.$$

Wie man leicht nachrechnen kann, gilt tatsächlich

$$Q_f B_f^T = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

Es folgt die erwähnte Reformulierung des ersten Kirchhoffschen Gesetzes:

Satz 2.71. *(Che76)*

$$AI = 0 \Leftrightarrow QI = 0$$

Da die Entfernung der Kanten eines Schnittes genau eine zusätzliche Zusammenhangskomponente erzeugt, sagt dieser Satz physikalisch interpretiert nichts anderes, als dass der Stromfluss in eine Graphkomponente hinein immer dem Fluss hinaus entspricht.

Wir nehmen nun an, es wird wie im gegebenen Beispiel nur eine einzige Stromquelle verwendet, die die Knoten a und b verbinde. Der entstehende Stromfluss ist dann $I_a := \sum_{x:(a,x)\in E} I_{ax} = \sum_{x:(x,a)\in E} I_{xa}$.

Definition 2.72. Energiedissipation (DS84)
Die (totale) Energiedissipation eines linearen elektrischen Netzwerkes ist gegeben durch

$$\begin{aligned} E &= \sum_{(x,y)\in E} I_{xy}^2 R_{xy} \\ &= \sum_{(x,y)\in E} I_{xy} U_{xy}. \end{aligned}$$

Definition 2.73. Einheitsfluss (DS84)
Ein Fluss I von a nach b heißt *Einheitsfluss*, wenn gilt

$$I_a = 1.$$

Das folgende Ergebnis ist bereits 1879 veröffentlicht worden:

Satz 2.74. *Thompsons Prinzip (DS84)*
Der Einheitsfluss I von a nach b, der die Kirchhoffschen Gesetze und das Ohmsche Gesetz erfüllt, minimiert die Energiedissipation über alle das Ohmsche Gesetz erfüllende Einheitsflüsse von a nach b:

$$\sum_{(x,y)\in E} I_{xy}^2 R_{xy} = \min_J \sum_{(x,y)\in E} J_{xy}^2 R_{xy}.$$

Korollar 2.75.
In einem nicht entarteten elektrischen Netzwerk minimiert der das Ohmsche Gesetz erfüllende Einheitsfluss I von a nach b genau dann die Energiedissipation, wenn er die Kirchhoffschen Gesetze erüllt.

Beispiel 2.76.
Im elektrischen Netzwerk aus Beispiel 2.60 wurde eine Spannungsquelle und damit die erste Formulierung des Ohmschen Gesetzes verwendet. Um einen Einheitsfluss zu erzwingen, wird nun auf der Kante e_5 statt dessen eine Stromquelle mit $I_5 = 1$ als Eingangsgröße verwendet.
Da nach wie vor gefordert wird, dass der elektrische Strom ein *Fluss* ist, wird das erste Kirchhoffsche Gesetz zwangsweise immer erfüllt. Wird als „Lösung“ der Stromfluss

$$I_1 = I_2 = I_3 = I_4 = 0{,}5$$

betrachtet, so ist die dadurch verursachte Energiedissipation

$$\begin{aligned}
&\sum_{(x,y)\in E} I_{xy}^2 R_{xy} \\
&= \left(\left(\frac{1}{2}\right)^2 \cdot 1 + \left(\frac{1}{2}\right)^2 \cdot 2 + \left(\frac{1}{2}\right)^2 \cdot 2 + \left(\frac{1}{2}\right)^2 \cdot 1\right) \\
&= \frac{3}{2},
\end{aligned}$$

während die berechnete optimale Lösung

$$\begin{aligned}
I_2 = I_4 = 1/3 \\
I_1 = I_3 = 2/3
\end{aligned}$$

eine geringere Energiedissipation von

$$\begin{aligned}
&\sum_{(x,y)\in E} I_{xy}^2 R_{xy} \\
&= \left(\left(\frac{2}{3}\right)^2 \cdot 1 + \left(\frac{1}{3}\right)^2 \cdot 2 + \left(\frac{1}{3}\right)^2 \cdot 2 + \left(\frac{2}{3}\right)^2 \cdot 1\right) \\
&= \frac{12}{9}
\end{aligned}$$

verursacht. Die Spannungen, die im suboptimalen Fall durch das Ohmsche Gesetz berechnet werden, erfüllen offenbar nicht die Maschenregel.

Es ist nun der Zusammenhang hergestellt zwischen dem Problem, Stromfluss und Spannung eines elektrischen Netzwerkes zu finden, und einem Optimierungsproblem. Die Ähnlichkeit des zweiten Kirchhoffschen Gesetzes zu der Kreisbedingung in (AF) ist offenbar. Lassen sich die Probleme aufeinander zurückführen? Betrachten wir zunächst die Zielfunktionen: Nach Thompsons Prinzip wird in einem elektrischen Netzwerk die Energiedissipation $\sum_{x,y} I_{xy}U_{xy}$ minimiert, in einem aperiodischen Fahrplanproblem dagegen die gewichtete Fahrzeit $\sum_{i,j} \omega_{ij}x_{ij}$. Da die Spannungen x auch das zweite Kirchhoffsche Gesetz erfüllen, ist eine Identifizierung

$$U \sim x$$

naheliegend. Die Passagiergewichte sind ein naheliegender Kandidat für den Stromfluss, doch selbst wenn in jedem Ereignis so viele Passagiere ein- wie aussteigen, um die Knotenregel zu erfüllen, wird es weiterhin Knoten geben, die diese Bedingung verletzen:

Satz 2.77.
Sei eine (AF)-Instanz gegeben. Ist die zulässige Lösungsmenge nicht leer, so gibt es keinen gerichteten Kreis c mit $l_{ij} \neq 0$ für eine Kante $(i,j) \in c$.

Beweis. Sei c ein gerichteter Kreis. Durch die Bedingung $B_f x = 0$ folgt $\sum_{(i,j)\in c} x_{ij} = 0$. Das ist genau dann der Fall, wenn $x_{ij} = 0\ \forall (i,j) \in c$. Ist also eine untere Schranke l_{ij} der Kanten des Kreises ungleich 0, so kann dieser Fall nicht eintreffen. □

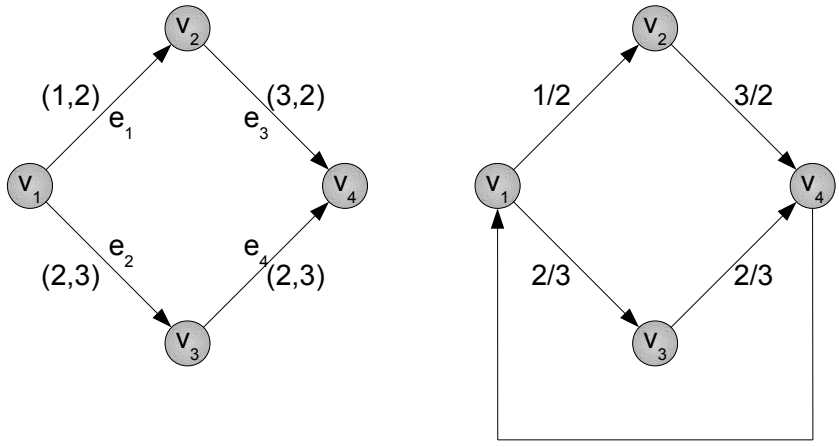
Satz 2.78.
Sind alle unteren Schranken einer (AF)-Instanz mit nichtleerer zulässiger Lösungsmenge positiv, so gibt es mindestens einen Knoten, der nur ausgehende Kanten besitzt, und mindestens einen Knoten, der nur eingehende Kanten besitzt.

Beweis. Angenommen, jeder Knoten, der eine eingehende Kante besitzt, besitzt auch eine ausgehende. Sei v_1 ein beliebiger Knoten mit einer eingehenden Kante. Dann gibt es eine ausgehende Kante (v_1, v_2), die zu einem weiteren Knoten führt, der damit eine eingehende Kante besitzt. Somit gibt es eine weitere Kante (v_2, v_3) und da der Graph endlich ist, muss ein bereits verwendeter Knoten bei Weiterverfolgung der Kanten ein zweites Mal besucht werden. Somit wäre ein gerichteter Kreis geschlossen, was Satz 2.77 widerspricht.
Der Fall, dass jeder Knoten mit einer ausgehenden Kante auch eine eingehende besitzt, ist analog zu behandeln. □

Korollar 2.79.
Gilt $A\omega = 0$ in einer (AF)-Instanz mit nichtleerer zulässiger Lösungsmenge und positiven unteren Schranken, so ist $\omega = 0$.

Damit die Knotenregel erfüllt sein kann, müssen also zusätzliche Kanten eingefügt werden.

Beispiel 2.80. In Abbildung 2.14 (a) wird ein gerichteter Graph wiedergegeben mit Spannungen, die jeweils den ersten Eintrag einer Kante bilden, und Passagiergewichten, die den zweiten Wert ausmachen.



(a) Originaler Fahrplan. (b) Resultierendes elektrisches Netzwerk.

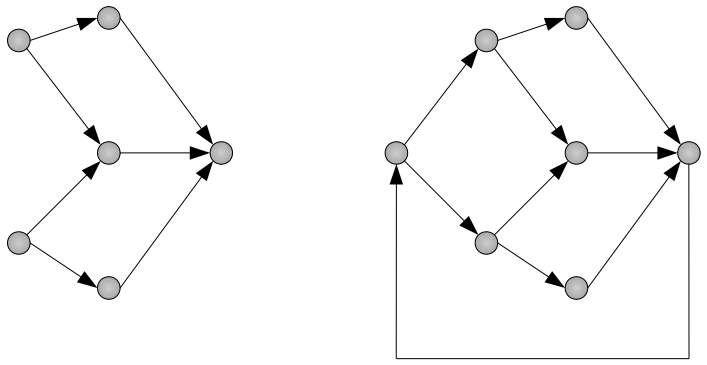
Abbildung 2.14: Erweiterung eines EANs für die Knotenregel.

Damit die Knotenregel erfüllt sein kann, wird eine zusätzliche Kante als „Abfluss“ hinzugefügt wie in (b) dargestellt. Werden für die ursprünglichen Kanten Widerstände $R_{ij} = x_{ij}/\omega_{ij}$ eingeführt, die jeweils neben die Kanten gezeichnet wurden, so ergibt das ein Problem eines elektrischen Netzwerks, nämlich:

Finde U und I, so dass die Energiedissipation minimiert wird und

$$I = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 5 \end{pmatrix} + \begin{pmatrix} 1/R_{12} & 0 & 0 & 0 & 0 \\ 0 & 1/R_{13} & 0 & 0 & 0 \\ 0 & 0 & 1/R_{24} & 0 & 0 \\ 0 & 0 & 0 & 1/R_{34} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} U.$$

Bemerkung 2.81.
In einem Graphen mit mehr als einer Quelle oder Senke lassen sich weitere Knoten und Kanten hinzufügen, um das Problem auf einen Graphen mit einer Quelle und Senke zurückzuführen. Bei mehreren Quellen wird ein Quellenknoten erzeugt, der mit jeweils einer Kante mit den Quellen verbunden ist. Ebenso wird bei mehreren Senken verfahren. Dadurch gibt es wieder genau eine Quelle und genau eine Senke, die mit einer „Abflusskante“ verbunden werden. In Abbildung 2.15 wird dieser Vorgang dargestellt.



(a) Ursprünglicher Graph. (b) Erweiterter Graph.

Abbildung 2.15: Rückführung mehrerer Quellen und Senken auf jeweils eine Quelle und Senke.

Wird das Ohmsche Gesetz in der Stromquellenform verwendet, muss entsprechend darauf geachtet werden, dass es mehrere Kanten ohne Widerstände, dafür mit Stromquellen gibt. Entlang einer Kante, die zum Beispiel von einem neu hinzugefügten Quellknoten zu einer bereits vorher vorhandenen Quelle führt, muss gerade der Strom fließen, der von dieser Kante wieder wegführt. Die Spannungen für die neu hinzugefügten Kanten sind undefiniert, was aber weder Thompsons Prinzip verletzt, da sie nicht in die Energiedissipation mit eingehen, noch im Folgenden verlangt wird.

Es wird nun also zunächst angenommen, dass es jeweils nur eine Quelle und Senke im Graphen gibt.

Satz 2.82.
Sei eine zulässige Lösung x einer (AF)-Instanz mit positiven unteren Schranken, genau einem Knoten a mit nur ausgehenden Kanten und genau einem Knoten b

mit nur eingehenden Kanten gegeben. Sei für alle anderen Knoten die Summe der ein- und ausgehenden Passagiergewichte gleich 0.
Im Graphen, der um eine Kante $|E|+1$ *von* b *nach* a *erweitert wird, ist dann ist* $I_i = \omega_i$ *für* $i \in E$ *und* $I_{|E|+1} = y$ *die Lösung mit minimaler Energiedissipation für das elektrische Netzwerk mit Widerständen* $R_i = x_i/\omega_i$ *und einer zusätzlichen Kante* (a, b)*, das das Ohmsche Gesetz*

$$\begin{pmatrix} I_1 \\ \vdots \\ I_{|E|+1} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ y \end{pmatrix} + \begin{pmatrix} 1/R_1 & & & \\ & 1/R_2 & & 0 \\ & & \ddots & \\ & 0 & & 1/R_{|E|} \\ & & & & 0 \end{pmatrix} \begin{pmatrix} U_1 \\ \vdots \\ U_{|E|+1} \end{pmatrix}$$

erfüllt, wobei y *die Summe aller vom Knoten* a *ausgehenden Passagiergewichte ist.*

Beweis. Es muss gezeigt werden, dass die beiden Kirchhoffschen Gesetze und das Ohmsche Gesetz erfüllt werden. Nach Thompsons Prinzip (Satz 2.74) wird dann die Energiedissipation minimiert.

- Das Ohmsche Gesetz. Für Kanten $i \in E$ ist $RI = U$ gefordert. Einsetzen gibt $x_i/\omega_i \cdot \omega_i = x_i$. Für die zusätzliche Kante durch das Ohmsche Gesetz kein Zusammenhang der Größen gefordert.

- Knotenregel. Nach Vorausstzung ist die Summe der eingehenden und ausgehenden Passagiergewichte für Kanten $i \in E$ immer 0. Die zusätzliche Kante ist so konstruiert, dass der eingehende Fluss in b nach a transportiert wird, wodurch auch für a und b die Knotenregel gilt.

- Maschenregel. Da x eine Lösung eines aperiodischen Fahrplanproblems ist, muss x insbesondere die Bedingung $Bx = 0$ und damit die Maschenregel erfüllen.

□

Dieser Satz zeigt, wie aus einer zulässigen Lösung eines (AF)-Problems eine optimale Lösung eines linearen elektrischen Netzwerks gewonnen wird. Das lässt sich auch umgekehrt betrachten.

Satz 2.83.
Sei ein gerichteter Graph $G(V, E)$ *mit genau einem Knoten* a *mit nur eingehenden Kanten und genau einem Knoten* b *mit ausgehenden Kanten gegeben. Es wird dieser Graph um eine Kante* $|E| + 1$*, die von* b *nach* a *geht, erweitert, und ein Ohmsches Gesetz der Form*

$$\begin{pmatrix} I_1 \\ \vdots \\ I_{|E|+1} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ y \end{pmatrix} + \begin{pmatrix} 1/R_1 & & & \\ & 1/R_2 & & 0 \\ & & \ddots & \\ & 0 & & 1/R_{|E|} \\ & & & & 0 \end{pmatrix} \begin{pmatrix} U_1 \\ \vdots \\ U_{|E|+1} \end{pmatrix}$$

gefordert. Sind I *und* U *Stromstärke und Spannung, so dass dieses Gesetz erfüllt und die Energiedissipation minimiert wird, dann ist* $x_i = U_i$ *für* $i \neq |E| + 1$ *die*

optimale Lösung eines (AF)-Problems

$$\begin{aligned} \min \quad & \sum_{i \in E} x_i \omega_i \\ \textit{so dass} \quad & Bx = 0 \\ & l_i := R_i I_i \leq x_i \leq u_i \qquad \forall i \in E \end{aligned}$$

mit EAN $G(V, E)$, Passagiergewichten $w_i = I_i$ und beliebigen oberen Schranken u_i.

Beweis. Nach Thompsons Prinzip (Satz 2.74) erfüllt $x_i = U_i$ die Bedingung $Bx = 0$. Durch das Ohmsche Gesetz ist zudem $R_i I_i = U_i$ und somit $x_i = l_i$. Daher ist x zulässig.
Da für jede andere Lösung x' aus den unteren Schranken $x_i \leq x'_i$ für jede Kante $i \in E$ folgt, ist x zudem optimale Lösung des aperiodischen Problems. □

Sollen die Freiheitsgrade dagegen in beide Richtungen erhalten bleiben, also nicht eine *fixe* Lösung in ein Problem umgewandelt werden, so müssen entsprechend die Widerstände des elektrischen Netzwerks variabel sein. Das wird durch das Problem (W) ausgedrückt.

(W).
Sei ein Graph $G(V, E)$ mit Schranken LR_{ij}, UR_{ij} gegeben. Seien ferner zwei Knoten a, b fixiert. Das Problem der Suche nach optimalen Widerständen (W) besteht dann in

$$\begin{aligned} \min_R \min_I \quad & \sum_{(i,j) \in E} I_{ij}^2 R_{ij} \\ \text{so dass} \quad & AI = 0 \\ & R_{ab} = 0 \\ & I_{ab} = 1 \\ & LR_{ij} \leq R_{ij} \leq UR_{ij}. \end{aligned}$$

Beispiel 2.84.
Es wird erneut der Graph aus Abbildung 2.14 (b) verwendet. Eine mögliche Instanz für (W) ist dann

$$\begin{aligned} \min_R \min_I \quad & \sum_{(i,j) \in E} I_{ij}^2 R_{ij} \\ \text{so dass} \quad & AI = 0 \\ & R_{41} = 0 \\ & I_{41} = 1 \\ & 1 \leq R_{12} \leq 3 \\ & 2 \leq R_{13} \leq 4 \\ & 1 \leq R_{24} \leq 5 \\ & 2 \leq R_{34} \leq 2. \end{aligned}$$

Werden nun Werte für R fixiert, zum Beispiel

$$\begin{aligned} R_{12} &= 2 \qquad R_{13} = 4 \\ R_{24} &= 3 \qquad R_{34} = 2, \end{aligned}$$

so entsteht daraus das Teilproblem

$$\begin{aligned} \min_{I} \quad & \sum_{(i,j)\in E} I_{ij}^2 R_{ij} \\ \text{so dass} \quad & AI = 0 \\ & I_{41} = 1, \end{aligned}$$

dass der Minimierung der Energiedissipation in einem elektrischen Netzwerk entspricht. Für alle zulässigen Werte R besitzt folglich die optimale Lösung des entstehenden Teilproblems die Eigenschaft, dass die Spannungen, die durch das Ohmsche Gesetz berechnet werden können, die zweite Kirchhoffsche Regel erfüllen.

Anhand dessen ergibt sich der folgende Zusammenhang.

Satz 2.85.
Sei ein aperiodisches Fahrplanproblem (AF) auf $G(V, E)$

$$\begin{aligned} \min \quad & \sum_{i\in E} w_i x_i \\ \textit{so dass} \quad & l_i \le x_i \le u_i \qquad \forall i \in E \\ & B_f x = 0 \end{aligned}$$

gegeben. Es gebe ferner genau einen Knoten a mit nur eingehenden Kanten und genau einen Knoten b mit nur ausgehenden Kanten. Für alle Knoten außer a und b sei die Summe der eingehenden Passagiergewichte gleich der Summe der ausgehenden Passagiergewichte. Sei y die Summe der ausgehenden Passagiergewichte im Knoten a.
Dann ist x genau dann optimale Lösung von (AF), wenn $R_i = x_i \cdot y/\omega_i$ und $I_i = \omega_i/y$ für $i \neq (a, b)$ und $R_i = 0$ und $I_i = 1$ für $i = (a, b)$ optimale Lösung eines Widerstandsproblems (W) auf dem erweiterten Graphen $G'(V, E \cup \{(a, b)\})$ mit

$$\begin{aligned} LR_i &= l_i \cdot y/\omega_i \\ UR_i &= u_i \cdot y/\omega_i \end{aligned}$$

ist.

Beweis.

- „⇒“. Sei x die optimale Lösung zu (AF) und R, I die daraus abgeleitete Lösung zu (W). Es ist $I_i = \omega_i/y$ und nach Voraussetzung erfüllen die Passagiergewichte die Knotenregel, weshalb auch $AI = 0$ ist. Daher ist die erzeugte Lösung R, I zulässig.
  Angenommen, es gibt eine optimale Lösung R', I', die von R, I verschieden ist. Dann lässt sich durch $x'_i := R'_i \cdot I'_i$ eine Lösung zu (AF) erzeugen. Es erfüllt x' die Bedingung $Bx' = 0$, da nach Thompsons Prinzip der die Energiedissipation minimierende Stromfluss eine Spannung durch das Ohmsche Gesetz erzeugt, die das zweite Kirchhoffsche Gesetz erfüllt. Die Bedingung $l_i \le x'_i \le u_i$ ist per Konstruktion ebenfals erfüllt. Somit gibt es

eine Lösung x' mit Zielfunktionswert

$$\begin{aligned}\sum_{i\in E} x'_i\omega_i &= \sum_{i\in E}(R'_i \cdot I'_i)\cdot(I'_i \cdot y)\\ &= y\cdot\sum_{i\in E} R'_i(I'_i)^2\\ &< y\cdot\sum_{i\in E} R_i I_i^2\\ &= y\cdot\sum_{i\in E}(x_i\cdot y/\omega_i)\cdot(\omega_i/y)^2\\ &= \sum_{i\in E} x_i\omega_i\end{aligned}$$

was der Optimalität von x widerspricht. Es ist also R, I bereits optimal für (W).

- „$\Leftarrow$“. Sei nun eine optimale Lösung R, I zu (W) gegeben und eine Lösung x zu (AF) daraus konstruiert. Dann ist x zulässig, wie bereits gezeigt wurde. Angenommen, es gäbe eine optimale Lösung x', die von x verschieden ist. Dann lässt sich wiederum eine Lösung R', I' herleiten, die aufgrund der Optimalität die beiden Kirchhoffschen Gesetze erfüllt. Da die Lösung eines elektrischen Netzwerks eindeutig ist, folgt $R' = R$ und $I' = I$.

□

Kapitel 3

Periodische Fahrplangestaltung

3.1 Problemstellung

In der *periodischen* Fahrplangestaltung wird nicht mehr ein einzelnes Zeitintervall betrachtet, für das der optimale Fahrplan berechnet werden soll, sondern es wird eine zyklische Wiederholung gefordert. Das Problem wird damit schwieriger, da zwei Ereignisse nicht mehr zeitlich geordnet werden können. In diesem Abschnitt wird die neue Problemformulierung erarbeitet und werden grundlegende Eigenschaften festgestellt.

Für den Kunden eines Verkehrsunternehmens ist es von großem Komfort, wenn sich Fahrzeiten regelmäßig wiederholen. Fährt zum Beispiel ein Zug immer 15 und 45 Minuten nach jeder vollen Stunde ab, so braucht sich der Fahrgast angenehmerweise bloß diese beiden Zahlen zu merken. Entsprechend sind solch *periodische* Fahrpläne beliebte und gängige Praxis.
Das *periodic event sheduling problem* (PESP), das 1989 von Paolo Serafini und Walter Ukovich im allgemeineren Zusammenhang in (SU89) eingeführt wurde, trägt dieser Periodizität Rechnung. Es wird zunächst die ursprüngliche Form des PESP eingeführt.

Definition 3.1. (SU89)
Ein *periodisches Ereignis* ϵ ist eine abzählbar unendliche Menge an Ereignissen e_p, $p \in \mathbb{Z}$, mit zugeordneten Zeiten $t(e_p) \in \mathbb{R}$, so dass für jedes p gilt $t(e_p) - t(e_{p-1}) = T$. T wird die *Periode* genannt und e_p das $p$te Auftreten der periodischen Eregnisses ϵ.

Definition 3.2. (SU89)
Ein periodisches Ereignis ϵ heißt geplant, wenn ihm ein Element $\tau(\epsilon) \in \Phi := \mathbb{R}/T$ zugeordnet wird, so dass $\tau(\epsilon) = \pi \cdot t(e_p)$ für alle p, wobei π die kanonische Projektion von $\mathbb{R}$ in die abelsche Gruppe Φ ist.

Definition 3.3. (SU89)
Sei $D = [d^-, d^+]$ ein reelles, geschlossenes Intervall. Dann ist eine Spanne Δ von Φ defininiert durch $\Delta := \pi \cdot D$. Sie heißt *echt*, wenn $\Delta \neq \Phi$ und $\Delta \neq \emptyset$.
Eine *Spannungsbedingung* a besteht aus einem geordneten Paar periodischer

Ereignisse $(\epsilon^-(a), \epsilon^+(a))$ mit einer echten Spanne $\Delta(a)$ und heißt erfüllt, wenn $\tau(\epsilon^+(a)) - \tau(\epsilon^-(a)) \in \Delta(a)$.

Mittels dieser Definitionen kann nun das *periodic event scheduling problem* folgendermaßen eingeführt werden:

(PESP) (SU89)
Sei eine endliche Menge $\mathcal{E}$ Ereignissen mit gemeinsamer Periode T und eine endliche Menge A von Spannungsbedingungen gegeben. Gesucht ist ein Plan $\tau(\epsilon) \in \Phi$ für alle $\epsilon \in \mathcal{E}$, der alle Spannungsbedingungen $a \in A$ erfüllt.

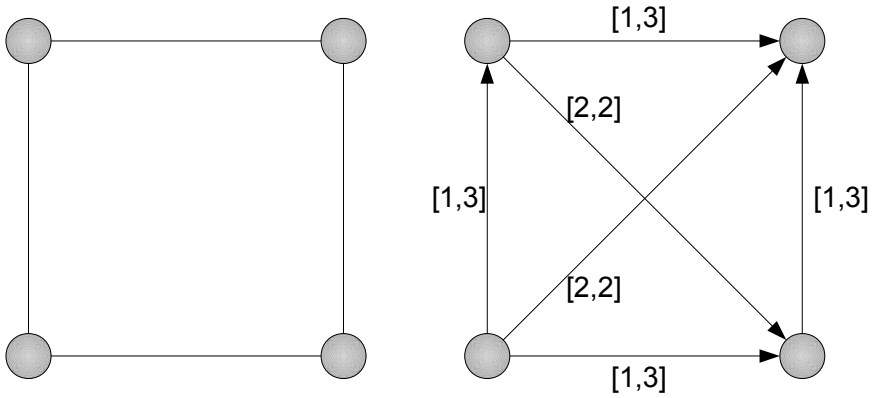
Im (PESP) lautet die Herausforderung also, *irgendeinen* Plan zu finden. Doch schon dieses Problem ist NP-schwer.

Satz 3.4. *(SU89)*
(PESP) ist NP-schwer.

Beweis. (SU89) Sei $G(V, E)$ ein ungerichteter Graph mit n Knoten, für den ein Hamiltonkreis gesucht wird. Sei E' die entsprechende Kantenmenge mit einer beliebigen Orientierung. Sei weiterhin K_n der vollständige Graph mit n Knoten und F' seine Kantenmenge mit beliebiger Orientierung. Erzeuge nun eine PESP-Instanz mit

$$T = n,\ \mathcal{E} = V,\ \mathcal{A} = F'$$

$$\Delta(a) = \begin{cases} [1, n-1], & \text{falls } a \in E', \\ [2, n-2] & \text{sonst.} \end{cases}$$



(a) Hamiltonkreis-Instanz. (b) PESP-Instanz.

Das so gewonnene PESP hat genau dann eine Lösung, wenn G einen Hamiltonkreis besitzt. Angenommen, eine Lösung $\tau(\epsilon)$ für das PESP ist bekannt, dann ist $\tau(\epsilon') - \tau(\epsilon'') \bmod n \geq 1$ für alle $\epsilon', \epsilon'' \in \mathcal{E}$, $\epsilon' \neq \epsilon''$, und es gibt eine zyklische Permutation P auf $\mathcal{E}$, so dass $\tau(P(\epsilon)) - \tau(\epsilon) \bmod n = 1$, wobei ϵ und $P(\epsilon)$ benachbart sind in G.
Sei andererseits eine zyklische Permutation P gegeben, die zu einem Hamiltonkreis gehörig ist, so wird eine Lösung zum PESP gewonnen, indem $\tau(\epsilon_1) = 0$ und $\tau(P(\epsilon_i)) = \tau(\epsilon_i) + 1$ gesetzt wird. □

In der periodischen Fahrplangestaltung ist das Ziel, wie im aperiodischen Fall, nicht alleine die Suche nach einem beliebigen zulässigen Fahrplan, sondern sogar nach einem *optimalen* gemäß einer gegebenen Menge an Kantengewichten. Als Grundlage zur Formulierung wird das (PESP) verwendet und an die speziellen Bedürfnisse angepasst.

(PF) (NO08)
Gegeben sei ein EAN $G(\mathcal{E}, \mathcal{A})$ mit Kantengewichten $\omega_{ij} \in \mathbb{N}$ und unteren und oberen Schranken $l_{ij}, u_{ij} \in \mathbb{N}$ für jede Kante $(i,j) \in \mathcal{A}$, sowie eine Periode T. Gesucht ist ein kostenminimaler periodischer Fahrplan π, das heißt

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in\mathcal{A}} \omega_{ij}(\pi_j - \pi_i + z_{ij}T) && && (3.5)\\
\text{so dass} \quad & l_{ij} \le \pi_j - \pi_i + z_{ij}T \le u_{ij} && \forall (i,j) \in \mathcal{A}, && (3.6)\\
& \pi_i \in \mathbb{R} && \forall i \in \mathcal{E}, && (3.7)\\
& z_{ij} \in \mathbb{Z} && \forall (i,j) \in \mathcal{A}. && (3.8)
\end{aligned}$$

Da in der periodischen Fahrplangestaltung ein Ereignis, das zum Zeitpunkt t_0 statt findet, das äquivalent auch zu den Zeitpunkten $\ldots, t_0 - T, t_0, t_0 + T, \ldots$ tut, wird als Schreibweise die *Modulo-Klammer*

$$\begin{aligned}
[\cdot]_T : \quad \mathbb{Z} &\longrightarrow \mathbb{Z}/T\\
t &\longmapsto t \bmod T
\end{aligned}$$

und die *Modulo-Gleichheit*

$$a =_T b :\Leftrightarrow [a]_T = [b]_T$$

eingeführt. Damit lässt sich das Problem (PF) analog zum aperiodischen Gegenstück (AF) auch mittels einer Fundamentalkreismatrix umformulieren. Der Fluss entlang eines Kreises muss sich nun nicht mehr auf 0 summieren, sondern es reicht aus, wenn er das in Bezug auf Modulo-Gleichheit tut. Um die Bedingung $B_f x =_T 0$ als lineares Programm zu formulieren, müssen allerdings wieder Modulo-Parameter z verwendet werden. Offenbar sind sie diesmal aber nicht für jede Kante, sondern nur für jeden Fundamentalkreis notwendig.

(PF') (NO08)
Gegeben sei ein EAN $G(\mathcal{E}, \mathcal{A})$ mit Kantengewichten $\omega_{ij} \in \mathbb{N}$ und unteren und oberen Schranken $l_{ij}, u_{ij} \in \mathbb{N}$ für jede Kante $(i,j) \in \mathcal{A}$, sowie eine Periode T. Sei B_f die Fundamentalkreismatrix bezüglich eines spannenden Baumes $\mathcal{T}$ mit Kantenmenge $\mathcal{A}_\mathcal{T}$. Gesucht ist ein kostenminimaler periodischer Fahrplan, das heißt

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in\mathcal{A}} \omega_{ij}x_{ij} && && (3.9)\\
\text{so dass} \quad & l_{ij} \le x_{ij} \le u_{ij} && \forall (i,j) \in \mathcal{A}, && (3.10)\\
& B_f x = Tz && && (3.11)\\
& x_{ij} \in \mathbb{R} && \forall (i,j) \in \mathcal{A}, && (3.12)\\
& z_{ij} \in \mathbb{Z} && \forall (i,j) \in \mathcal{A} \setminus \mathcal{A}_\mathcal{T}. && (3.13)
\end{aligned}$$

Es lässt sich also ablesen, dass für die Kanten des spannenden Baumes keine Modulo-Parameter nötig sind. In der Formulierung (PF) dürfen sie deswegen auf 0 gesetzt werden.

Korollar 3.14. *Modulo-Parameter spannender Bäume*
Sei eine zulässiger Fahrplan (π, z) einer (PF)-Probleminstanz und ein spannender Baum $\mathcal{T}$ im dazugehörigen EAN gegeben. Dann gibt es eine zulässige Lösung zu (PF) mit gleichem Zielfunktionswert, für die $z_{ij} = 0$ für alle Kanten (i, j) des spannenden Baumes gilt.

Es stellt sich heraus, dass in der Formulierung (PF) nicht alle ganzen Zahlen als Suchraum für die Modulo-Parameter nötig ist.

Bemerkung 3.15.
Es lässt sich ohne Beschränkung der Allgemeingültigkeit für ein EAN $0 \leq l_{ij} \leq T - 1$ annehmen. Ist in einer Probleminstanz eine untere Schranke l_{ij} größer als T, so ändert sich nur der Zielfunktionswert um eine Konstante, wird statt dessen $[l_{ij}]_T$ verwendet.

Satz 3.16. *Einschränkung der Modulo-Paramter*
Sei ein zulässiger Fahrplan (π, z) einer (PF)-Probleminstanz mit $u_{ij} - l_{ij} \leq T-1$ für alle Kanten und $\pi_i \in [0, T-1]$ für alle Knoten des EANs gegeben. Es sei ohne Beschränkung der Allgemeingültigkeit $0 \leq l_i j \leq T - 1$. Dann gilt $z_{ij} \in \{0, 1, 2\}$ für alle Kanten des EANs.

Beweis. Nach Voraussetzung gilt

$$l_{ij} \leq \pi_j - \pi_i + z_{ij}T \leq u_{ij} \tag{3.17}$$
$$\Leftrightarrow \quad l_{ij} - \pi_j + \pi_i \leq z_{ij}T \leq u_{ij} - \pi_j + \pi_i \tag{3.18}$$

und

$$\pi_i - \pi_j \in [-T + 1, T - 1] \tag{3.19}$$
$$l_{ij} \in [0, T - 1] \tag{3.20}$$
$$u_{ij} \in [0, 2T - 2]. \tag{3.21}$$

Es folgt

$$l_{ij} - \pi_j + \pi_i \in [-T + 1, 2T - 2] \tag{3.22}$$
$$u_{ij} - \pi_j + \pi_i \in [-T + 1, 3T - 3]. \tag{3.23}$$

Somit muss $-T + 1 \leq z_{ij}T \leq 3T - 3$ gelten und daher $z_{ij} \in \{0, 1, 2\}$. □

Bemerkung 3.24.
Gilt in den Voraussetzungen des vorherigen Satzes sogar $u_{ij} \leq T - 1$, so ist es ausreichend, binäre Modulo-Parameter zu verwenden.

Bemerkung 3.25.
Die Voraussetzungen von Korollar 3.14 und Satz 3.16 lassen sich nicht immer kombinieren; das heißt, wird für einen spannenden Baum erzwungen, dass die Modulo-Paramter auf 0 gesetzt sind, kann nicht mehr davon ausgegangen werden, dass alle Modulo-Paramter in der Menge $\{0, 1, 2\}$ liegen.

Beispiel 3.26.
Betrachte das EAN aus Abbildung 3.1 mit Periode 10.

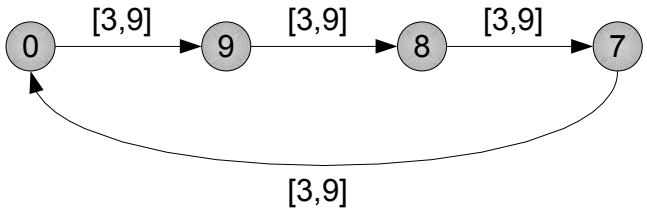


Abbildung 3.1: Beispielinstanz für (PF).

Die Intervalle an den Kanten bezeichnen untere und obere Schranken, die Zahlen in den Knoten einen gegebenen Fahrplan. In diesem Fall ist ein Modulo-Parameter auf 0 gesetzt und die weiteren drei auf 1. Angenommen, es werden nun als spannender Baum diejenigen Kanten gewählt, die in der Zeichnung von links nach rechts verlaufen. Die Spannungen sollen erhalten bleiben und die Modulo-Parameter für diese Kanten auf 0 gesetzt werden. Der resultierende Fahrplan ist in Abbildung 3.2 dargestellt.

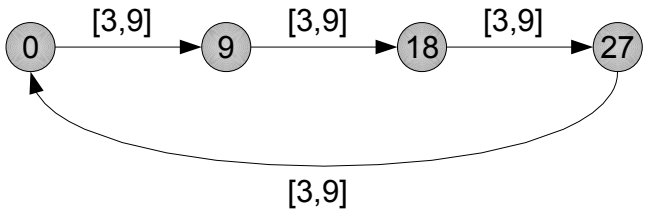


Abbildung 3.2: Veränderte Beispielinstanz für (PF).

Der Modulo-Parameter der einzigen Nichtbaumkante ist nun 3 und die Voraussetzung $\pi_i \in [0, T-1]$ nicht mehr gegeben.

Bei Betrachtung der Zielfunktion

$$f(x) = \sum_{(i,j)\in\mathcal{A}} \omega_{ij} x_{ij} = \sum_{(i,j)\in\mathcal{A}} \omega_{ij}(y_{ij} + l_{ij})$$

fällt auf, dass auf jeder Kante die untere Schranke l_{ij} in die Gewichtung mit eingeht. Da sie gegeben sind und nicht variabel, bilden sie eine Konstante, die für die Optimierung irrelevant ist:

$$f(x) = \sum_{(i,j)\in\mathcal{A}} \omega_{ij} y_{ij} + \sum_{(i,j)\in\mathcal{A}} \omega_{ij} l_{ij}.$$

Die Variablen y_{ij} werden *Slack* (frei übersetzt: „Verzögerung“) genannt und stellen die Verzögerung dar, die über die Minimalzeit hinaus geht. Den Slack zu minimieren ist also äquivalent zur Minimierung der Spannung.

Bemerkung 3.27.
Als Erweiterung des PESP führen Serafini und Ukovich in (SU89) das *Extended Periodic Event Scheduling Problem* ein, das verschiedene Perioden erlaubt. Da sich zum Beispiel nicht alle Züge zur vollen Stunde wiederholen, sondern auch alle

20 oder 30 Minuten, ist das eine Eigenschaft, die ein in der Praxis verwendbares Modell widerspiegeln muss. Karl Nachtigall gibt in (Nac98) ein Verfahren an, wie sich in diesem Fall der Suchbereich für Modulo-Parameter einschränken lässt. Wie in Bemerkung 3.25 lassen sich dann allerdings nicht mehr die Modulo-Parameter spannender Bäume auf 0 setzen, was im Folgenden benötigt wird.
Der bessere Weg, verschiedene Perioden handzuhaben, liegt darin, das kleinste gemeinsame Vielfache zu verwenden und Ereignisse im EAN entsprechend oft zu konstruieren. Kanten mit fixierter Spannung erzwingen dann, dass zwei Ereignisse den Abstand gemäß ihrer ursprünglichen Periode einhalten. Ist das kleinste gemeinsame Vielfache aller Perioden zum Beispiel 60, so wird ein Ereignis mit Periode 20 verdreifacht und durch Kanten mit unterer und oberer Schranke von 20 verbunden.

3.2 Der Modulo-Simplex Algorithmus

In Anlehnung an den Netzwerk-Simplex-Algorithmus wurde eine Anpassung an das periodische Fahrplanproblem von Karl Nachtigall und Jens Opitz in (NO08) vorgeschlagen. Es wird diese Algorithmusadaption präsentiert und theoretisch analysiert.

3.2.1 Aufbau

Nachtigall und Opitz präsentieren in (NO08) einen Algorithmus, der in Anlehnung an den klassischen Netzwerk-Simplex-Algorithmus eine Anfangslösung durch sukzessive Anwendung von Schnitten verbessert. Folgende Betrachtungen motivieren diesen Ansatz:

Satz 3.28.
Sei G ein zusammenhängender, gerichteter Graph und $\mathcal{T}$ ein spannender Baum in G. Sei ferner ein Vektor $x \in \mathbb{R}^{|E|}$ mit $B_f x = 0$ gegeben. Dann gibt es ein $y \in \mathbb{R}^{|V|-1}$, so dass gilt

$$y^T Q_f = x^T.$$

Beweis. Sei

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},$$

wobei ohne Beschränkung der Allgemeingültigkeit x_2 der den Baumkanten zugeordnete Teil des Vektors x bezeichne. Setze $y := x_2$. Da Q_f dann der Form (A, Id) ist, gilt

$$y^T Q_f = \begin{pmatrix} z \\ x_2 \end{pmatrix}^T$$

für ein z. Andererseits ist nach Korollar 2.69

$$B_f (y^T Q_f)^T = B_f Q_f^T y = 0.$$

Da B_f der Form (Id, A^*) ist, folgt

$$Idz = -A^* x_2$$

und z ist eindeutlig gegeben. Andererseits ist nach Konstruktion auch

$$Id x_1 = -A^* x_2,$$

weshalb $x_1 = z$ und die Behauptung folgt. □

Bemerkung 3.29.
Fordert man im vorherigen Satz allgemeiner $B_f x =_T 0$ statt $B_f x = 0$, so bleibt er gültig, falls bekannt ist, dass die Gleichung $x_i + z =_T 0$ für alle Kanten weiterhin höchstens eine Lösung besitzt. Das ist der Fall, wenn x_i in einem Intervall von kleinerer Länge als T liegt.

Bemerkung 3.30.
Ein Vektor $y \in \mathbb{R}^{|V|-1}$ wird *Linearkombination von Schnitten* genannt. Anschaulich wird jedem Fundamentalschnitt eine reelle Zahl zugeordnet, die angibt, um wieviele Einheiten entlang der jeweils zu dem Schnitt gehörenden Kanten die Spannung verändert wird. Es ist zu beachten, dass damit keine Linearkombination im Sinne von Vektorräumen gemeint ist - nicht einmal die Vereinigung zweier Schnitte ergibt zwangsweise wieder einen Schnitt.

Korollar 3.31.
Sei eine Probleminstanz (PF') mit zulässigen Lösungen x und x' gegeben. Sei ohne Beschränkung der Allgemeinheit $u_{ij} - l_{ij} \leq T - 1$ für alle Kanten $(i,j) \in \mathcal{A}$. Dann lässt sich x' aus x durch eine Linearkombination von Schnitten erzeugen, das heißt, es gibt ein $y \in \mathbb{R}^{|V|-1}$, so dass

$$x' = x + y^T Q_f.$$

Beweis. Es ist $B_f x =_T B_f x' =_T 0$ und daher $B_f(x' - x) =_T 0$. Nach Satz 3.28 und Bemerkung 3.29 folgt die Behauptung. □

Korollar 3.32.
Von jeder zulässigen Lösung für (PF') aus ist das globale Optimum durch eine einzige Linearkombination von Schnitten erreichbar.

Andererseits wird durch die Verwendung von Schnitten keine Kreisbedingung gestört. Anschaulich ist das leicht zu verstehen, wird bedacht, dass ein Schnitt den Graphen in zwei Teile „zerschneidet". Summiert sich die Spannung zwischen diesen Teilen auf 0, so gilt das auch für jeden Kreis, der diese Teile verbindet.

Satz 3.33.
Sei $y^T Q_f = x^T$. Dann gilt $B_f x = 0$.

Beweis. Nach Korollar 2.69 ist

$$B_f x = B_f (y^T Q_f)^T = B_f Q_f^T y = 0.$$

□

Die Idee des Modulo-Netzwerk-Simplex Verfahrens besteht nun darin, analog zum „klassischen" Netzwerk-Simplex Verfahrens eine bestehende Lösung durch einen spannenden Baum zu codieren und jeden Schnitt als *Änderung des Baumes* aufzufassen. Daher werden Spannender-Baum-Strukturen *im Sinne des Modulo-Netzwerk-Simplex Verfahrens* definiert:

Definition 3.34. Spannender-Baum-Struktur (NO08)
Eine *Spannender-Baum-Struktur* $(\mathcal{T}_l, \mathcal{T}_u)$ ist ein spannender Baum $\mathcal{T} = \mathcal{T}_l \cup \mathcal{T}_u$, dessen Kantenmenge so partitioniert ist, dass die Spannung x_{ij} auf den Kanten $(i,j) \in \mathcal{T}_l$ gleich l_{ij} und auf den Kanten $(i,j) \in \mathcal{T}_u$ gleich u_{ij} ist.

Definition 3.35.
Eine zulässige Lösung (π, z) eines Problems (PF) oder x eines Problems (PF') heißt *einer Spannender-Baum-Struktur $(\mathcal{T}_l, \mathcal{T}_u)$ zugeordnet*, wenn $\pi_j - \pi_i = l_{ij}$, bzw. $x_{ij} = l_{ij}$, für alle Kanten $(i,j) \in \mathcal{T}_l$ und $\pi_j - \pi_i = u_{ij}$, bzw. $x_{ij} = u_{ij}$, für alle Kanten $(i,j) \in \mathcal{T}_u$ gilt.

Bemerkung 3.36.

- Dadurch sind bereits alle Potentiale und die Spannungen entlang der restlichen Kanten festgelegt.
- Es werden also die Modulo-Parameter des spannenden Baumes auf 0 fixiert. Alle fehlenden Modulo-Parameter sind aus der Lösung berechenbar und insofern in der Spannender-Baum-Struktur bereits codiert.

Satz 3.37.
Sei (π^*, z^*) die optimale Lösung eines periodischen Fahrplanproblems (PF). Dann gibt es eine einer Spannender-Baum-Struktur zugeordnete zulässige Lösung mit gleichem Zielfunktionswert.

Beweis. Für fixe Modulo-Parameter z^* ist (PF) äquivalent zu einem aperiodischen Fahrplanproblem. Mittels des klassischen Netzwerk-Simplex Algorithmus kann das Duale dieses Problems gelöst werden und die Lösung ist als spannender Baum im erweitertem EAN gegeben. Kanten dieses spannenden Baumes, deren Orientierung der des ursprünglichen Netzwerkes entspricht, sollen nun zu $\mathcal{T}_u$ und Kanten entgegengesetzter Orientierung zu $\mathcal{T}_l$ gehören.
Die Spannender-Baum-Struktur $(\mathcal{T}_l, \mathcal{T}_u)$ ist dann optimal bezüglich der Modulo-Parameter z^*, da sie genau die in Satz 2.47 geforderten optimalen Knotenpotentiale erzeugt. Da das globale Optimum von (PF) in der Menge der zulässigen Lösungen mit Modulo-Parametern z^* liegt, müssen die Zielfunktionswerte gleich sein. □

Dies Überlegung wird vom folgenden Satz noch verallgemeinert:

Satz 3.38. *(Nac98)*

$$\begin{pmatrix}\pi\\ z\end{pmatrix} \in \mathcal{Q} := \mathit{conv.hull}\left(\left\{\begin{pmatrix}\pi\\ z\end{pmatrix} \mid l_{ij} \le \pi_j - \pi_i + T z_{ij} \le u_{ij}; z \in \mathbb{Z}^m; \pi \in \mathbb{R}^n\right\}\right)$$

ist ein Extrempunkt von $\mathcal{Q}$ genau dann, wenn es eine Spannender-Baum-Struktur-Lösung ist.

Es ist also wie im klassischen Netzwerk-Simplex Algorithmus ausreichend, lediglich Spannender-Baum-Struktur-Lösungen zu betrachten. Ist eine solche Struktur bereits vorhanden, wird iterativ pivotisiert, das heißt eine Kante aus dem Baum entfernt und eine andere dafür hinzugefügt, bis ein lokales Optimum erreicht worden ist.

Satz 3.39.
Sei ein spannender Baum in einem Graphen $G(V, E)$ gegeben. Eine Nichtbaumkante $i = (a, b)$ ist genau dann in dem durch eine Baumkante $j = (c, d)$ induzierten Fundamentalschnitt enthalten, wenn j in dem durch i induzierten Fundamentalkreis liegt.

Beweis.

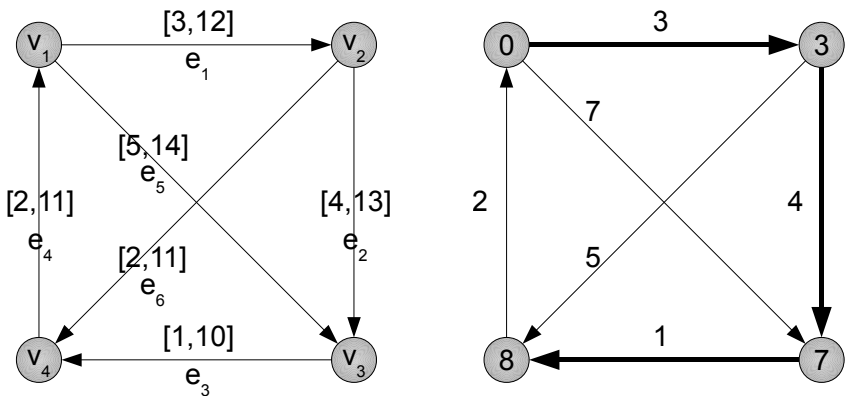
- „⇒“. Es sei zunächst i im durch j induzierten Fundamentalschnitt enthalten, der durch die Partition $V_1 \cup V_2$ mit $c \in V_1$ und $d \in V_2$ gegeben ist. Dann ist entweder $a \in V_1$ und $b \in V_2$ oder $a \in V_2$ und $b \in V_1$. In beiden Fällen führt der Pfad im spannenden Baum von a nach b an der Kante j vorbei, da sie die einzige Baumkante ist, die beide Partitionen verbindet. Somit ist j im durch i induzierten Fundamentalkreis enthalten.

- „⇐“. Es sei nun j im durch i induzierten Fundamentalkreis enthalten. Dann führt der Pfad im spannenden Baum von a nach b durch j. Seien V_1 diejenigen Knoten, die entlang des Pfades bis zu der Kante j besucht werden, und V_2 die folgenden Knoten. Dann ist der durch diese Partition entstehende Schnitt bis auf die Orientierung der durch die Kante j induzierte Fundamentalschnitt. Da $a \in V_1$ und $b \in V_2$, liegt auch i in diesem Schnitt.

□

Angenommen, ein Baum $\mathcal{T} = \mathcal{T}_l \cup \mathcal{T}_u$ ist die augenblickliche Lösung und eine Nichtbaumkante e_n soll als untere Baumkante für eine Baumkante e_b pivotisiert werden. Das ist nur dann möglich, wenn der durch e_n erzeugte Fundamentalkreis auch die Kante e_b enthält, da sonst durch Verwendung von e_n ein Kreis geschlossen würde. Dann kann die Pivotisierung durch den von der Baumkante erzeugten Schnitt vollzogen werden. Denn wie in Satz 3.33 bereits gezeigt, bleibt durch einen Schnitt die Bedingung $B_f x = 0$ und damit insbesondere $B_f x =_T 0$ erhalten, und es sind neben den Kanten e_b und e_n nur Nichtbaumkanten enthalten. Entlang dieses Schnittes wird die Spannung um ein y verändert, das so gewählt wird, dass die Spannung der Kante e_n auf l_{e_n} reduziert wird (beziehungsweise auf u_{e_n}, wenn sie als obere Baumkante gewünscht ist). Ob der Wert y auf eine Kante addiert oder subtrahiert wird, hängt davon ab, in welcher Orientierung sie durch den Fundamentalkreis durchlaufen wird, bzw. in welcher Orientierung sie im Fundamentalschnitt enthalten ist.
Das Prinzip dieser Pivotoperation wird durch ein Beispiel verdeutlicht.

Beispiel 3.40.
In Abbildung 3.3 (a) ist eine Probleminstanz gegeben mit Periode $T = 10$ und in 3.3 (b) eine zulässige Spannender-Baum-Struktur, wobei die fett gedruckten Kanten Teil des Baums sein sollen und $\mathcal{T} = \mathcal{T}_l$ gilt.

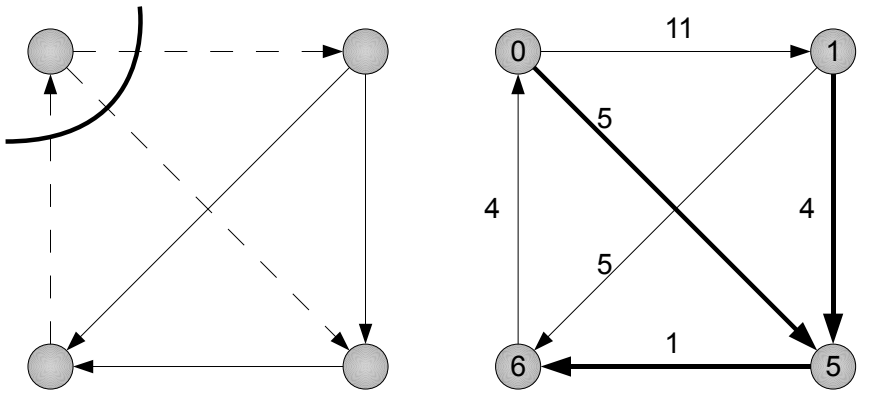


(a) Eine (AF)-Instanz. (b) Eine Spannender-Baum-Struktur-Lösung.

Abbildung 3.3: Beispiel für eine Pivotisierung (1).

Angenommen, es soll die Baumkante e_1 gegen die Nichtbaumkante e_5 getauscht werden. Das ist zulässig, da e_1 in dem durch e_5 erzeugten Fundamentalkreis liegt. Anschaulich gesprochen ist nun der Schnitt gefragt, der e_1 und e_5 beinhaltet, aber den Rest des Baumes nicht betrifft, da die bestehende Struktur

erhalten bleiben soll. Das ist genau der Fall bei dem durch die Baumkante e_1 induzierten Fundamentalschnitt, da nach Definition 2.67 in einem Fundamentalschnitt nur eine Baumkante enthalten ist (und diejenigen Nichtbaumkanten, in deren Fundamentalkreisen diese Baumkante liegt). Abbildung 3.4 (a) stellt diesen Schnitt dar.



(a) Der gewählte Schnitt.

(b) Die neue Spannender-Baum-Struktur-Lösung.

Abbildung 3.4: Beispiel für eine Pivotisierung (2).

Die Lösung x wird nun entlang des Schnittes so verändert, dass die Spannung der Kante e_5 auf 5 reduziert wird. Entsprechend der Kantenorientierungen wird also die Spannung auf e_1 um 2 verringert und auf e_4 um 2 erhöht. Abbildung 3.4 (b) stellt die neu gewonnene Lösung dar.

Bemerkung 3.41.
Auch wenn eine Baumkante in dem durch eine Nichtbaumkante induzierten Fundamentalkreis liegt, ist es möglich, dass sich eine Pivotoperation als unzulässig erweist. Das ist genau dann der Fall, wenn eine Spannung sich derart verändert, dass sie ihre obere oder untere Schranke verletzt. Im vorherigen Beispiel wurde für jede Kante $u_{ij} - l_{ij} = T - 1$ gesetzt, wodurch dieser Fall ausgeschlossen wird.

Nachtigall und Opitz schlagen in (NO08) vor, die verfügbaren Pivotoperationen in ein Tableau zu schreiben, das im Aufbau an ein Simplex-Tableau erinnert. Es besteht im wesentlichen aus der Fundamentalkreismatrix; am rechten Rand werden noch Slack und Kosten für Nichtbaumkanten hinzugefügt. Sollte $\mathcal{T}_u = \emptyset$ gelten, kann sofort der gegenwärtige Zielfunktionswert abgelesen werden, ansonsten muss der enstprechende Betrag noch hinzuaddiert werden.

Beispiel 3.42.
In Abbildung 3.5 wird das vorherige Beispiel noch einmal aufgenommen und mit Kosten versehen.

Das zu dem ersten spannenden Baum $\mathcal{T} = \mathcal{T}_l = \{e_1, e_2, e_3\}$ gehörende Modulo Simplex Tableau wird in Tabelle 3.1 widergegeben. Für das Verfahren sind die Spalten, die zu Nichtbaumkanten gehören und eine Identitätsmatrix bilden, nicht von Interesse und können ignoriert werden.

Werden die Einträge der Fundamentalkreismatrix mit b bezeichnet, lässt sich der durch die Pivotoperation, eine Baumkante j gegen eine Nichtbaumkante i

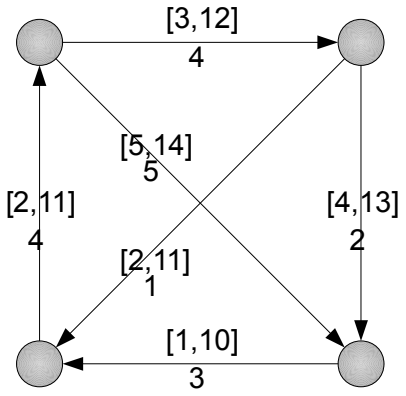


Abbildung 3.5: Erweiterung des vorherigen Beispiels.

	e_1	e_2	e_3	e_4	e_5	e_6	y	ω
e_4	1	1	1	1	0	0	0	4
e_5	-1	-1	0	0	1	0	2	5
e_6	0	-1	-1	0	0	1	3	1
ω							13	

Tabelle 3.1: Modulo Simplex Tableau zur Anfangslösung.

nach $\mathcal{T}_l$ zu tauschen, entstehende neue Zielfunktionswert berechnen durch

$$\omega_{ij} = \sum_{k \in \mathcal{A} \setminus (\mathcal{T} \cup \{i\})} \omega_k \left[y_k - \frac{b_{kj}}{b_{ij}} y_i \right]_T + \omega_j \left[\frac{y_i}{b_{ij}} + y_j \right]_T + \sum_{k \in \mathcal{T}_u} \omega_k y_k.$$

Der Wert b_{ij} gibt die Orientierung vor, entlang derer die Spannung verändert wird. Sie ließe sich natürlich auch in den Zähler schreiben, doch wird so noch einmal deutlich, dass keine Pivotisierung gewählt werden darf, wenn die Baumkante nicht im von der Nichtbaumkante induzierten Fundamentalkreis liegt. Die Summe als erster Summand der Gleichung gibt den Zielfunktionswert an, der durch Nichtbaumkanten außer i erzeugt wird. Er wird für eine Kante k genau dann durch den Slack der Kante i verändert, wenn $b_{kj} \neq 0$, also j im von k induzierten Fundamentalschnitt liegt. Der zweite Summand der Gleichung gibt den Zielfunktionswert an, der durch die Baumkante j erzeugt wird, deren Slack entsprechend angepasst werden muss. Der letzte Summand gibt den durch die Baumkanten, deren Slack auf das Maximum fixiert wird, erzeugten Zielfunktionswert, der nicht beeinflusst wird, da der Fundamentalschnitt genau eine Baumkante enthält.

Die Kostendifferenz liegt dann bei

$$\begin{aligned}\Delta\omega_{ij} &= \omega_{ij} - \omega \\ &= \sum_{k\in\mathcal{A}\setminus(\mathcal{T}\cup\{i\})} \omega_k \left[y_k - \frac{b_{kj}}{b_{ij}} y_i\right]_T + \omega_j \left[\frac{y_i}{b_{ij}} + y_j\right]_T + \sum_{k\in\mathcal{T}_u} \omega_k y_k - \sum_{k\in\mathcal{A}} \omega_k y_k \\ &= \sum_{k\in\mathcal{A}\setminus(\mathcal{T}\cup\{i\})} \omega_k \left(\left[y_k - \frac{b_{kj}}{b_{ij}} y_i\right]_T - y_k\right) + \omega_j \left(\left[\frac{y_i}{b_{ij}} + y_j\right]_T - y_j\right) - \omega_i y_i.\end{aligned}$$

Diese Formel wird in (NO08) inkorrekt widergegeben. Wird eine Nichtbaumkante dagegen nach $\mathcal{T}_u$ pivotisiert, so wird die Lösung entlang des Schnitts nicht um y_i, sondern um $y_i - (u_i - l_i)$ verändert. Entsprechend lautet die Kostendifferenz in diesem Fall:

$$\begin{aligned}\Delta\omega_{ij} &= \omega_{ij} - \omega \\ &= \sum_{k\in\mathcal{A}\setminus(\mathcal{T}\cup\{i\})} \omega_k \left(\left[y_k - \frac{b_{kj}}{b_{ij}}(y_i - u_i + l_i)\right]_T - y_k\right) \\ &\quad + \omega_j \left(\left[\frac{y_i - u_i + l_i}{b_{ij}} + y_j\right]_T - y_j\right) - \omega_i (y_i - u_i + l_i).\end{aligned}$$

Auf diese Weise lassen sich alle Kostendifferenzen für die verfügbaren Pivotoperationen berechnen.

Bemerkung 3.43.
Obwohl die Bezeichung als „Simplex-Tableau“ gewählt gewählt wurde, lassen sich durch die Modulo-Parameter leider keine reduzierten Kosten bestimmen. Es lässt sich einer Baumkante nicht „ansehen“, ob sie die Lösung verlassen sollte, ohne jede einzelne Pivotmöglichkeit zu testen. Dadurch wird das Verfahren stark verlangsamt.

Die Durchführung einer Pivotoperation verändert die Spannender-Baum-Struktur, weshalb das ModuloSimplex Tableau wieder neu aufgebaut werden muss.
Im Falle des klassischen Netzwerk-Simplex Verfahrens ist aufgrund der Konvexität das globale Optimum gefunden, sobald es keine verbessernde Pivotoperation mehr gibt. Das ist durch die Einführung der diskreten Modulo-Parameter in diesem Fall leider nicht mehr möglich. Sind alle Kostendifferenzen ≥ 0, ist leider nur ein lokales Optimum gewonnen und die ermittelte Lösung ließe sich eventuell noch weiter verbessern.
Um dieses lokale Optimum wieder verlassen zu können, wird eine weitere Menge an Schnitten verwendet, die *Single Node Cuts* („Einzelknoten-Schnitte“). Wie der Name bereits nahe legt, sind dies diejenigen Schnitte, die durch Knotenpartitionen $(E \setminus \{i\}) \cup \{i\}$ für Knoten $i \in V$ entstehen. Da es nur $|V|$ Schnitte dieser Art gibt, lassen sie sich einzeln mit allen $\delta \in (0, T-1]$ als Spannungsänderung durchgehen. Um die Laufzeit nicht unnötig zu belasten, wird der erste solche Schnitt verwendet, der die bestehende Lösung verbessert.

Abbildung 3.6 zeigt schematisch einen Single Node Cut zum Knoten i. Werden die Spannungen entlang der Kanten um ein δ verändert, verbessert sich die bestehende Lösung, wenn

$$\sum_{e_{ij}\in\mathcal{A}} \omega_{ij}(y_{ij} - \delta) + \sum_{e_{ji}\in\mathcal{A}} \omega_{ji}(y_{ji} + \delta) < 0$$

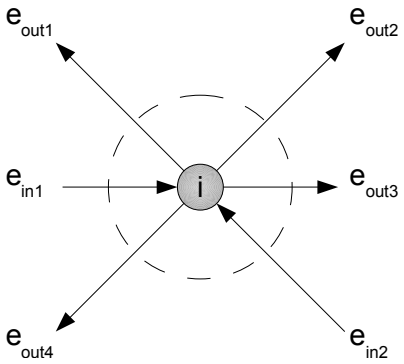


Abbildung 3.6: Ein Single Node Cut.

gilt, also der Fahrplan, der dadurch entsteht, dass Ereignis i um δ verzögert wird, den bestehenden Fahrplan verbessert. Dadurch wird allerdings die Spannender-Baum-Struktur zerstört und es muss für fixierte Modulo-Parameter der klassische Netzwerk-Simplex Algorithmus verwendet werden, um eine neue zu gewinnen.

Bemerkung 3.44.
Die Spannender-Baum-Struktur-Lösung, die nach der Durchführung eines Single Node Cuts gewonnen wird, kann den alten Zielfunktionswert noch weiter verbessern, als bei der Wahl des Single Node Cuts vorausberechnet wurde. Das liegt daran, dass der Fahrplan, der entsteht, wenn ein Ereignis verzögert wird, nur ein Repräsentant aller zulässigen Fahrpläne mit diesen Modulo-Parametern ist. Durch den Netzwerk-Simplex Algorithmus wird dagegen der optimale Vertreter bestimmt.

Selbstverständlich muss bei der Wahl eines Single Node Cuts weiterhin auf die Zulässigkeit in Hinsicht auf die unteren und oberen Schranken geachtet werden. Insgesamt erhalten wir somit ein Verfahren, dass in einer äußeren Schleife nach Single Node Cuts, und in einer inneren Schleife nach Fundamentalschnitten sucht. Algorithmus 5 stellt dies dar.

Bemerkung 3.45.
Wird anstelle der linearen Zielfunktion $F(x) = \sum \omega_{ij} x_{ij}$ die *bottleneck*-Funktion

$$F(x) = \max_{(i,j)\in\mathcal{A}} \omega_{ij}(x_{ij} - l_{ij})$$

verwendet, die eine Lösung nach ihrer schlechtesten Kante bewertet, so lässt sich das Modulo Netzwerk Simplex Verfahren sogar in einer beschleunigten Variante verwenden.
Mit linearem Aufwand in der Anzahl der Kanten lässt sich diejenige Kante finden (oder eine derjenigen), die den Zielfunktionswert erzeugt. Ist dies eine *Baumkante*, so muss im Modulo Simplex Tableau nur diejenige Spalte untersucht werden, die von dieser Kante erzeugt wird. Ist es dagegen eine *Nichtbaumkante*, so muss nur die entsprechende Zeile verwendet werden. Wird die Kante mit $e_{\max}$ bezeichnet und gibt es keine weiteren Kanten mit demselben gewichteten Slack,

Algorithmus 5 Der Modulo Netzwerk Simplex Algorithmus nach (NO08).

Eingabe: Eine (PF)-Instanz und eine zulässige Spannender-Baum-Struktur $\mathcal{T} = \mathcal{T}_l \cup \mathcal{T}_u$.
Ausgabe: Eine heuristische Lösung für (PF).

1: Berechne das Modulo Netzwerk Simplex Tableau.
2: Berechne für jeden Basiswechsel (i, j) mit $b_{ij} \neq 0$ die Kostendifferenz $\Delta\omega_{ij}$.
3: **if** ($\Delta\omega_{ij} \geq 0$ für alle zulässigen Basiswechsel (i, j).) **then**
4: Gehe zu 9.
5: **else**
6: Wähle die beste zulässe Pivotoperation und führe sie aus.
7: Gehe zu 1.
8: **end if**
9: Suche einen zulässigen verbessernden Single Node Cut.
10: **if** (Es gibt keinen zulässigen verbessernden Single Node Cut.) **then**
11: STOP: Heuristische Lösung gefunden.
12: **else**
13: Führe den gefundenen Schnitt aus.
14: Fixiere die Modulo-Parameter.
15: Erzeuge die duale Probleminstanz.
16: Löse sie unter Verwendung des Netzwerk-Simplex Verfahrens.
17: Erzeuge aus der Lösung eine neue Spannender-Baum-Struktur.
18: Gehe zu 1
19: **end if**

bestimmt sich die Kostendifferenz eines Basiswechsels durch

$$\Delta\omega_{ij} = \omega_e y_e - \max\left\{\omega_k\left(\left[y_k - \frac{b_{kj}}{b_{ij}}y_i\right]_T - y_k\right), \omega_j\left(\left[\frac{y_i}{b_{ij}} + y_j\right]_T - y_j\right)\right\},$$

wobei das Maximum über alle Kanten im durch e erzeugten Fundamentalschnitt, bzw. Fundamentalkreis gebildet wird. Ist dagegen eine weitere Kante mit demselben gewichteten Slack vorhanden, ändert sich der Zielfunktionswert nicht.

Bemerkung 3.46.
Das Verfahren, nach dem Basiswechsel gesucht werden, entspricht Dantzigs Pivotregel aus Abschnitt 2.2.2.

3.2.2 Single Node Cuts

Es wird nun genauer untersucht, wann ein Single Node Cut zulässig und verbessernd ist. Als Beispiel diene zunächst einen Knoten v mit einer eingehenden Kante e_1 und einer ausgehenden Kante e_2. Die unteren und oberen Schranken l_1, l_2, u_1 und u_2, sowie die Spannung x_1, x_2 eines zulässigen Fahrplans seien bekannt.

Wird nun der Zeitpunkt π_v, zu dem das Ereignis v stattfinden soll, um ein $delta \in (0, T-1]$ verschoben, so ist der resultierende Fahrplan offenbar genau dann zulässig, wenn gilt

$$x_1 + \delta \in [l_1, u_1] \cup [l_1 + T, u_1 + T]$$
$$x_2 - \delta \in [l_2, u_2] \cup [l_2 - T, u_2 - T],$$

was sich wiederum umformulieren lässt zu

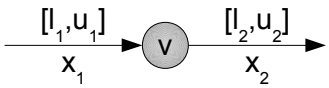


Abbildung 3.7: Beispiel für einen einfachen Single Node Cut.

$$\delta \in [0, u_1 - x_1] \cup [T + l_1 - x_1, T - 1] \tag{3.47}$$
$$\delta \in [0, x_2 - l_2] \cup [T - u_2 + x_2, T - 1] \tag{3.48}$$

.

Es lässt sich somit festhalten:

Satz 3.49. *Hinreichendes und notwendiges Kriterium für die Zulässigkeit von Single Node Cuts.*
Sei ein zulässiger Fahrplan π gegeben. Dann lässt sich der Zeitpunkt π_i genau dann zulässig um ein δ verschieben, wenn gilt:

$$\delta \in \left(\bigcap_{j:(j,i)\in\mathcal{A}} ([0, u_{ji} - x_{ji}] \cup [T + l_{ji} - x_{ji}, T - 1]) \right)$$
$$\cap \left(\bigcap_{j:(i,j)\in\mathcal{A}} ([0, x_{ij} - l_{ij}] \cup [T - u_{ij} + x_{ij}, T - 1]) \right).$$

Nun wird ausgenutzt, dass wir im Laufe des Modulo-Simplex-Algorithmus immer eine *Spannender-Baum-Lösung* verwenden. Also ist die Spannung mindestens einer der an einen Knoten grenzenden Kanten an ihrer unteren oder oberen Schranke. Ist im eingangs genannten Beispiel die eingehende Kante eine untere Baumkante, so wird Bedingung (3.47) zu

$$\delta \in [0, u_1 - l_1]$$

Das δ muss weiterhin auch die zweite Bedingung, nämlich

$$\delta \in [0, x_2 - l_2] \cup [T - u_2 + x_2, T - 1]$$

erfüllen. Offenbar ist die Bedingung $T - u_2 + x_2 \leq u_1 - l_1$ im in der Praxis üblichen Fall $T = 60$ nur selten erfüllt, während eine Überschneidung des ersten Intervalls über die 0 hinaus üblich sein sollte. Der folgende Satz gibt leider an, dass ein solcher Single Node Cut zwar zulässig, aber nicht verbessernd ist:

Satz 3.50.
Sei eine (PF)-Instanz und eine bezüglich der verfügbaren Basiswechsel lokal optimale Spannender-Baum-Struktur $\mathcal{T} = \mathcal{T}_l + \mathcal{T}_u$ gegeben. Werden die durch diese Struktur gegebenen Modulo-Parameter fixiert, so ist die durch die Struktur gegebene Lösung bezüglich dieser Modulo-Parameter optimal.

Beweis. Für fixierte Modulo-Parameter z ist die (PF)-Instanz äquivalent zu einer (AF)-Instanz. Wird wie in Abschnitt 2.2.1 beschrieben daraus die duale (MCF)-Instanz gewonnen, so erzeugt $\mathcal{T}$ eine Spannender-Baum-Struktur im Sinne der Definition 2.46. Diese Struktur kann als Anfangslösung für den klassischen Netzwerk-Simplex Algorithmus verwendet werden.
Angenommen, es gibt nun eine Nichtbaumkante, die eine der Optimalitätsbedingungen aus Satz 2.47 verletzt. Dann gibt es einen verbessernden Basiswechsel und somit auch im ursprünglichen Problem (PF) eine benachbarte Spannender-Baum-Struktur mit besserem Zielfunktionswert. Da die Lösung nach Voraussetzung lokal optimal ist, entsteht somit ein Widerspruch und es gibt keine die Optimalitätsbedingungen verletztende Baumkante. Somit ist im (MCF)-Problem die Lösung ebenfalls lokal und damit auch global optimal. Es gibt somit für fixierte Modulo-Parameter keine bessere Lösung. □

Korollar 3.51. *Notwendige Bedingung zur Verbesserung durch Single Node Cuts.*
Nur ein Single Node Cut, der mindestens einen Modulo-Parameter verändert, kann den Zielfunktionswert einer lokal optimalen Spannender-Baum-Struktur verbessern.

Im Beispiel einer eingehenden und einer ausgehenden Kante muss somit einer der drei Fälle eintreten:

1. $\delta \in [0, u_1 - x_1] \cap [T - u_2 + x_2, T - 1]$
2. $\delta \in [0, x_2 - l_2] \cap [T + l_1 - x_1, T - 1]$
3. $\delta \in [T - u_2 + x_2, T - 1] \cap [T + l_1 - x_1, T - 1]$

Durch die weitere Einschränkung, dass mindestens eine der Kanten eine untere oder obere Baumkante ist, fällt jeweils mindestens eine Möglichkeit weg. Diese Einschränkungen machen deutlich, warum in der Praxis relativ selten ein verbessernder Single Node Cut auffindbar ist. Besteht ein Single Node Cut aus mehr als zwei Kanten, so gibt es einerseits *mehr* Möglichkeiten, einen Modulo-Parameter zu verändern, andererseits *weniger* Möglichkeiten, die Zulässigkeit zu erhalten.

3.3 Allgemeine Schnitt-Heuristiken

Das Modulo-Simplex-Verfahren, wie es in obiger Form vorgestellt worden ist, benutzt abwechselnd zwei Sorten von Schnitten: Fundamentalschnitte und Single Node Cuts. Einerseits lässt sich das Verfahren, auf welche Weise ein Schnitt ausgewählt wird, andererseits auch die Menge, aus der heraus gewählt werden soll, verändern. Die so gewonnen Verfahren werden als *allgemeine Schnitt-Heuristiken* bezeichnet. Einzelne Vertreter werden in diesem Abschnitt erläutert.

3.3.1 Abstrakter Aufbau

Der große Vorteil der Verwendung von Fundamentalschnitten liegt wie gezeigt darin, dass sie in Linearkombination *jede beliebige* zulässige Veränderung der Lösung erzeugen können, gleichzeitig aber in einer Anzahl existieren, die ein

Vorgehen erlaubt, das auch jede Möglichkeit untersucht. Insofern ist es lohnenswert, diese Schnitte weiterhin zu verwenden, allerdings ihre Auswahlprozedur zu ändern.
Single Node Cuts andererseits haben den Nachteil, dass sie, wie in Abschnitt 3.2.2 gezeigt, nur beschränkt einsetzbar sind. Sollte also kein verbessernder Fundamentalschnitt mehr verfügbar sein, kann es wünschenswert sein, einen anderen Schnitt zu wählen. In allgemeiner Form lauten also Schnitt-Heuristiken wie in Algorithmus 6 beschrieben.

Algorithmus 6 Meta-Schnittheuristik

Eingabe: Eine (PF)-Instanz und eine zulässige Spannender-Baum-Struktur $\mathcal{T} = \mathcal{T}_l \cup \mathcal{T}_u$.
Ausgabe: Eine heuristische Lösung für (PF).
1: Berechne das Modulo Netzwerk Simplex Tableau.
2: Suche einen zulässigen Basiswechsel gemäß einem gegebenen Suchkriterium.
3: **if** (Kein Basiswechsel wurde gefunden.) **then**
4: Gehe zu 9.
5: **else**
6: Führe den Basiswechsel aus.
7: Gehe zu 1.
8: **end if**
9: Wähle einen zulässigen Schnitt aus einer gegebenen Schnittmenge.
10: **if** (Es wurde kein Schnitt gefunden.) **then**
11: STOP: Heuristische Lösung gefunden.
12: **else**
13: Führe den gefundenen Schnitt aus.
14: Fixiere die Modulo-Parameter.
15: Erzeuge die duale Probleminstanz.
16: Löse sie unter Verwendung des Netzwerk-Simplex Verfahrens.
17: Erzeuge aus der Lösung eine neue Spannender-Baum-Struktur.
18: Gehe zu 1
19: **end if**

Beispiele für das im Schritt 2 genannte Suchkriterium für Fundamentalschnitte werden in den nun folgenden Abschnitten 3.3.2, 3.3.3 und 3.3.4 beschrieben, während in 3.3.5 und 3.3.6 zwei Vertreter für die Schnittmengen aus Schritt 9 vorgestellt werden.

3.3.2 Steilster Abstieg

Im originalen Modulo-Simplex-Verfahren ist das Suchkriterium nach Fundamentalschnitten dadurch gegeben, dass die gesamte Nachbarschaft betrachtet und der lokal beste Zustand gewählt wird. Verfahren dieser Art sind als *Verfahren des steilsten Abstiegs* (steepest descent) oder, vom Standpunkt der Maximierung aus betrachtet, als *hill climbing* bekannt. Abbildung 3.8 veranschaulicht skizzenhaft im eindimensionalen Fall die Problematik, dass nur das nächstliegende lokale Optimum erreicht wird.

Aufgrund dessen wurden verbesserte Verfahren entwickelt, die die Nachbarschaft eines Zustandes durchsuchen, ohne den jeweils kostenbesten Schritt zu wählen.

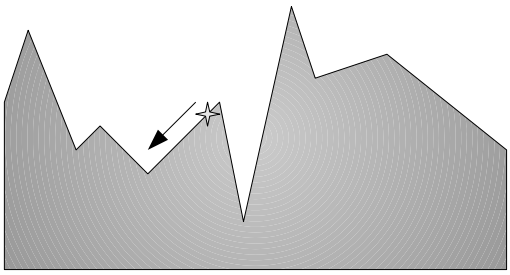


Abbildung 3.8: Der steilste Abstieg sucht das nächstgelegene lokale Optimum.

3.3.3 Tabu Search

Das *Tabu Search* Verfahren wurde 1989/1990 von Fred Glover in den Artikeln (Glo89) und (Glo90) mit dem Ziel entwickelt, Steilste-Abstieg Verfahren so zu verbessern, dass sie auch lokale Optima wieder verlassen können.
In der einfachsten Form werden die im Laufe des Verfahrens bisher besuchten Zustände in einer *Tabu Liste* gespeichert und verhindern so, dass gleiche Zustände in kurzer Zeit hintereinander besucht werden. Von allen Zügen, die nicht als Tabu gelten, wird in jedem Schritt wie im Verfahren des steilsten Abstiegs derjenige gewählt, der den kleinsten Zielfunktionswert verursacht. Das Verfahren in der einfachsten Form ist in Algorithmus 7 dargestellt.

Algorithmus 7 Einfaches Tabu Search nach (Glo89)

Eingabe: Ein Optimierungsproblem $(P) : \{\min c(x) : x \in X\}$, eine Nachbarschaftsfunktion $s : x \mapsto S(x)$ und eine zulässige Lösung $x \in X$.
Ausgabe: Eine heuristische Lösung x^* von (P).

```
1: Setze T := ∅ und k = 0.
2: while (Das Abbruchkriterium wurde nicht erfüllt.) do
3:   if (S(x) \ T = ∅) then
4:     STOP: x*, falls definiert, ist heuristische Lösung.
5:   else
6:     Setze k := k + 1.
7:     Wähle x := argmin{c(y) : y ∈ S(x) \ T}.
8:     Ist c(x) < c(x*), wobei x* die beste bisher gefundene Lösung bezeichne,
       setze x* := x.
9:     Erneuere T.
10:  end if
11: end while
```

Ein typisches Abbruchkriterium ist dabei, dass eine vorgegebene Anzahl an Iterationen von Beginn des Algorithmus an oder seit der letzten Verbesserung nicht überschritten werden darf. Ein mögliches Erneuerungsverfahren für T ist, den neu besuchten Zustand mit in die Liste aufzunehmen und den ältesten Zustand dafür zu löschen, falls eine Länge l der Liste erreicht worden ist.
Gerade im Zusammenhang mit der periodischen Fahrplangestaltung kann es einerseits sehr speicheraufwändig werden, eine Liste von besuchten spannenden Bäumen zu führen, und andererseits sehr lange dauern, *jeden* Baum der Nach-

barschaft mit *jedem* Baum der Liste zu vergleichen. Diese Probleme werden dadurch umgangen, dass nicht mehr der gesamte Zustand in der Liste geführt wird, sondern nur eine Auswahl an charakteristischen Daten. Je nach Wahl können dadurch mehr Zustände verboten werden, als eigentlich besucht wurden, was durch die Einführung eines *Aspirationskriteriums* wieder ausgeglichen wird. Zustände, die das Aspirationskriterium erfüllen, werden immer zugelassen; ob sie von der Tabu Liste verboten werden oder nicht. Ein typisches Kriterium lautet, einen Zustand zuzulassen, wenn er das bisher gefundene Optimum noch übertrifft. Algorithmus 8 fasst diese Verbesserungen zusammen.

Algorithmus 8 Verbessertes Tabu Search nach (Glo89)

Eingabe: Ein Optimierungsproblem $(P) : \{\min c(x) : x \in X\}$, eine Nachbarschaftsfunktion $s : x \mapsto S(x)$, eine zulässige Lösung $x \in X$, ein Aspirationskriterium $A : x \mapsto A(x)$ und eine Zustandscharakteristik C auf X.
Ausgabe: Eine heuristische Lösung x^* von (P).

Setze $T := \emptyset$ und $k = 0$.
while (Das Abbruchkriterium wurde nicht erfüllt.) **do**
  if $(S(x) \cup A(x) \setminus \{x \in X : C(x) \neq C(y)\ \forall y \in T\} = \emptyset)$ **then**
    STOP: x^*, falls definiert, ist heuristische Lösung.
  else
    Setze $k := k + 1$.
    Wähle
    $x := \text{argmin}\{c(y) : y \in S(x) \cup A(x) \setminus \{x \in X : C(x) \neq C(y)\ \forall y \in T\}\}$.
    Ist $c(x) < c(x^*)$, wobei x^* die beste bisher gefundene Lösung bezeichne, setze $x^* := x$.
    Erneuere T.
  end if
end while

Angewandt auf das Problem der periodischen Fahrplangestaltung stellt dieses Verfahren ein Suchkriterium gemäß Schritt 2 des Algorithmus 6 dar, wird als Nachbarschaftsfunktion die Menge der Spannender-Baum-Strukturen verwendet, die von der augenblicklichen Lösung aus erreichbar sind. Die Implementationsdetails werden im Abschnitt 5.2.4 des experimentellen Teils gegeben.

3.3.4 Simulated Annealing

Das Verfahren der *simulierten Abkühlung* (simulated annealing) wurde unter anderem von V. Černý 1985 in (Č85) für das Travelling Salesman Problem als eine Art Monte-Carlo Verfahren entwickelt und allgemeiner schon 1983 von Kirkpatrick, Gelatt und Vecchi für kombinationstheoretische Probleme in (KJV83). Die Inspiration stammt aus der thermophysikalischen Beobachtung, dass Atome eines heißen Gegenstandes wie zufällig große Bewegungen vollführen, aber bei abnehmender Hitze immer kleinere Bewegungen vollführen, bis sie schließlich einen energetisch lokal optimalen Zustand wie zum Beispiel ein Kristallgitter erreichen.
Dies wird modelliert durch eine mitgeführte Temperatur T, die im Laufe der Iterationen nach einem Temperaturschema verringert wird. Ein typischer Vertreter ist die geometrische Abkühlung $T := \lambda T$, die die Temperatur jeweils um einen festen Faktor verringert. Solange die Temperatur hoch ist, sollen mit hoher Wahrscheinlichkeit auch auch benachbarte Zustände besucht werden, die den Zielfunktionswert stark verschlechtern; diese Wahrscheinlichkeit wird mit

sinkender Temperatur kleiner. Algorithmus 9 stellt das gesamte Verfahren dar.

Algorithmus 9 Simulated Annealing

Eingabe: Ein Optimierungsproblem $(P) : \{\min c(x) : x \in X\}$, eine Nachbarschaftsfunktion $s : x \mapsto S(x)$, eine Temperatur T, ein Temperaturschema $f : (T, k) \mapsto T'$ und eine zulässige Lösung $x \in X$.

Ausgabe: Eine heuristische Lösung x von (P).

```
1:  Setze k = 0.
2:  Setze found auf TRUE.
3:  while (found = TRUE) do
4:    Setze found auf FALSE.
5:    S := S(x).
6:    while (S ≠ ∅ und found = FALSE) do
7:      Wähle ein zufälliges y ∈ S,
8:      if (c(y) < c(x)) then
9:        Setze x := y.
10:       Setze T := f(T, k).
11:       Setze k := k + 1.
12:       Setze found = TRUE.
13:     else
14:       Setze P := exp((c(x) − c(y))/T).
15:       Finde ein p gleichverteilt auf [0, 1].
16:       if (p < P) then
17:         Setze x := y.
18:         Setze T := f(T, k).
19:         Setze k := k + 1.
20:         Setze found = TRUE.
21:       else
22:         Setze S := S \ {y}.
23:       end if
24:     end if
25:   end while
26: end while
```

Offenbar kann es unter Umständen geschehen, dass das Verfahren in dieser Form nicht abbricht, wenn das gefundene lokale Optimum auf einem Plateau liegt. Entsprechend müssen Vorkehrungen wie ein zusätzliches Abbruchkriterium, wenn eine gewünschte Temperatur erreicht worden ist, oder die Veränderung der Wechselwahrscheinlichkeit auf $P := \exp((c(x) - c(y) + 1)/T)$ getroffen werden. Ebenso wie Tabu Search lässt sich dieses Verfahren auf die periodische Fahrplangestaltung anwenden, wenn die Menge der durch einen Basiswechsel erreichbaren Spannender-Baum-Strukturen als Nachbarschaft aufgefasst wird.

3.3.5 Waiting Edge Cuts

Anstatt das Verfahren zu ändern, über das der Fundamentalschnitt gewählt wird, lassen sich auch für die anschließende „lokale Verbesserung“ Alternativen finden. Wie in Abschnitt 3.2.2 bereits gezeigt wurde, ist es für einen Single Node Cut notwendig, dass die zulässigen Kantenintervalle $[l_{ij}, u_{ij}]$ möglichst groß sind. Das ist häufig gerade für die *Warteaktivitäten* nicht der Fall. Um nicht durch diese Kanten die Zulässigkeit von Single Node Cuts zu sehr einzuschränken, lassen sich *Waiting Edge Cuts* verwenden, das heißt Schnitte, die durch eine Warteaktivität

(i, j) erzeugt werden, indem die Knotenpartition als $\{i, j\} \cup (\mathcal{A} \setminus \{i, j\})$ gewählt wird. Abbildung 3.9 stellt einen solchen Schnitt dar.

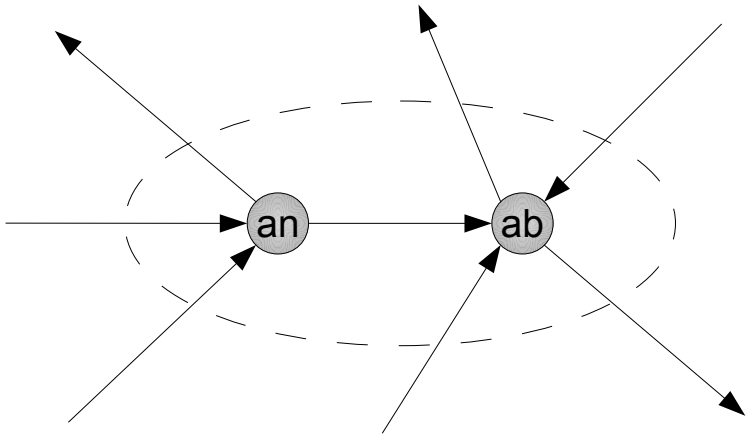


Abbildung 3.9: Ein Waiting Edge Cut.

Durch die Konstruktion der EANs ist es unmöglich, dass zwei Warteaktivitäten zu demselben Knoten adjazent sind. In keinem Waiting Edge Cutbefindet sich somit eine weitere Warteaktivität, was dem ursprünglichen Ziel entgegen gestanden hätte.

Bemerkung 3.52.
Ein hinreichendes Kriterium für die Verbesserung einer Lösung durch einen Waiting Edge Cutentlang der Kante (i, j) mit $\delta \in [0, T-1]$ ist

$$\begin{aligned} &\sum_{k:(i,k)\in\mathcal{A}} \omega_{ik}(y_{ik} - \delta) + \sum_{k:(k,i)\in\mathcal{A}} \omega_{ki}(y_{ki} + \delta) \\ &+ \sum_{k:(j,k)\in\mathcal{A}} \omega_{jk}(y_{jk} - \delta) + \sum_{k:(k,j)\in\mathcal{A}} \omega_{kj}(y_{kj} + \delta) \\ &-2\omega_{ij}y_{ij} < 0. \end{aligned}$$

Wie im Falle der Single Node Cuts ist es lediglich hinreichend, da durch die anschließende Lösung des dualen Problems eine Verbesserung des Zielfunktionswert auch dann möglich ist, wenn diese Gleichung positiv wird.

Bemerkung 3.53.
Ein Waiting Edge Cutentlang einer Kante (i, j) ist genau dann zulässig, wenn

gilt

$$\delta \in \left(\bigcap_{k:(k,i)\in\mathcal{A}} ([0, u_{ki} - x_{ki}] \cup [T + l_{ki} - x_{ki}, T - 1]) \right)$$
$$\cap \left(\bigcap_{\substack{k:(i,k)\in\mathcal{A} \\ k\neq j}} ([0, x_{ik} - l_{ik}] \cup [T - u_{ik} + x_{ik}, T - 1]) \right)$$
$$\cap \left(\bigcap_{\substack{k:(k,j)\in\mathcal{A} \\ k\neq i}} ([0, u_{kj} - x_{kj}] \cup [T + l_{ji} - x_{ji}, T - 1]) \right)$$
$$\cap \left(\bigcap_{k:(j,k)\in\mathcal{A}} ([0, x_{jk} - l_{jk}] \cup [T - u_{jk} + x_{jk}, T - 1]) \right).$$

3.3.6 Multi Node Cuts

Selbstverständlich lässt sich *jeder beliebige* Schnitt verwenden, um das lokale Optimum, das durch Fundamentalschnitte gefunden wurde, wieder zu verlassen. In diesem Abschnitt wird gezeigt, dass gar nicht *alle*, sondern nur sogenannte *zusammenhängende* Schnitte betrachtet werden müssen, und sich diese durch Single Node Cuts erzeugen lassen.

Definition 3.54.
Seien in einem gerichteten Graphen $G(V, E)$ ein Schnitt c, der durch die Knotenpartition $V_1 \dot\cup V_2$ erzeugt wird, und ein Single Node Cut c_i zum Knoten $i \in V_2$ gegeben. Dann heißt der Schnitt, der aus der Partition $(V_1 \cup \{i\})$ und $(V_2 \setminus \{i\})$ erzeugt wird, die *exklusive Vereinigung* dieser Schnitte und wird $c \oplus c_i$ geschrieben.

Beispiel 3.55.

In Abbildung 3.10 wird gezeigt, wie die exklusive Vereinigung eines Schnittes mit einem Single Node Cut aussieht. Im Teil 3.10 (a) ist der Schnitt dargestellt, der aus der Partition $V_1 = \{v_1, v_3, v_4, v_6\}$, $V_2 = \{v_2, v_5\}$ entsteht. Die Orientierung der Schnittkanten ist entsprechend von V_1 nach V_2 eingezeichnet.
Im zweiten Teil 3.10 (b) ist der Single Node Cut zum Knoten v_2 eingezeichnet. Abbildung 3.10 (c) zeigt schließlich die exklusive Vereinigung. Es lässt sich erkennen, dass die Orientierungen der den beiden Schnitten gemeinsamen Kanten entgegengesetzt verlief, wodurch sie sich „ausglichen“.

Satz 3.56.
Seien c_i und c_j Single Node Cuts zu benachbarten Knoten in einem gerichteten Graphen $G(V, E)$. Dann entspricht der Schnitt $c_i \oplus c_j$ einem Edge Cut wie in Abschnitt 3.3.5 beschrieben.

Beweis. Der Schnitt $c_i \oplus c_j$ wird per Definition durch die Knotenpartition $(\{i, j\}) \dot\cup (V \setminus \{i, j\})$ erzeugt. Das ist die in Abschnitt 3.3.5 gegebene Definition für Edge Cuts. □

Satz 3.57.
Sei ein Schnitt c im Graphen $G(V, E)$ durch die Knotenpartition $V_1 \dot\cup V_2$ gegeben.

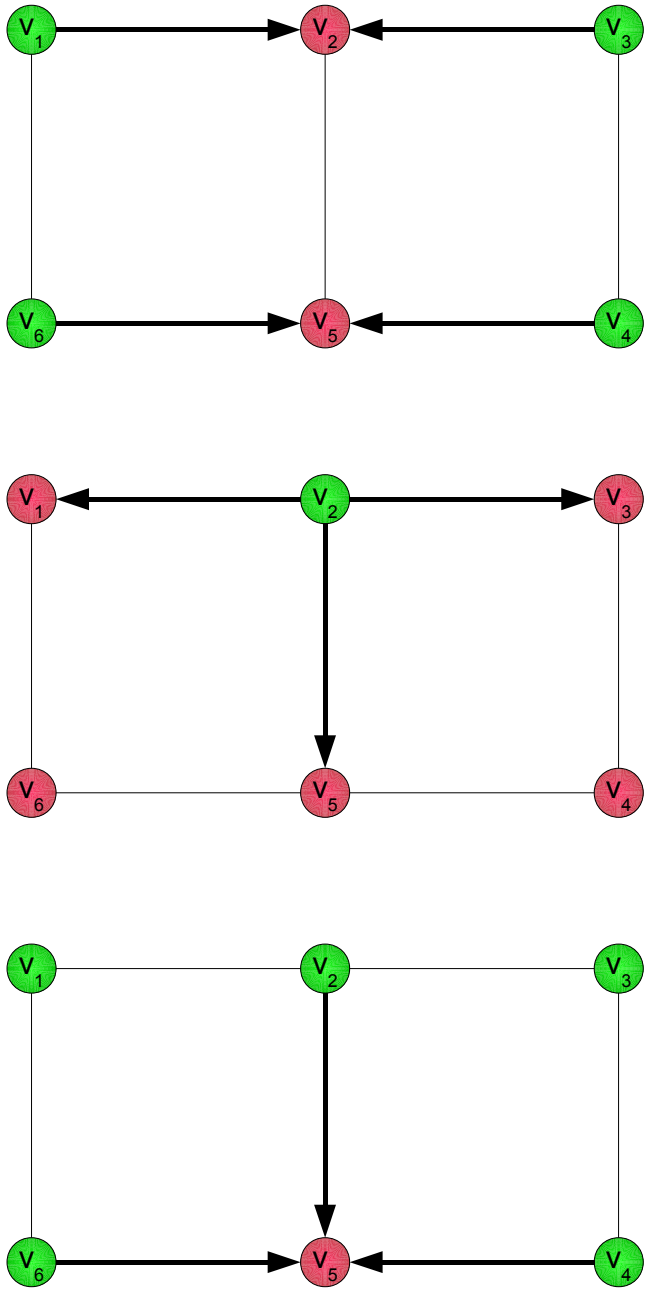


Abbildung 3.10: Ein Beispiel für die exklusive Vereinigung.

Dann ist c als exklusive Vereinigung von Single Node Cuts darstellbar:

$$\bigoplus_{i \in V_1} c_i = c.$$

Beweis. Sei zunächst $e \in c$. Dann ist $e = (i, j)$ oder $e = (j, i)$ mit $i \in V_1$ und $j \in V_2$. Da es einen Single Node Cut zum Knoten i, nämlich c_i, gibt, aber nach Konstruktion keinen für Knoten j, ist e in der exklusiven Vereinigung der Single Node Cuts enthalten.
Sei andererseits $e \in \oplus c_i$. Nach Konstruktion der Single Node Cuts ist dann entweder der Ursprung oder das Ziel der Kante in V_1, der jeweils andere Knoten in V_2. Somit ist $e \in c$. □

Korollar 3.58.
Die Veränderung durch jeden beliebigen Schnitt in einer (PF)-Instanz um ein δ kann äquivalent durch die sukzessive Anwendung von Single Node Cuts mit diesem δ erzeugt werden.

Satz 3.59.
Sei $G(V, E)$ ein zusammenhängender Graph und $c = \oplus_{i \in I} c_i$ ein nichtleerer Schnitt, der durch Single Node Cuts dargestellt und durch die Partition $V_1 \dot\cup V_2$ erzeugt wird. Dann ist $I = V_1$ (oder $I = V_2$, wird die Orientierung des Schnittes nicht berücksichtigt).

Beweis.

- Sei zunächst $V_1 \subsetneq I$. Angenommen, es gibt benachbarte Knoten $i, j \in I$ mit $i \in V_1$ und $j \notin V_1$. Dann ist $(i, j) \notin \oplus_{i \in I} c_i$, aber $(i, j) \in c$ - somit kann es solche benachbarte Knoten nicht geben. Da c nichtleer ist, kann ferner $I = V$ nicht sein. Da G zusammenhängend ist, gibt somit es einen Knoten $i \in I \setminus V_1$, der zu einem Knoten $j \notin I$ benachbart ist. Somit wäre die Kante (i, j) in $\oplus_{i \in I} c_i$ enthalten, aber wiederum nicht in c.

- Sei nun $I \subsetneq V_1$. Dann ist bis auf Orientierung $\oplus_{i \in I} c_i = \oplus_{j \in V \setminus I} c_j$ und der Fall lässt sich auf $V_2 \subsetneq J$ mit $J = V \setminus I$ zurückführen, was bereits behandelt wurde.

- Sei nun $I \cap V_1 \neq \emptyset$, $I \cap V_2 \neq \emptyset$ und I keine Obermenge zu V_1 oder V_2. Damit $c = \oplus_{i \in I} c_i$ gelten kann, muss für jede Kante (i, j) mit $i \in V_1$ und $j \in V_2$ ohne Beschränkung der Allgemeingültigkeit auch $i \in I$ und $j \notin I$ sein. Nach Voraussetzung gibt es $i \in V_2 \cap I$ und $j \in V_2 \setminus I$. Da der Graph zusammenhängend ist, gibt es einen Pfad von i nach j. Es kann i nicht zu einem Knoten in V_1 benachbart sein, da sonst $i \in I$ nicht gelten könnte. Somit gibt es eine Kante (x, y) mit $x \in V_2 \cap I$ und $y \in V_2 \setminus I$, die folglich in $\oplus_{i \in I} c_i$, aber nicht in c enthalten ist.

□

Es ist einleuchtend, dass die Veränderung einer Lösung durch zwei Single Node Cuts, die genügend weit voneinander entfernt sind, einfach der Summe der jeweiligen Veränderungen entspricht. Sollte einer dieser Single Node Cuts eigentlich verschlechternd sein, sollte er besser ausgelassen werden. Diese Überlegung wird nun präzisiert.

Definition 3.60. Zusammenhang von Schnitten
Ein Schnitt, der durch die Knotenpartition $V_1 \dot\cup V_2$ entsteht, heiße *zusammenhängend*, wenn die Teilgraphen $G_1(V_1, E_{V_1})$ und $G_2(V_2, E_{V_2})$ jeweils zusammenhängend sind.

Beispiel 3.61.
Betrachte den Schnitt in Abbildung 3.11, der durch die Knotenpartition $V_1 \dot\cup V_2$ mit $V_1 = \{v_1, v_4\}$ und $V_2 = \{v_2, v_3, v_5, v_6\}$ erzeugt wird.

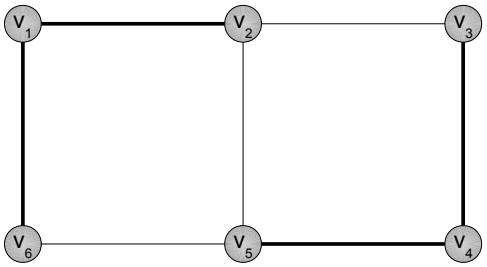


Abbildung 3.11: Ein nicht zusammenhängender Schnitt.

Zwar ist $G_2(V_2, E_{V_2})$ zusammenhängend, doch $G_1(V_1, E_{V_1})$ ist es nicht, weshalb der Schnitt als unzusammenhängend gilt. Würden die Schnittkanten aus dem Graphen entfernt werden, entstünden *drei* zusammenhängende Graphen. Die Schnitte $\{v_1\} \dot\cup \{v_2, v_3, v_4, v_5, v_6\}$ und $\{v_4\} \dot\cup \{v_1, v_2, v_3, v_5, v_6\}$ sind dagegen zusammenhängend. Sie zerlegen den Graphen in *zwei* zusammenhängende Graphen.

Bemerkung 3.62.
Nach Definition sind genau diejenigen Schnitte zusammenhängend, die durch Entfernen ihrer Kanten den Graphen in genau zwei zusammenhängende Teilgraphen zerlegen.

Korollar 3.63.
Ein Schnitt $c = \oplus_{i \in I} c_i$ in einem Graphen $G(V, E)$, wobei die c_i jeweils Single Node Cuts sind, ist genau dann zusammenhängend, wenn die durch die Mengen I und $V \setminus I$ induzierten Teilgraphen zusammenhängend sind.

Definition 3.64. Zusammenhangskomponenten von Schnitten
Seien ein Schnitt c, der durch die Knotenpartition $V_1 \dot\cup V_2$ erzeugt wird, und ein Schnitt $c' \subset c$, der durch die Knotenpartition $V_1' \dot\cup V_2'$ mit $V_1' \subset V_1$ und $V_2 \subset V_2'$ erzeugt wird, gegeben. c' wird *Zusammenhangskomponente* von c genannt, wenn c' zusammenhängend ist, also jeweils die Graphen $G_1'(V_1', E_{V_1'})$ und $G_2'(V_2', E_{V_2'})$ zusammenhängend sind, und es keinen Schnitt c'' gibt mit $c' \subsetneq c'' \subset c$, der ebenfalls zusammenhängend ist.

Beispiel 3.65.

- Der Schnitt aus Abbildung 3.11 besteht aus den beiden Zusammenhangskomponenten $\{(v_1, v_2), \{v_1, v_6)\}$ und $\{(v_3, v_4), (v_4, v_5)\}$. Wird jeweils eine Schnittkante hinzugefügt, so sind die den Knotenpartitionen entsprechenden Teilgraphen nicht mehr zusammenhängend.

- Der Schnitt in Abbildung 3.12 umfasst den gesamten Graphen und besteht aus den Zusammenhangskomponenten $\{(v_1, v_2)\}$, $\{(v_2, v_3)\}$ und $\{(v_3, v_4)\}$.

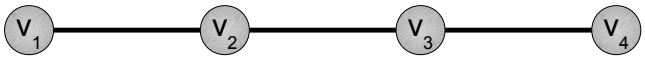


Abbildung 3.12: Ein Schnitt mit drei Zusammenhangskomponenten.

Satz 3.66.
Zusammenhangskomponenten eines Schnittes sind paarweise disjunkt.

Beweis. Seien c^1 und c^2 Zusammenhangskomponenten eines Schnittes c mit $c^1 \cap c^2 \neq \emptyset$. Dann ist $c' := c^1 \cup c^2$ zusammenhängend und $c^1 \subsetneq c' \subset c$. Nach Definition 3.64 können c^1 und c^2 somit keine gemeinsame Kante besitzen. □

Wird in einer (PF)-Instanz ein Schnitt angewandt, um eine Lösung zu verändern, so ändert sich der Zielfunktionswert entsprechend der Summe der Veränderungen auf jeder einzelnen Kante. Da Zusammenhangskomponenten disjunkt sind, ändert sich der Zielfunktionswert somit entsprechend der Veränderung durch jede einzelne Komponente.

Korollar 3.67.
In einer (PF)-Instanz verändert sich der Zielfunktionswert einer Lösung durch Anwendung eines Schnittes c mit einem $\delta \in [0, T-1]$ um die Summe der Veränderungen des Zielfunktionswertes durch die Zusammenhangskomponenten.

Korollar 3.68.
Die Suche nach verbessernden Schnitten in einem EAN darf sich auf zusammenhängende Schnitte beschränken.

Das inspiriert die Heuristik 10, einen verbessernden Schnitt zu finden:

Dieses Verfahren kann für verschiedene Knoten $i \in V$, von denen aus gestartet wird, wiederholt werden.

Bemerkung 3.69.
Wird Heuristik 10 für jeden Knoten $i \in V$ wiederholt, so wird sie immer einen verbessernden Schnitt finden, wenn es einen verbessernden Single Node Cut gibt. In diesem Sinne ist das Verfahren *garantiert* besser als die Suche nach Single Node Cuts, doch kostet es wesentlich mehr Zeit.

Algorithmus 10 Suche nach einem verbessernden Schnitt.

Eingabe: Eine (PF)-Instanz mit einer zulässigen Lösung x.
Ausgabe: Ein verbessernder Schnitt c mit $\delta \in [1, T-1]$, falls einer gefunden wird.

1: Wähle einen Knoten $i \in V$.
2: Setze $S := \{i\}$.
3: Sei c der durch S induzierte Schnitt.
4: **if** (Es gibt ein $\delta \in [1, T-1]$, so dass c zulässig und verbessernd ist.) **then**
5: STOP: c ist verbessernder Schnitt mit dem gefundenen δ.
6: **else**
7: **if** ($c \neq \emptyset$.) **then**
8: $I := \{i \in V \setminus S : V \setminus \{i\}$ und $S \cup \{i\}$ sind jeweils zusammenhängend$\}$.
9: Sortiere die Kanten $e \in c$ aufsteigend nach $u_e - l_e$.
10: Wähle die erste so sortierte Kante, die mit einem Knoten $i \in I$ inzident ist.
11: Füge den so gefundenen Knoten zu S hinzu.
12: Gehe zu 4
13: **else**
14: STOP: Kein Schnitt gefunden.
15: **end if**
16: **end if**

Teil II

Experimenteller Teil

„Nun würde man den empirischen Maschinisten, welcher über die allgemeine Mechanik, oder den Artilleristen, welcher über die mathematische Lehre vom Bombenwurf so absprechen wollte, daß die Theorie davon zwar fein ausgedacht, in der Praxis aber gar nicht gültig sei, weil bei der Ausübung die Erfahrung ganz andere Resultate gebe als die Theorie, nur belachen (denn, wenn zu der ersten noch die Theorie der Reibung, zur zweiten die des Widerstandes der Luft, mithin überhaupt nur noch mehr Theorie hinzu käme, so würden sie mit der Erfahrung gar wohl zusammen stimmen).“

-Immanuel Kant,
Über den Gemeinspruch: Das mag in der Theorie richtig sein, taugt aber nicht für die Praxis

Kapitel 4

Programmierung

4.1 LinTim

Mit diesem Abschnitt beginnt die angewandte Untersuchung der periodischen Fahrplangestaltung. Es wird das Projekt „LinTim“ präsentiert, in das die Programmierung integriert wurde, und erläutert, wie EANs für die folgenden Versuche konstruiert wurden.

Im Rahmen des europäischen ARRIVAL-Projektes entstand das Projekt „LinTim“ an der Universität Göttingen (SS09). Obwohl der Name irreführenderweise für „lineplanning and timetabling“ steht, wächst der Funktionsumfang zunehmend auch über weitere Bereiche der Verkehrsoptimierung hinaus. Das Ziel ist, die in der Einführung erwähnte gegenseitige Einflussnahme der einzelnen Planungsphasen durch vereinheitliche, sehr allgemein gehaltene Datenformate und modulare Programme besser untersuchen zu können.
Momentan sind bereits Programme für die Schritte Linienplanung, Fahrplangestaltung und Delay Management implementiert. Die Schnittstellen sind im plain-text Format gehalten, was sogar die Einbindung verschiedener Programmiersprachen je nach Präferenz desjenigen, der einen weiteren Algorithmus hinzufügen möchte, ermöglicht. Bisher definierte Formate umfassen unter anderem

- die OD-Matrix,
- die Kantenmenge des PTN,
- die Kantengewichte des PTN,
- den Linienpool,
- das Linienkonzept,
- die Aktivitäten und Ereignisse des EAN
- und den Fahrplan.

So lautet zum Beispiel der Anfang der Datei EDGE.GIV, die die Kanten des PTN eines einfachen Spielbeispiels beinhaltet, folgendermaßen:

```
# small example of PTN
# traffic planning script, AS, 10.9.2007
edge-id; left-stop-id; right-stop-id ; length ; lower-bound; upper-bound
```

```
1; 1; 3; 1; 5;7
2; 2; 3; 1; 3;4
3; 3; 4; 1; 4;5
...
```

Mittels `GraphViz` ist LinTim auch in der Lage, automatisch das PTN zu plotten. Das oben genannte Beispiel sieht dann so aus:

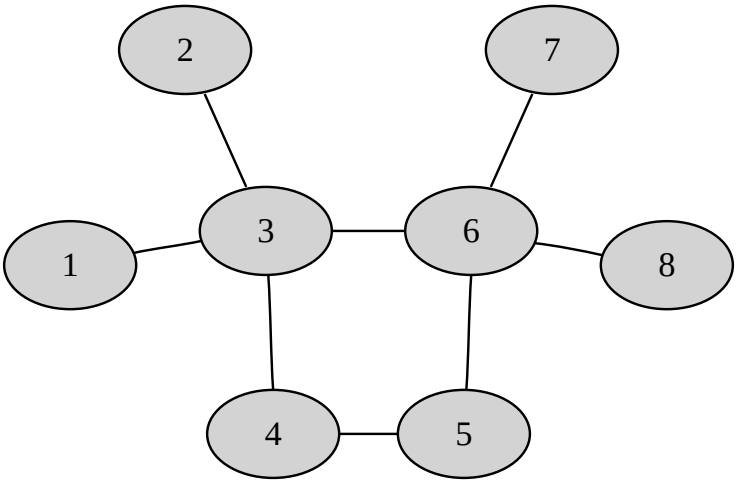


Abbildung 4.1: Ein einfaches PTN aus LinTim.

Neben diesem einfachen Beispiel, das es erlaubt, Algorithmen per Hand nachzuvollziehen, gibt es zwei Datensätze von praktisch relevanter Größenordnung, im folgenden „Bahn-Groß“ und „Bahn-Klein“ genannt. Die Tabelle 4.1 gibt die wesentlichen Kennzahlen der PTNs wieder.

	Knoten	Kanten	Linien	OD-Paare
Spiel	8	8	8	22
Bahn-Klein	250	326	132	31.125
Bahn-Gross	319	452	2.770	50.721

Tabelle 4.1: Kennzahlen der LinTim-PTNs.

Im Zusammenhang dieser Arbeit ist nun von Bedeutung, auf welche Weise aus diesen Probleminstanzen Linienkonzepte gewonnen werden, da sich diese wiederum stark auf das spätere EAN auswirken und damit den Input für die hier zu untersuchenden Algorithmen erzeugen.
Bisher stehen zur Lösung des Linienplanungsproblems (LP1) zur Verfügung:

- Die Lösung der LP-Formulierung mittels eines Solvers, in diesem Fall „Xpress“ von *dash optimization*.
- Die Greedy-Algorithmen 2 und 3, die in Abschnitt 2.1.1 bereits vorgestellt wurden. Im Rahmen der LinTim-Implementierung werden sie H6 und H7 genannt, was auf ihr Auftauchen im Skript (Sch09) zurückgeht.

- Ein auf der Suche nach Matchings basierender Ansatz, der aus der Diplomarbeit von Rasmus Fuhse (Fuh08) unter „Cutting-Down“ hervorgegangen ist.

Wie im Abschnitt 2.1.2 erläutert, wird das gewonnene Linienkonzept automatisch in ein EAN mit Periode $T = 60$ umgewandelt. Dafür werden zunächst *alle* Umsteigekanten zwischen Fahrten verschiedener Linien, die sich am selben Bahnhof treffen, gezogen; durch das anschließende approximierte Routen der Passagiere durch das Netzwerk fällt der Großteil der solcherart gezogenen Umsteigekanten wieder weg, da sie mit einem Kantengewicht 0 belegt werden. Tabelle 4.2 stellt Kennzahlen der resultierenden EANs dar.

		Ereignisse	Aktivitäten	Anteil an Umsteigeaktivitäten
Xpress	Spiel	52	56	25,0%
	Bahn-Klein	3.528	4.706	27,4%
	Bahn-Gross	5.008	8.140	40,6%
H6	Spiel	44	58	34,5%
	Bahn-Klein	4.872	5.654	16,6%
	Bahn-Gross	10.560	12.055	15,0%
H7	Spiel	60	63	23,8%
	Bahn-Klein	4.184	5.407	25,0%
	Bahn-Gross	6.368	8.642	29,0%
Matchings	Spiel	76	76	10,5%
	Bahn-Klein	unbekannt	unbekannt	unbekannt
	Bahn-Gross	unbekannt	unbekannt	unbekannt

Tabelle 4.2: Kennzahlen der LinTim-EANs.

Leider lagen die Rechenzeiten für den Matchings-Algorithmus über zwei Tagen, weshalb die Verfahren abgebrochen wurden. Da das Spielbeispiel wegen seiner geringen Größe wenig aussagekräftig für die spätere Auswertung ist, wurden für die weiteren Experimente die EANs gewählt, die aus den PTNs Bahn-Klein und Bahn-Groß mit den Verfahren Xpress, H6 und H7 gewonnen wurden.
Alle Warteaktivitäten dieser EANs wurden mit einer erlaubten Dauer von $[1, 3]$ versehen. In der Praxis üblich sind zwei Minuten Aufenthalt, die somit um jeweils eine Minute Spielraum nach unten und oben erweitert wurden.

4.2 Bibliotheken

Ebenso wenig, wie das Rad ständig neu erfunden werden muss, müssen für ein Programm alle Standardverfahren neu geschrieben werden. Durch Verwendung von Code-Bibliotheken kann auf verlässliche und schnell laufende Algorithmen zurückgegriffen werden, die den eigenen Code nicht nur kürzer, sondern auch stabiler werden lassen. Dieser Abschnitt stellt die verwendeten Bibliotheken vor.

Im Rahmen der Implementierung wurde ausschließlich auf Algorithmen und Datenstrukturen aus frei verfügbaren Bibliotheken zurückgegriffen. Dies waren die *Boost Library*, das *GNU Linear Programming Kit* (GLPK) und die *Goblin Library*.

4.2.1 Boost C++ Libraries

Die Homepage der Boost Library (Boo09) wirbt mit dem schmeichelhaften Zitat der C++ - Spezialisten Herb Sutter und Andrei Alexandrescu, sie wäre „one of the most highly regarded and expertly designed C++ library projects in the world“. Tatsächlich sind im Laufe der Zeit bereits einige Bestandteile der Boost Library in die C++ Standard Template Library (STL) aufgenommen worden, die wiederum im Großteil aller C++ - Anwendungen verwendet wird. Ein hervorhebendes Merkmal von Boost ist, dass fast alle Komponenten reine Header-Codes sind und keiner Kompilierung bedürfen.
Teil der Boost Library ist die auch separat verfügbare „Boost Graph Library“, die sich durch ihren stark generischen Aufbau auszeichnet. Sie bietet neben den nach Belieben den persönlichen Anforderungen anpassbaren Datenstrukturen eine Reihe von Algorithmen, unter anderem zur Suche nach kürzesten Wegen, minimalen spannenden Bäumen, Zusammenhangskomponenten, maximalen Flüssen und Graphenisomorphismen (dies allerdings nur primitiv). Sie bietet aufgrund ihrer hohen Flexibilität und bewährten Stabilität eine gute Grundlage für die Implementierung von Heuristiken zur Lösung des Fahrplanproblems.
Boost ist frei verfügbar entsprechend der „Boost Software License“. Verwendet wurde die Version 1.38.0.

4.2.2 GNU Linear Programming Kit

Wie der Name bereits verdeutlicht, handelt es sich bei GLPK um einen unter der GNU-Lizenz entwickelten Solver linearer Programme. Er gilt zwar als weniger effizient als seine kommerziellen Gegenstücke, ist dafür relativ klein im Umfang und als Teilkomponente dieses Programmes ausreichend. GLPK ist sowohl als Stand-Alone Solver verwendbar, als auch per *application programming interface* (API) integrierbar, was für die Programmierung in diesem Fall auch ausgenutzt wurde. Die Bibliothek wird im laufenden Algorithmus nur einmal angewandt, um eine Anfangslösung zu finden, wobei die Version 4.36 von (GLP09) für diese Experimente verwendet wurde.

4.2.3 Goblin Library

Die GOBLIN graph library ist eine unter der GNU Lesser Public Licence stehende C++ - Bibliothek, deren Schwerpunkt auf Optimierungsalgorithmen für Graphen liegt. Wie GLPK ist sie sowohl als Stand-Alone Executable verwendbar, als auch per API anbindbar, was entsprechend verwendet wurde. Die Version 2.8b28 von (Gob09) wurde in das Programm aufgenommen, da sie eine Implementation des Netzwerk-Simplex-Algorithmus nach Ahuja (AMO93), wie sie auch in dieser Arbeit beschrieben wurde, anbietet. Das ermöglichte das Auslesen des spannenden Baumes, der die optimale Lösung erzeugt und von größerem Interesse als der optimale Fluss selbst ist.

4.3 Programmaufbau

Es wird ein kleiner Einblick in den Aufbau der Implementation gegeben, indem einige wesentliche Funktionen näher erläutert werden. Anschließend wird darauf eingegangen, wie das NP-schwere Problem, eine Anfangslösung zu finden, gelöst wurde.

4.3.1 Wesentliche Funktionalitäten

Wie bereits erwähnt, wurden die in dieser Arbeit vorgestellten Algorithmen unter C++ programmiert. Da der Code insgesamt über 2500 Zeilen lang ist, kann das Programm selbstverständlich nicht in voller Länge hier vorgestellt werden; statt dessen werden nun einige wesentliche Funktionen präsentiert. Die Heuristik 10 zur Suche nach einem verbessernden Schnitt wurde nicht implementiert. Die Klasse `simplex` wird mit den nötigen Parametern über eine Funktion `init` initialisiert und stellt anschließend die Funktion `solve()` bereit, die den Fahrplan gemäß dem gewünschten Algorithmus berechnet. Eine entsprechende `main`-Funktion sieht dann folgendermaßen aus, wobei Verfahren, die bisher nicht präsentiert wurden, in Kapitel 5 näher erläutert werden:

```
#include "modulo_network_simplex.h"

using namespace std;
using namespace modulosimplex;

int main(int argc, char** argv)
{

simplex solver;

solver.init(argv[1], argv[2], 60);

switch(atoi(argv[3]))
{
case 1: //single node cuts
solver.set_loc_improvement(SINGLE_NODE_CUT);
break;
case 2: //multi node cuts
solver.set_loc_improvement(RANDOM_NODE_CUT);
break;
case 3: //waiting edge cuts
solver.set_loc_improvement(WAITING_CUT);
break;
default: exit(0); break;
}

switch(atoi(argv[4]))
{
case 1: //steepest decent
solver.set_tab_search(TAB_FULL);
break;
```

```
case 2: //tabu search
solver.set_tab_search(TAB_SIMPLE_TABU_SEARCH);
solver.set_ts_memory(atoi(argv[5]));
solver.set_ts_max_iterations(atoi(argv[6]));
break;
case 3: //simulated annealing
solver.set_tab_search(TAB_SIMULATED_ANNEALING);
solver.set_sa_init_temperature(atoi(argv[5]));
solver.set_sa_coolness_factor(atof(argv[6]));
break;
case 4: //simulated annealing steepest descent hybrid
solver.set_tab_search(TAB_STEEPEST_SA_HYBRID);
solver.set_sa_init_temperature(atoi(argv[5]));
solver.set_sa_coolness_factor(atof(argv[6]));
break;
case 5: //percentage
solver.set_tab_search(TAB_PERCENTAGE);
solver.set_percentage_improvement(atoi(argv[5]));
break;
case 6: //fastest
solver.set_tab_search(TAB_FASTEST);
solver.set_min_pivot_improvement(atof(argv[5]));
solver.set_dynamic_pivot_factor(atof(argv[6]));
solver.set_tab_min_improvement(DYNAMIC);
break;
default: exit(0); break;
}

solver.solve();
solver.write_result("timetable.tim");

return 0;

}
```

In dieser Variante werden die Parameter direkt über die Konsole mit eingegeben. Ein Aufruf

```
./network_simplex Activities.giv Events.giv 1 3 50000 0.95
```

startet das Verfahren mit dem EAN, das durch die Dateien `Activities.giv` und `Events.giv` gegeben wird, im Modus Simulated Annealing (`3`) mit einer Starttemperatur von 50.000, einem Kühlungsfaktor von $0,95$ und Single Node Cuts (`1`) als lokale Verbesserung.
Die verfügbaren Algorithmen, um einen Fundamentalschnitt zu finden, sind in einem `enum` codiert, der den Aufruf erleichtert.

```
enum TABLEAU_SEARCH
{
//Takes the best of all neighbours.
TAB_FULL = 1,
```

```
//Sorts the columns so that the shortest will be used first
//in the search. Takes the first fundamental cut that is
//good enough according to the min_pivot_improvement percentage.
TAB_FASTEST = 2,

//Sorts the columns so that the shortest will be used first
//in the search. Takes the best fundumental cut out of the
//first percentage_improvement percentage.
TAB_PERCENTAGE = 3,

//Uses simulated annealing to find the next pivot step. Parameters
//are sa_temperature and sa_cooling_factor.
TAB_SIMULATED_ANNEALING = 4,

//Tabu search.
TAB_SIMPLE_TABU_SEARCH = 5,

//Uses steepest descend until it arrives at the local optimum,
//then switches to simulated annealing.
TAB_STEEPEST_SA_HYBRID = 6
};
```

Ebenso die implementierten lokalen Schnittverfahren.

```
enum LOCAL_IMPROVEMENT
{
SINGLE_NODE_CUT = 1,
MULTI_NODE_CUT = 2,
WAITING_CUT = 3
};
```

Der `solve()`-Aufruf muss nun nichts weiter tun, als abwechselnd den gewünschten Fundamentalschnitt-Algorithmus zu verwenden, bis er in ein lokales Optimum gerät, um dann den gewünschten Node-Cut zu suchen. Der wesentliche Aufbau ist also folgendermaßen:

```
void simplex::solve()
{

cut.active = false;
do
{
if (cut.active)
{
//Cut found, applying it.
transform();

//Solving non-periodic flow.
```

```
non_periodic();

//Building simplex tableau.
build();
}

//Pivoting.
while(pivot())
{
};

//Searching for an improving cut.
improvable();
}
while (cut.active);
}
```

Über eine boolesche `verbose`-Variable kann gesteuert werden, ob nur die Zielfunktionswerte jeder Iteration oder nähere Informationen ausgegeben werden sollen. Ist der Algorithmus beendet, wird der Fahrplan standardmäßig in eine Datei `timetable.tim` geschrieben.

4.3.2 Finden einer Anfangslösung

Bevor es allerdings so weit kommen kann, muss zunächst eine zulässige Anfangslösung gefunden werden. Nachtigall und Opitz (NO08) empfehlen einen constraint-propagation-Ansatz, der zu jedem Knoten eine Menge von potentiell zulässigen Zeitpunkten bestimmt. Als Beispiel diene die Probleminstanz, die durch Abbildung 4.2 mit Periode 10 gegeben ist.

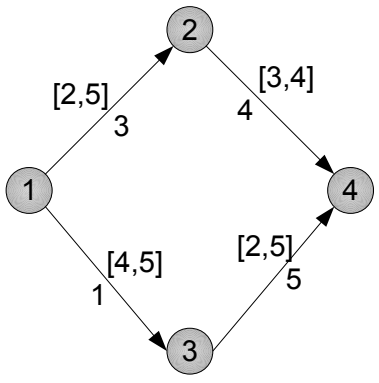
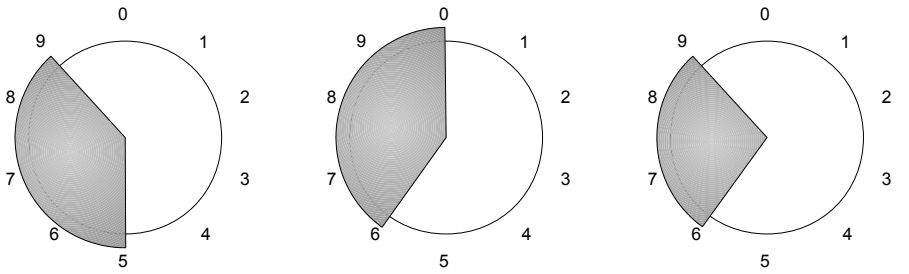


Abbildung 4.2: Eine (PF)-Probleminstanz.

Wird für Knoten 1 der Zeitpunkt 0 festgelegt, werden nun die Pfade untersucht, die von dort aus zu weiteren Knoten führen. Über den Pfad 1-2-4 zum Beispiel kann der Zeitpunkt von Knoten 4 eingeschränkt werden. Über den Pfad 1-3-4 wird eine weitere Bedingung gefunden und mit der vorherigen geschnitten.



(a) Bedingung 1-2-4. (b) Bedingung 1-3-4. (c) Resultierende Bedingung

Abbildung 4.3: Fortpflanzung von Bedingungen.

Das Problem bleibt nun dadurch schwer, dass die Wahl eines Zeitpunktes, auch wenn er im potentiell zulässigen Bereich liegt, die Bereiche von anderen Knoten beeinflussen wird und so zur Unzulässigkeit führen kann. Es wird also *rekursiv* nach einem zulässigen Fahrplan gesucht.
Im Rahmen dieses Programmes wurde ein alternativer Ansatz gewählt. Da es die Kreise sind, die in der Formulierung (PF') die Bedingungen des linearen Programmes bilden, kann die Lösung mit einem Solver beschleunigt werden, indem die Anzahl der Kreise im Graphen reduziert wird. Indem nur Kanten (i, j) entfernt werden, die eine Dauer $u_{ij} - l_{ij} = T - 1$ besitzen, kann garantiert werden, dass die so gewonnene Lösung auch im Original zulässig ist. Es ist üblich, dass Umsteigeaktivitäten nur mit einer Minimaldauer belegt werden, die die Zeit darstellen, die zum Gleiswechsel nötig ist, aber auf eine Maximalzeit verzichtet wird, da sich die Dauer bereits durch die Zielfunktion regulieren wird und es de facto nicht *unmöglich* ist, an einem Bahnhof länger zu warten.
Deshalb werden vom eingelesenen Graphen zunächst die Umsteigekanten entfernt, woraufhin das resultierende EAN schnell vom GLPK-Solver gelöst werden kann. Es werden anschließend die Modulo-Parameter der Lösung ausgelesen, eine duale Probleminstanz als GOBLIN-Graph erzeugt und mittels dem klassischen Netzwerk-Simplex in Bezug auf diese Parameter optimal gelöst. Die Anfangslösung für das Modulo-Netzwerk-Simplex Verfahren ist gewonnen.

Kapitel 5

Empirische Verfahren und Auswertung

5.1 Allgemeines

Nach der Vorstellung der verwendeten Rechner wird gezeigt, wie sichergestellt wurde, dass die EANs nur eine Zusammenhangskomponente besitzen.

Alle Versuche wurden auf Linux Rechnern mit 12 GB Arbeitsspeicher und zwei 64 Bit Dual Core AMD Opteron Prozessoren mit jeweils 2000 MHz Taktfrequenz durchgeführt. Der durchschnittliche Bedarf an Arbeitsspeicher lag lediglich bei circa 50 MB, was deutlich macht, dass die entstehenden Laufzeiten vor allem von der Prozessorleistung und nicht vom verfügbaren Arbeitsspeicher beeinflusst werden. Da der Programmcode nicht auf eine Parallelisierung ausgelegt war, wurde vom Verfahren jeweils nur ein einzelner Kern angesprochen, weshalb gleich vier Prozesse parallel gestartet wurden. Die Kompilierung erfolgte durch den GCC C++ Compiler in Version 4.2.4.
Überraschenderweise wurde bei den ersten Programmdurchläufen kein spannender Baum durch die Goblin Library als Anfangslösung zurückgegeben. Nähere Betrachtungen zeigten, dass die durch LinTim erzeugten EANs *nicht zusammenhängend* sind. In der Praxis wäre das ein Glücksfall, da jede Zusammenhangskomponente für sich gelöst werden kann und somit ein großes Problem in mehrere Teilprobleme zerfällt. Um die interessanten Problemgrößen zu erhalten, wurden für diese Arbeit statt dessen n Zusammenhangskomponenten durch $n-1$ Kanten mit Gewicht 0 und Dauer $[0,0]$ verbunden. Da eine solche Kante nicht Teil eines Fundamentalkreis sein kann, ist die entsprechende Tableau-Spalte leer und es wird die Laufzeit nicht beeinflusst. Tabelle 5.1 gibt die Anzahl der Zusammenhangskomponenten wider.

Es lässt sich deutlich ablesen, dass schlechtere Linienkonzepte im Sinne der Anzahl der entstehenden Fahrten auch häufiger in Zusammenhangskomponenten zerfallen. Das liegt daran, dass derselbe Passagierbedarf durch größere Netzwerke verteilt wird, wodurch häufiger Kanten ohne Gewicht auftreten.

Instanz	Anzahl Zusammenhangskomponenten
Xpress Groß	1
H7 Groß	7
H6 Groß	57
Xpress Klein	7
H7 Klein	19
H6 Klein	33

Tabelle 5.1: Zusammenhangskomponenten der EANs.

5.2 Fundamentalschnitt-Variationen

In diesem Abschnitt werden die Ergebnisse zu verschiedenen Fundamentalschnitt-Suchverfahren auf den bereits vorgestellten sechs EANs widergegeben. Begonnen wird mit dem Verfahren des steilsten Abstiegs, das mit der Originalvariante von Nachtigall und Opitz (NO08) übereinstimmt; es folgen die Verfahren „Fastest“ und „Percentage“, die versuchen, sich die Struktur des Simplex-Tableaus zunutze zu machen. Nach der anschließenden Auswertung von Tabu Search und Simulated Annealing wird ein Mischverfahren besprochen, das versucht, die Vorteile des steilsten Abstiegs und des Simulated Annealings zu kombinieren.

5.2.1 Modus Steilster Abstieg

Die Auswertung wird begonnen mit dem Verfahren des steilsten Abstiegs, das in jeder Iteration alle benachbarten Spannender-Baum-Strukturen untersucht und diejenige auswählt, die den kleinsten Zielfunktionswert liefert.
Anhand folgenden Beispiels, das an das EAN aus (NO08) angelehnt ist, werden einige wesentliche Eigenschaften verdeutlicht. Abbildung 5.1 skizziert eine (PF)-Instanz, auf die das Verfahren angewandt werden soll.

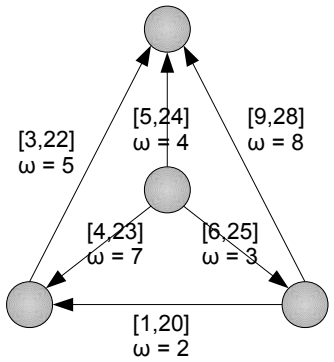


Abbildung 5.1: Ein Beispiels-EAN.

Der Einfachheit halber werden nur Spannender-Baum-Strukturen betrachtet,

für die $\mathcal{T}_U = \emptyset$ ist. Da es im Graphen vier Kreise der Länge drei und insgesamt $\binom{6}{3} = 20$ Möglichkeiten, eine dreielementige Kantenmenge zu wählen, gibt, muss es also genau $20 - 4 = 16$ Spannender-Baum-Strukturen geben. In Abbildung 5.2 wird der gesamte zulässige Suchbereich dargestellt und nummeriert. Die Nachbarschaftsverhältnisse werden von Abbildung 5.3 im oberen Teil dargestellt: Die Knoten tragen Nummern, die der vorherigen Nummerierung entsprechen, und eine Kante wird genau dann gezogen, wenn zwei Bäume durch Austauschen einer Kante auseinander hervorgehen. Im unteren Teil der Abbildung 5.3 finden sich statt der Baumnummern nun die Funktionswerte der zugehörigen Lösungen.

Startet das Verfahren zum Beispiel mit dem spannenden Baum Nummer 13, so werden anschließend die Bäume 12 und 16 besucht. Baum 16 ist ein lokales Minimum bezüglich der Basiswechsel, doch ist der dazugehörige Zielfunktionswert von 149 geringer als der Wert 94, der durch Baum Nummer 10 erzeugt wird. Dieses Beispiel deutet darauf hin, dass gilt:

1. Die Nachbarschaftsmenge eines Zustandes kann sehr groß sein.
2. Durch wenige Iterationen kann ein lokales Optimum erreicht werden.
3. Der Zielfunktionswert ist nicht „stetig“ in dem Sinne, dass benachbarte Lösungen ähnliche Zielfunktionswerte erzeugten.

Die große Menge an Nachbarn ist es entsprechend, was das Verfahren am meisten verlangsamt. Ist ein EAN mit n Ereignissen und m Kanten gegeben, so besitzt das Simplex-Tableau in jeder Iteration eine Größe von $m - n + 1 \times n - 1$ Einträgen. Für jeden Eintrag, der von Null verschieden ist, muss die Veränderung im Zielfunktionswert berechnet werden, was wiederum linear in der Anzahl der Einträge der entsprechenden Spalte geschieht. Im schlechtesten Fall eines vollbesetzten Tableaus sind somit

$$\begin{aligned} & c \cdot (m - n + 1) \cdot (n - 1) \cdot (m - n + 1) \\ =& c \cdot (m^2 n - 2mn^2 + 4mn - m^2 - 2m + n^3 - 3n^2 + n - 1) \\ =& \mathcal{O}(m^2 n + n^3) \end{aligned}$$

Operationen, also kubisch viele, nötig, um den nächsten Iterationsschritt zu berechnen.
Es folgen nun Diagramme, die den Zielfunktionswert zu jedem Iterationsschritt darstellen. Die Markierungen heben diejenigen Stellen hervor, an denen kein verbessernder Fundamentalschnitt gefunden und stattdessen ein Single Node Cut angewandt wurde.

Die Tabellen 5.2 und 5.3 fassen die wesentlichen Daten zusammen. Als Iteration wird jeder Basiswechsel gezählt; Single Node Cuts zählen nicht als Iteration. Die Anfangslösung ist Iteration 1.

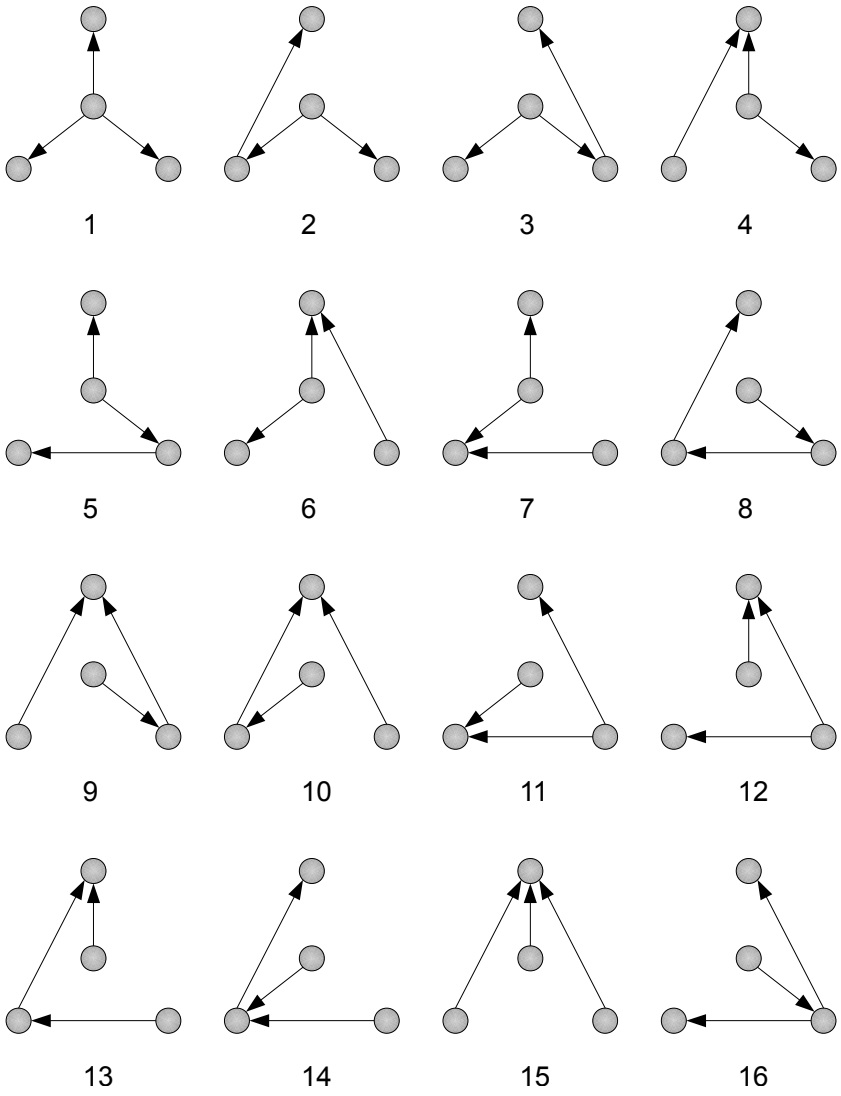


Abbildung 5.2: Eine Liste aller zulässigen Lösungen.

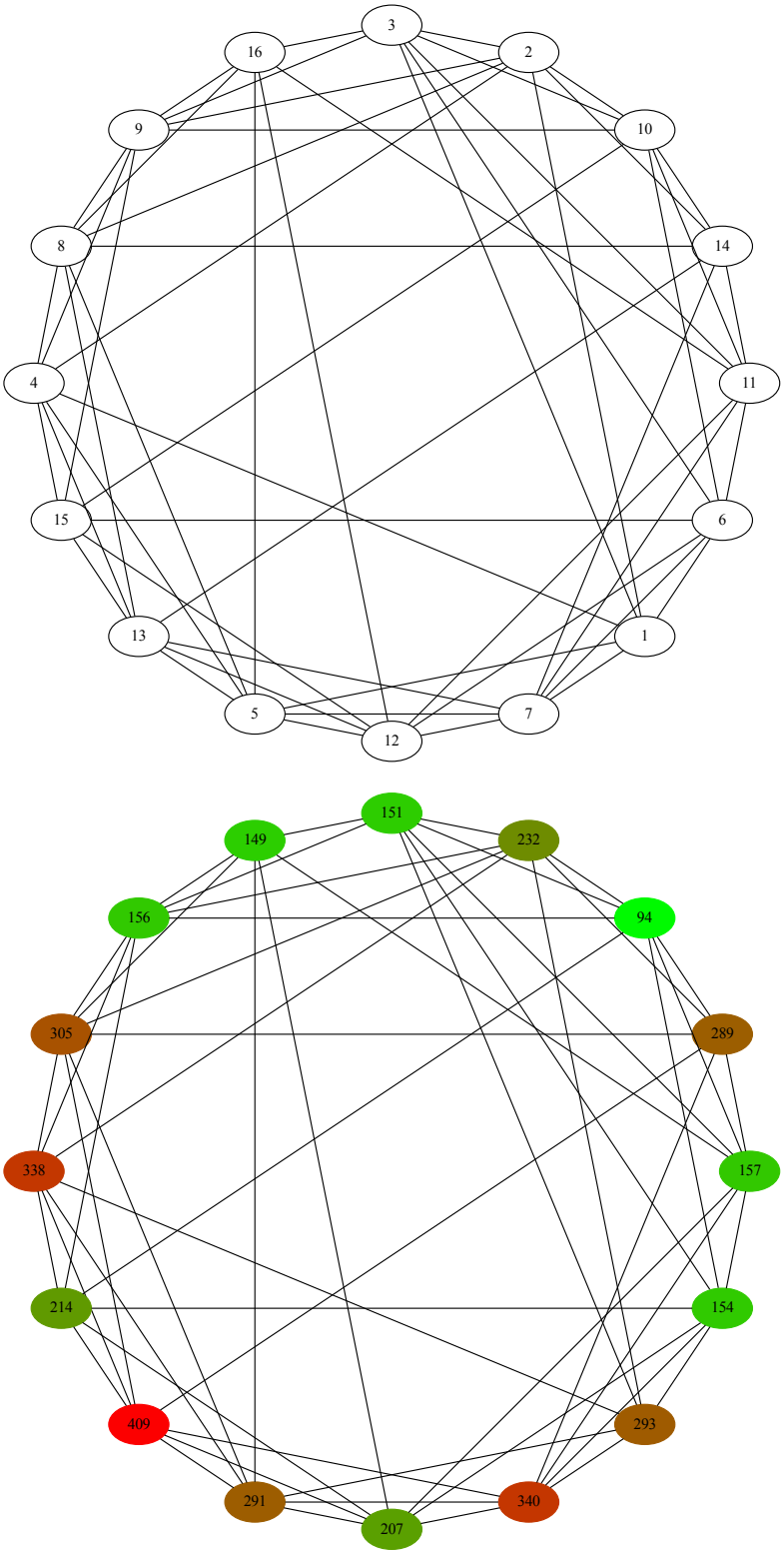
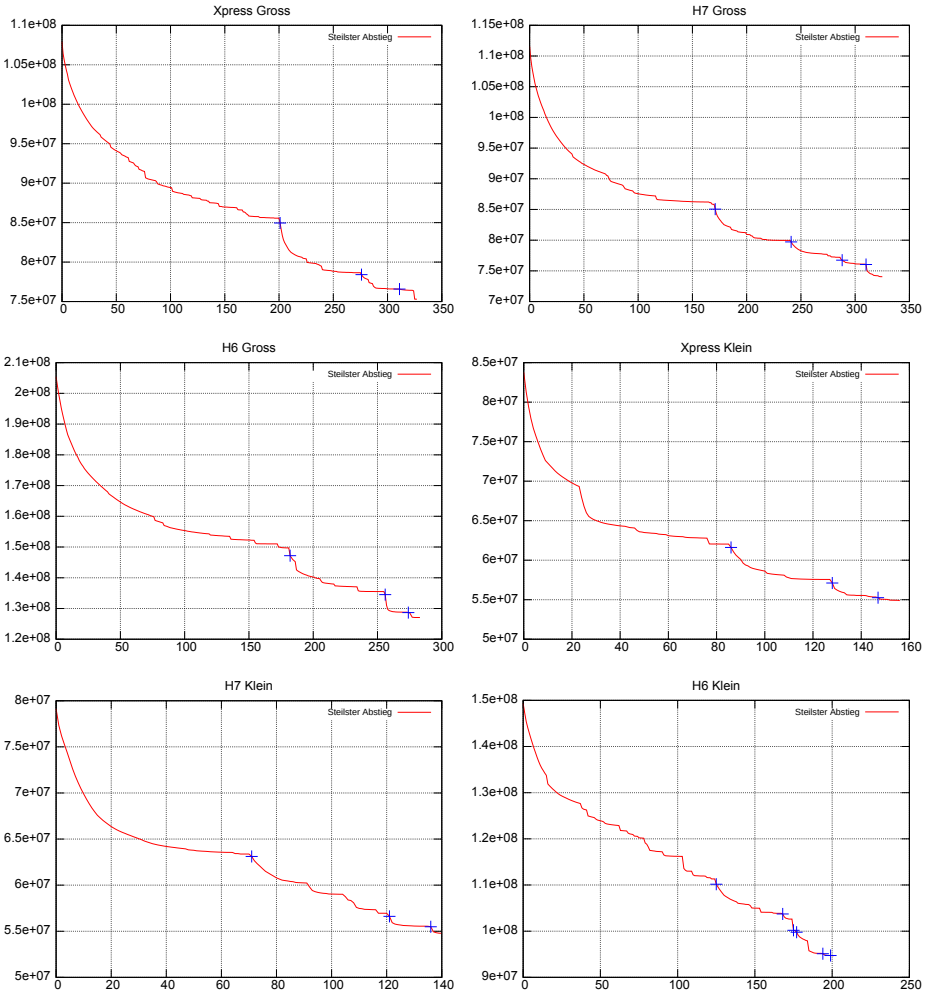


Abbildung 5.3: Nachbarschaften und Zielfunktionswerte der Lösungen.



Instanz	Anfangslösung	Endlösung	Verbesserung
Xpress Groß	107.928.736	75.301.572	30,2 %
H7 Groß	111.583.840	74.048.827	33,6 %
H6 Groß	205.388.540	127.071.960	38,1 %
Xpress Klein	83.798.525	54.890.924	34,5 %
H7 Klein	79.119.703	54.780.460	30,8 %
H6 Klein	151.450.954	94.666.655	37,5 %

Tabelle 5.2: Daten zu Modus Steilster Abstieg.

Instanz	Anzahl single node cuts	Anzahl Iterationen	Rechenzeit in s	Zeit in s pro Iteration
Xpress Groß	3	325	14.918	45,9
H7 Groß	4	322	13.974	43,4
H6 Groß	3	281	9.770	34,8
Xpress Klein	3	154	7.920	51,4
H7 Klein	3	138	2.326	16,9
H6 Klein	6	196	1.657	8,5

Tabelle 5.3: Daten zu Modus Steilster Abstieg

5.2.2 Modus Fastest

Die Überlegungen aus Abschnitt 5.2.1 zeigen, dass Spalten des Simplex-Tableaus mit vielen Einträgen mehr Zeit pro Eintrag benötigen, um die Veränderung der Zielfunktion zu berechnen, als Spalten mit weniger Einträgen, da die benötigte Zeit linear in der Anzahl der Einträge wächst. Das legt ein beschleunigtes Verfahren nahe, das zunächst alle Spalten des Tableaus nach ihrer Größe sortiert und dann von der kleinsten zur größten Spalte sucht. Würden nach wie vor alle Spalten durchsucht werden, ginge der Geschwindigkeitsvorteil, den dieses Verfahren gegenüber dem steilsten Abstieg besitzt, wieder verloren. Daher wird *der erste verbessernde Schnitt* gewählt.
Da durch Basiswechsel mit nur sehr kleiner Verbesserung des Zielfunktionswertes unerwünscht viele Iterationen die Folge sind, die die Laufzeit zu sehr belasten, wird ein Kriterium eingeführt, das bestimmt, dass ein Wechsel nur gewählt wird, wenn er den Wert der bestehenden Lösung um eine bestimmte prozentuale Größe übertrifft.
Die Diagramme des vorherigen Abschnitts haben gezeigt, dass die Verbesserungen des Zielfunktionswertes mit zunehmenden Iterationen bis auf vereinzelte Spitzen geringer werden. Abbildung 5.4 zeigt die Verbesserung jeder Iteration im Verfahren des steilsten Abstiegs für das EAN „Xpress Groß“ bis zum ersten Single Node Cut.

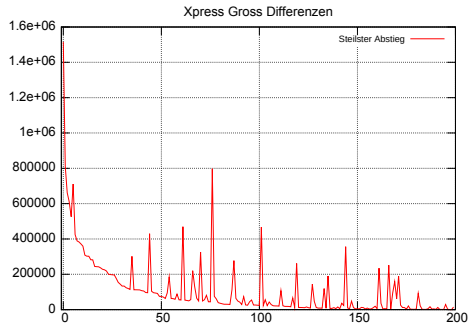
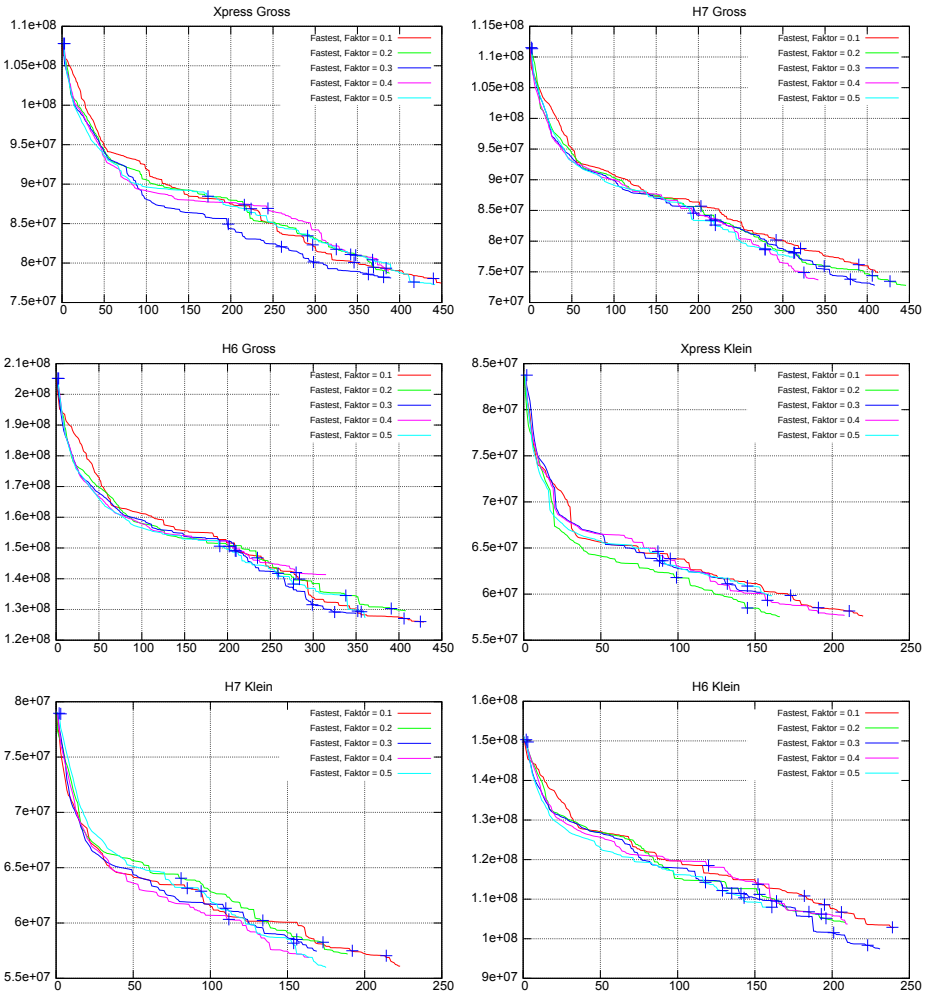


Abbildung 5.4: Verbesserungen der Basiswechsel im steilsten Abstieg auf Xpress Groß

Daher wird das prozentuale Kriterium, ob ein Schritt als hinreichend gut akzeptiert wird oder nicht, nicht fixiert gehalten, sondern dynamisch angepasst. Es wird immer dann, wenn kein Nachbar mehr gefunden wird, der das augen-

blickliche Kriterium erfüllen kann, ein fester Faktor $f < 1$ auf den Wert des Kriteriums multipliziert. Sollte auch dann kein hinreichend guter Zug gefunden werden, gilt der Pivotisierungsalgorithmus als beendet und es wird nach einem Single Node Cut gesucht.
Für die Auswertung wurde mit einem Kriterium von 10% Mindestverbesserung begonnen, das anschließend um einen Faktor 0,1, 0,2, 0,3, 0,4 oder 0,5 reduziert wird. In den Diagrammen lässt sich durch das „+“ ablesen, dass teilweise gleich zu Beginn des Algorithmus nach Single Node Cuts gesucht werden musste, wenn der Relaxationsfaktor zu groß gewählt wurde.



Es werden nun die jeweils erfolgreichsten Vertreter des Verfahrens tabellarisch zusammengefasst.

Instanz	Anfangslösung	Endlösung	Verbesserung
Xpress Groß, $f = 0{,}5$	107.928.736	77.273.645	28,4 %
H7 Groß, $f = 0{,}2$	111.583.840	72.791.077	34,8 %
H6 Groß, $f = 0{,}1$	205.388.540	125.862.290	38,7 %
Xpress Klein, $f = 0{,}2$	83.798.525	57.548.162	31,3 %
H7 Klein, $f = 0{,}5$	79.119.703	55.978.270	29,2 %
H6 Klein, $f = 0{,}3$	151.450.954	97.405.627	35,7 %

Tabelle 5.4: Auswahl an Daten zu Modus Fastest.

Instanz	Anzahl single node cuts	Anzahl Iterationen	Rechenzeit in s	Zeit in s pro Iteration
Xpress Groß, $f = 0{,}5$	6	437	4.626	10,6
H7 Groß, $f = 0{,}2$	6	441	5.398	12,2
H6 Groß, $f = 0{,}1$	5	425	10.910	25,7
Xpress Klein, $f = 0{,}2$	2	165	884	5,4
H7 Klein, $f = 0{,}5$	4	173	791	4,6
H6 Klein, $f = 0{,}3$	4	228	898	3,9

Tabelle 5.5: Auswahl Daten zu Modus Fastest.

Der Vergleich zum Verfahren des steilsten Abstiegs zeigt, dass die Qualität der Lösungen ähnlich zueinander ist - häufig etwas schlechter, teilweise sogar besser -, die Laufzeit des Verfahrens dagegen ungefähr gedrittelt wird, wobei die EANs H6 Groß und Xpress Klein stark davon abweichen. Wie zu erwarten erhöht sich die Anzahl der benötigten Iterationen. Wie groß der Relaxationsfaktor gewählt werden sollte, hängt offenbar stark vom EAN und auch vom Zufall ab. Es lässt sich keine grundsätzliche Überlegenheit von Verfahren mit geringem Faktor ablesen, wie naiv zu erwarten wäre, da so besonders stark verbessernde Pivots gewählt werden.

5.2.3 Modus Percentage

Wie im Abschnitt 5.2.1 bereits gezeigt wurde, hängt die Dauer der Prüfung einer Spalte des Simplex-Tableaus *quadratisch* von der Anzahl ihrer Einträge ab. Das macht sich der Modus „Fastest“ zunutze, indem er die Spalten nach der Anzahl ihrer Einträge sortiert. Experimentelle Versuche zeigen, dass insbesondere ein kleiner Teil der Spalten den Großteil der Laufzeit verursacht.
Um diese Behauptung zu stützen, werden die Simplex-Tableaus von 100 Iterationen im Simulated Annealing-Modus für Xpress Groß mit hoher Temperatur ausgewertet, da hier die Schritte zufällig gewählt werden und so die Repräsentativität erhöht wird. Es wird zu jeder Spalte gezählt, welche Größe sie besitzt. Das Diagramm 5.5 zeigt an, wieviele Einträge in den 10% kleinsten Spalten gezählt wurden, wieviele in den $11\% - 20\%$ kleinsten etc. Es ist zu beachten, dass die y-Skala logarithmisch gewählt wurde.

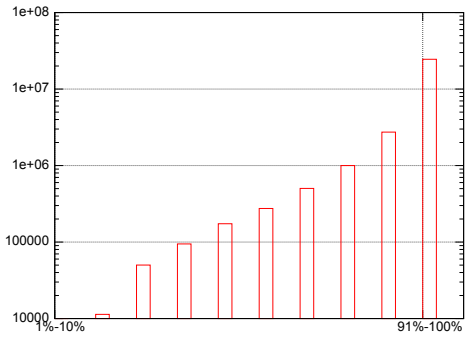
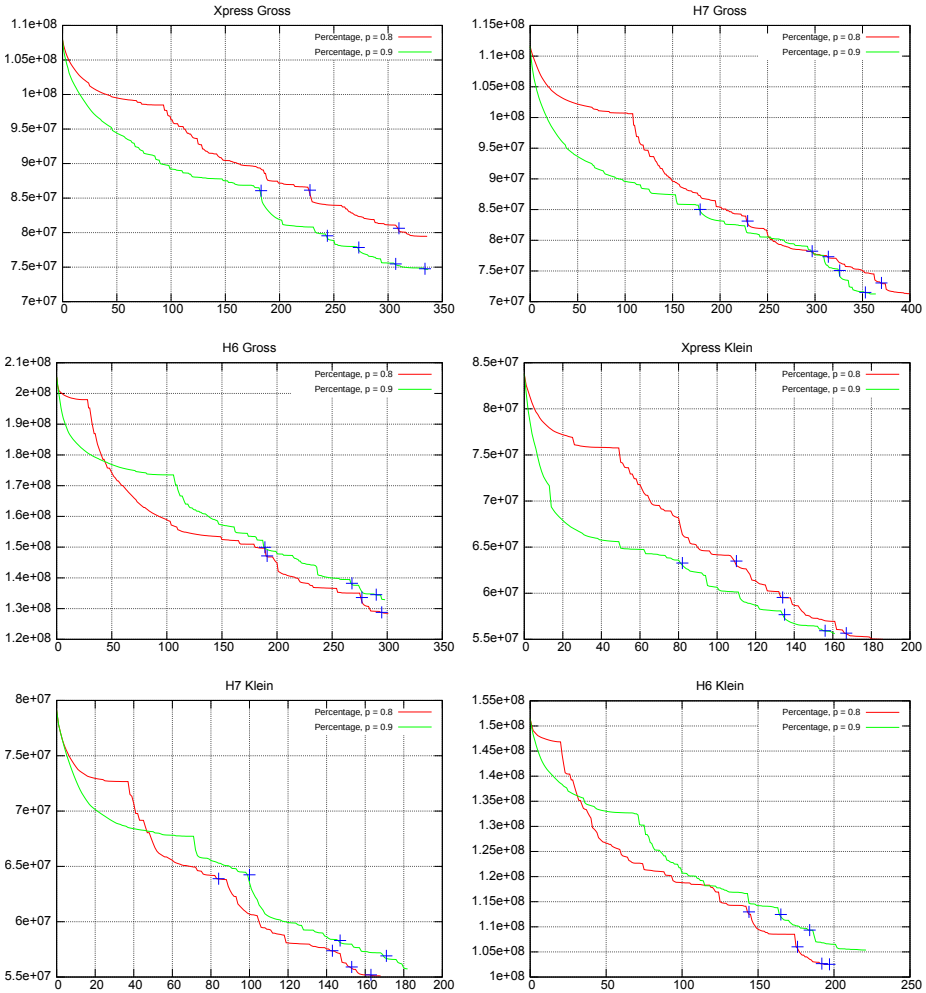


Abbildung 5.5: Anzahl der Einträge von 100 Simplex-Tableaus den relativen Spaltengrößen nach.

Der Wert für den Eintrag $1\% - 10\%$ liegt bei genau 0 - das heißt, innerhalb der kleinsten 10% aller Spalten gibt es keine Einträge. Das ist derart zu interpretieren, dass die entsprechenden Nichtbaumkanten in überhaupt keinem Kreis liegen.
Es lässt sich ablesen, dass die größten 10% aller Spalten über 80% aller Einträge beinhalten. Das motiviert ein Verfahren ähnlich dem Modus „Fastest“, bei dem nicht der Sortierung nach gesucht wird, bis ein Basiswechsel einem Kriterium genügt, sondern es wird ein fester, schnell durchsuchbarer prozentualer Anteil der Spalten nach der dort besten Lösung durchsucht. Das bedeutet zwar, dass der Großteil der möglichen Basiswechsel ignoriert wird, gleichzeitig kann aber der Großteil aller Baumkanten untersucht werden. Für jedes EAN wurde dieses Verfahren einmal mit $p = 0{,}8$ und einmal mit $p = 0{,}9$ durchgeführt - das heißt, es wurde jeweils 80%, bzw. 90%, der Spalten durchsucht. Die anschließenden Tabellen zeigen wieder die Werte des besseren Vertreters an.

Es ist überraschend, dass dasjenige Verfahren, bei dem der Suchbereich *weiter eingeschränkt* ist, häufiger das bessere Ergebnis erzielen konnte als der Vertreter mit größerem Suchbereich. Dies ist eine schöne Illustration der „Unvorhersagbarkeit“ einer Black Box, da anscheinend schlechter verlaufende Suchwege zu ganz anderen Lösungen führen können. Wie erwartet hat sich die Laufzeit gegenüber dem steilsten Abstieg mit Ausnahme von H6 Groß verbessert, wobei auch hier überrascht, dass die Lösungen teilweise sogar besser geworden sind.



Instanz	Anfangslösung	Endlösung	Verbesserung
Xpress Groß, $p = 0{,}9$	107.928.736	74.672.422	30,8 %
H7 Groß, $p = 0{,}9$	111.583.840	71.249.741	36,1 %
H6 Groß, $p = 0{,}8$	205.388.540	128.488.864	37,4 %
Xpress Klein, $p = 0{,}8$	83.798.525	55.001.497	34,4 %
H7 Klein, $p = 0{,}8$	78.044.361	55.098.459	29,4 %
H6 Klein, $p = 0{,}8$	151.450.954	102.530.560	32,3 %

Tabelle 5.6: Auswahl an Daten zu Modus Percentage.

Instanz	Anzahl single node cuts	Anzahl Iterationen	Rechenzeit in s	Zeit in s pro Iteration
Xpress Groß, $p = 0{,}9$	5	333	4.194	12,6
H7 Groß, $p = 0{,}9$	4	361	4.679	13,0
H6 Groß, $p = 0{,}8$	3	299	10.800	36,1
Xpress Klein, $p = 0{,}8$	3	184	1.214	6,6
H7 Klein, $p = 0{,}8$	4	165	1.873	11,4
H6 Klein, $p = 0{,}8$	4	193	1.562	8,1

Tabelle 5.7: Auswahl Daten zu Modus Percentage.

5.2.4 Modus Tabu Search

Die Verwendung von Tabu Search wird wesentlich durch zwei Herausforderungen erschwert:

1. Es gibt viele Basiswechsel, die den Zielfunktionswert nicht beeinflussen. Einerseits ist das offensichtlich der Fall, wenn eine Kante, dessen untere und obere Schranke übereinstimmen, gegen eine weitere solche Kante ausgetauscht wird, andererseits tritt das auch ein, wenn ein Basiswechsel durch einen „0-Schnitt“ vollzogen werden kann, wenn also die Dauer einer Nichtbaumkante mit der einer Baumkante bereits übereinstimmt. Drittens sind die betrachteten Netzwerke von einer solchen Größe, dass es möglich ist, dass auch ein nichtentarteter Schnitt keine Veränderung des Zielfunktionswertes erwirkt, da sich die Gewichte gegenseitig zu 0 summieren.

2. Wird in jeder Iteration der betrachtete Zustand mit allen Zuständen der unter Umständen sehr langen Tabu Liste verglichen, braucht jede Pivotoperation sehr lange Rechenzeit.

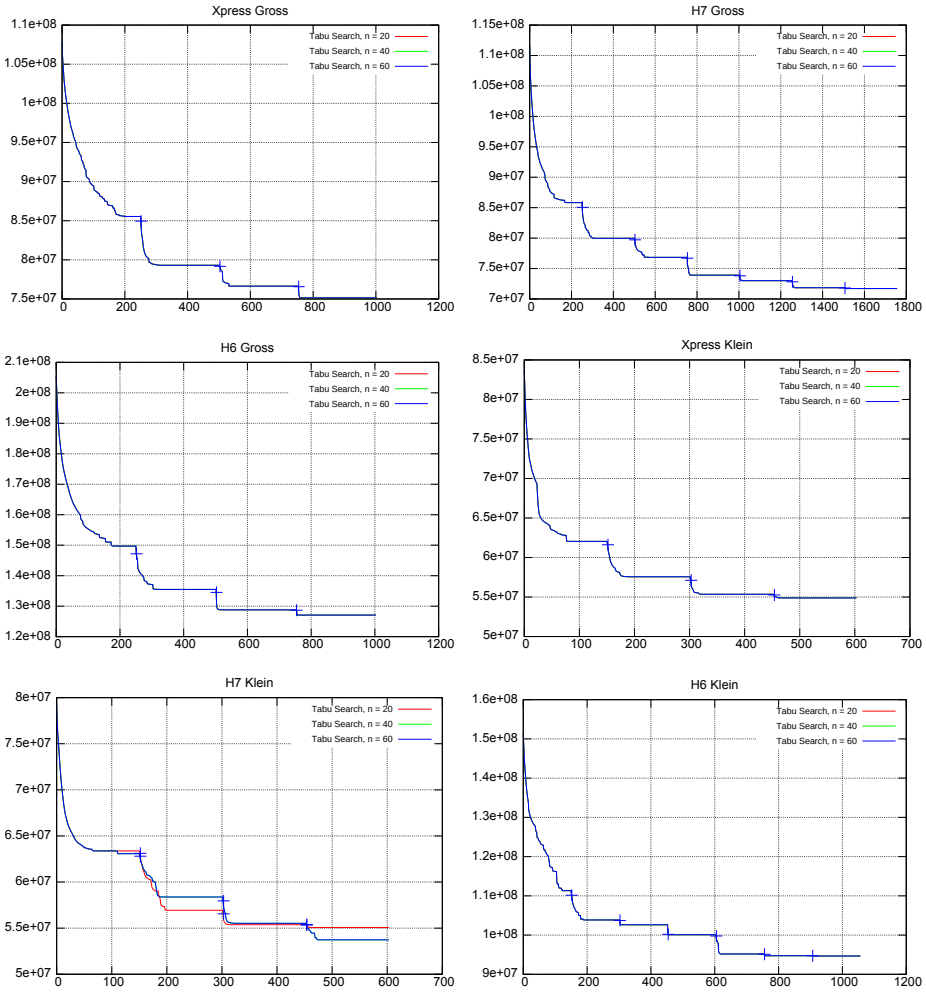
Die Basiswechsel, die den Zielfunktionswert nicht verändern, behindern insofern das Verfahren, dass im lokalen Optimum nur auf einem *Plateau* gewandert wird. Auch wenn es weiterhin theoretisch möglich ist, kann sich Tabu Search in der Praxis nicht genug von der ursprünglichen Lösung entfernen, um ein neues lokales Optimum finden. Daher werden *alle Züge*, die den Zielfunktionswert nicht verändern, *in die Tabu Liste mit aufgenommen.*
Um die aufwändigen Vergleiche in der Tabu Liste zu umgehen, werden die Listeneinträge als `struct` gespeichert:

```
struct ts_memory
{
std::set<unsigned short> lower_tree;
std::set<unsigned short> upper_tree;

uint lower_checksum;
uint upper_checksum;
};
```

Sets werden für gewöhnlich von der STL-Library als binäre Suchbäume gespeichert, liegen also in sortierter Form vor und können schnell verglichen werden.

Zu einer Spannender-Baum-Struktur $\mathcal{T} = (\mathcal{T}_L, \mathcal{T}_U)$ werden zudem die aufsummierten Kantennummern aus $\mathcal{T}_L$ und$\mathcal{T}_U$ in eigenen Variablen gespeichert. Vor der Suche nach einem Basiswechsel werden diese Summen auch für die momentane Lösung berechnet. Ein Basiswechsel kann dann leicht durch Veränderung der beiden `checksums` simuliert werden und nur bei Gleichheit müssen die gesamten Strukturen verglichen werden.
Es folgen nun die Ergebnisse des Algorithmus mit Listenlängen $n = 20$, $n = 40$ und $n = 60$.



Im EAN H7 Klein tritt der einzige Fall auf, dass ein spürbarer Unterschied zwischen den verschiedenen Listenlängen erkennbar ist. Für $n = 60$ und, was im Plot durch diesen Fall verdeckt wird, auch für $n = 40$ wird das Plateau ohne einen Single Node Cut verlassen (um, was die Unverhersagbarkeit des Problems noch einmal deutlich unterstreicht, kurz darauf vom scheinbar schlechteren Verfahren mit $n = 20$ wieder eingeholt zu werden). Die Plateaus, die durch den Maßstab als waagerechte Linien erscheinen, sind im Detail betrachtet zyklisch wiederkehrene Zustände, die sich in Hinsicht auf ihren Zielfunktionswert nur minimal unterscheiden. Das Diagramm 5.6 zeigt dieses Verhalten anhand eines Ausschnitts. Die Zielfunktionswerte wiederholen sich alle $27 > 20 = n$ Iterationen.

Instanz	Anfangslösung	Endlösung	Verbesserung
Xpress Groß, $n = 20$	107.928.736	75.138.267	30,4 %
H7 Groß, $n = 20$	111.583.840	71.709.567	35,7 %
H6 Groß, $n = 20$	205.388.540	127.071.960	38,1 %
Xpress Klein, $n = 20$	83.798.525	54.890.924	34,5 %
H7 Klein, $n = 40$	79.119.703	53.732.125	32,1 %
H6 Klein, $n = 20$	151.450.954	94.666.655	37,5 %

Tabelle 5.8: Auswahl an Daten zu Modus Tabu Search.

Instanz	Anzahl single node cuts	Anzahl Iterationen	Rechenzeit in s	Zeit in s pro Iteration
Xpress Groß, $n = 20$	3	1.001	57.872	57,8
H7 Groß, $n = 20$	6	1.751	70.156	40,1
H6 Groß, $n = 20$	3	1.001	37.777	37,7
Xpress Klein, $n = 20$	3	601	14.915	24,8
H7 Klein, $n = 40$	3	601	7.614	12,7
H6 Klein, $n = 20$	6	1.051	7.439	7,1

Tabelle 5.9: Auswahl Daten zu Modus Tabu Search.

Abbildung 5.6: Aussschnitt aus Tabu Search.

5.2.5 Modus Simulated Annealing

Ähnlich wie bei Tabu Search wird das Simulated Annealing-Verfahren durch große Tableaus erschwert, da die Temperatur abkühlt, ohne dass sich die Qualität der Lösung verbessert. Daher werden auch hier alle Pivotisierungen blockiert, die den Zielfunktionswert *nicht verändern*. Es wurden zwei Testreihen mit einer geometrischen Abkühlung durchgeführt. In der ersten wurde der Kühlungsfaktor c auf 0,95 gesetzt, in der zweiten auf 0,98. Verwendet wurden jeweils die Starttemperaturen 50.000, 100.000, 150.000, 200.000 und 250.000. Die Tabellen geben wie zuvor den jeweils besten Vertreter wieder.

Wie zu erwarten werden gerade zu Beginn des Verfahrens Fundamentalschnitte gewählt, die den Zielfunktionswert verschlechtern. In Abbildung 5.7 wird ein Ausschnitt vergrößert wiedergegeben, an dem sich dieses Verhalten deutlicher ablesen lässt.

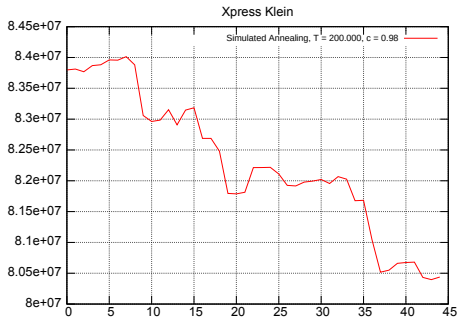
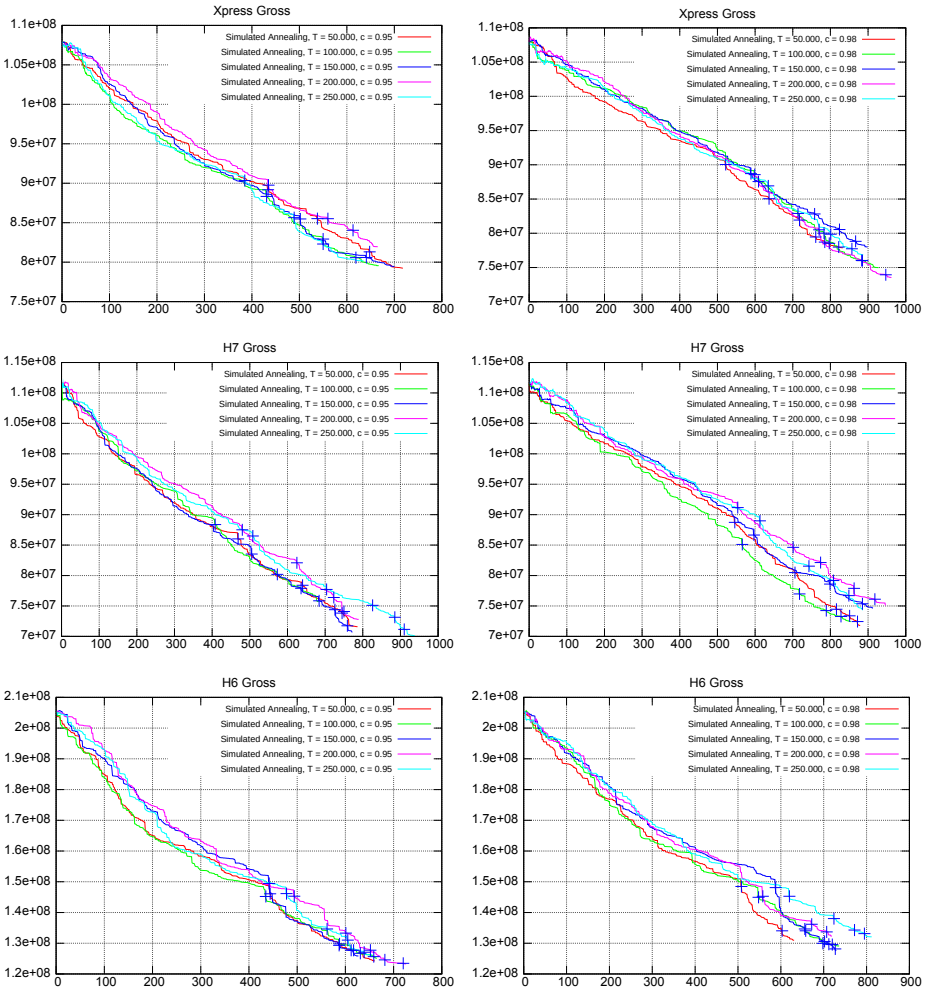


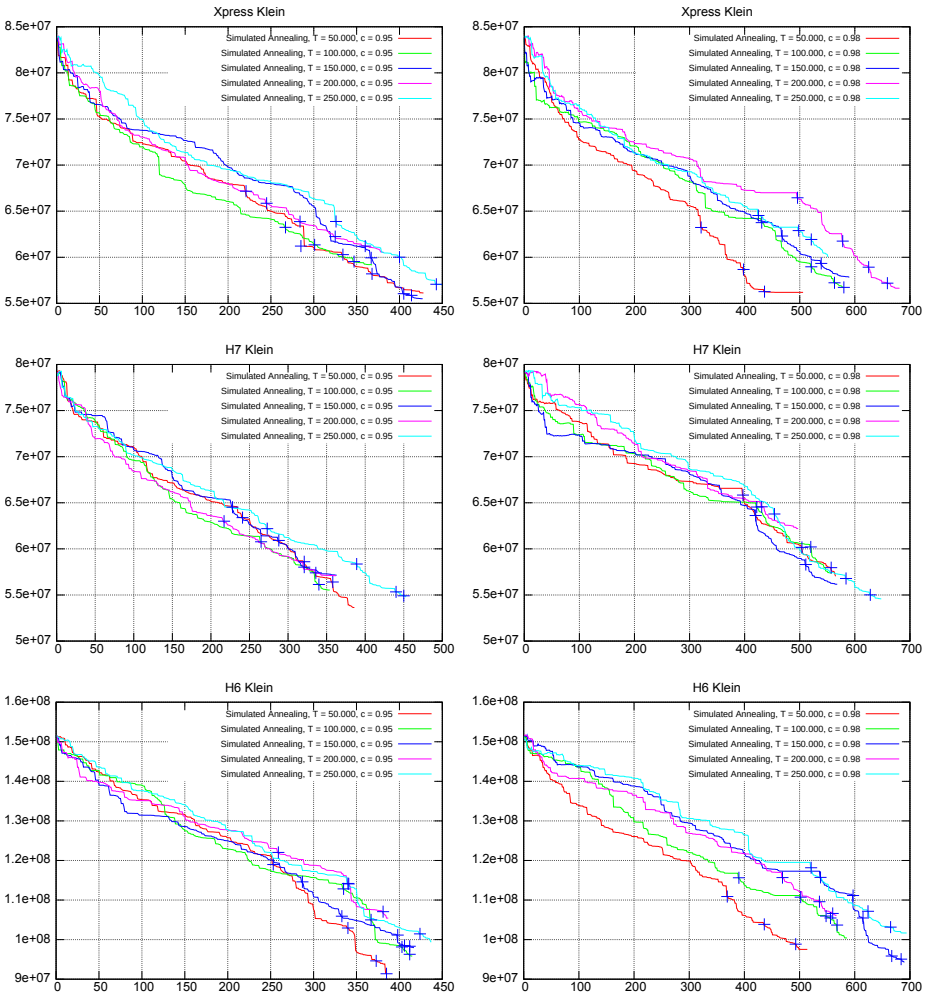
Abbildung 5.7: Ausschnitt aus Simulated Annealing.

Es stellt sich die Frage, ob ein geometrisches Abkühlungsschema die beste Wahl darstellt. Ein Single Node Cut wird erst gesucht, wenn die Temperatur bereits so stark abgekühlt ist, dass auch leicht verschlechternde Basiswechsel nicht mehr gewählt werden. Daher ist zu erwarten, dass nach dem ersten Single Node Cut die Möglichkeit, ein lokales Minimum durch eine glückliche Wahl von Fundamentalschnitten wieder zu verlassen, nicht mehr eintritt. Ein Schema, dass die Temperatur nach einem lokalen Schnitt wieder erhöht, trägt diesem Umstand Rechnung und könnte bessere Ergebnisse erzielen.



Instanz	Anfangslösung	Endlösung	Verbesserung
Xpress Groß, $T = 200.000$, $c = 0{,}98$	107.928.736	73.548.353	31,9 %
H7 Groß, $T = 250.000$, $c = 0{,}95$	111.583.840	70.101.044	37,2 %
H6 Groß, $T = 200.000$, $c = 0{,}95$	205.388.540	123.281.382	40,0 %
Xpress Klein, $T = 150.000$, $c = 0{,}95$	83.798.525	55.501.934	33,8 %
H7 Klein, $T = 50.000$, $c = 0{,}95$	79.119.703	53.637.589	32,2 %
H6 Klein, $T = 50.000$, $c = 0{,}95$	151.450.954	91.370.387	39,7 %

Tabelle 5.10: Auswahl an Daten zu Modus Simulated Annealing.



Instanz	Anzahl single node cuts	Anzahl Iterationen	Rechenzeit in s	Zeit in s pro Iteration
Xpress Groß, $T = 200.000$, $c = 0{,}98$	4	959	8.477	8,8
H7 Groß, $T = 250.000$, $c = 0{,}95$	5	934	10.975	11,8
H6 Groß, $T = 200.000$, $c = 0{,}95$	5	722	18.500	25,6
Xpress Klein, $T = 150.000$, $c = 0{,}95$	4	424	1.316	3,1
H7 Klein, $T = 50.000$, $c = 0{,}95$	3	384	1.286	3,3
H6 Klein, $T = 50.000$, $c = 0{,}95$	4	381	1455	3,8

Tabelle 5.11: Auswahl Daten zu Modus Simulated Annealing.

5.2.6 Modus Simulated Annealing / Steilster Abstieg Hybrid

Es lässt sich erkennen, dass die hohe Zahl an Iterationen im Simulated Annealing, die nötig sind, um in ein lokales Optimum zu geraten, dazu führt, dass das Verfahren größtenteils einem Modus „Fastest“ gleicht, bei dem auf ein Auswahlkriterium verzichtet wurde. Nur zu Beginn des Verfahrens sind spürbare Verschlechterungen des Zielfunktionswertes zu beobachten.
Dieser Umstand lässt sich dadurch umgehen, das nicht gleich von der relativ schlechten Anfangslösung aus ein zufälliger Schritt gewählt wird, sondern erst dann, wenn ein anderes Verfahren wie der steilste Abstieg in ein lokales Optimum gelangt ist. Gerade zu Beginn der bisher ausgewerteten Verfahren sind Basiswechsel mit hohen Verbesserungen des Zielfunktionswertes möglich, die von Simulated Annealing nicht gefunden werden.
Der Modus Simulated Annealing / Steilster Abstieg Hybrid (SD/SA) besteht somit darin, den originalen Modulo-Netzwerk-Simplex Algorithmus zu verwenden, bis er weder einen verbessernden Basiswechsel, noch einen verbessernden Single Node Cut finden kann, um von dieser Lösung aus ein Simulated Annealing-Verfahren zu beginnen. Diagramm 5.8 zeigt einen Ausschnitt des Verfahrens, bei dem gerade dieser Übergang stattfindet.

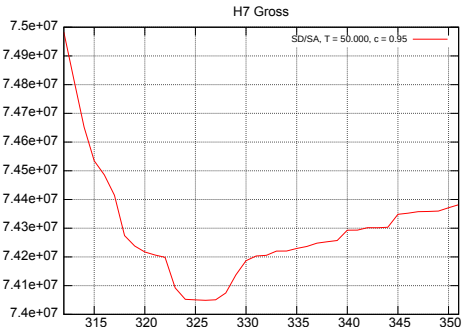
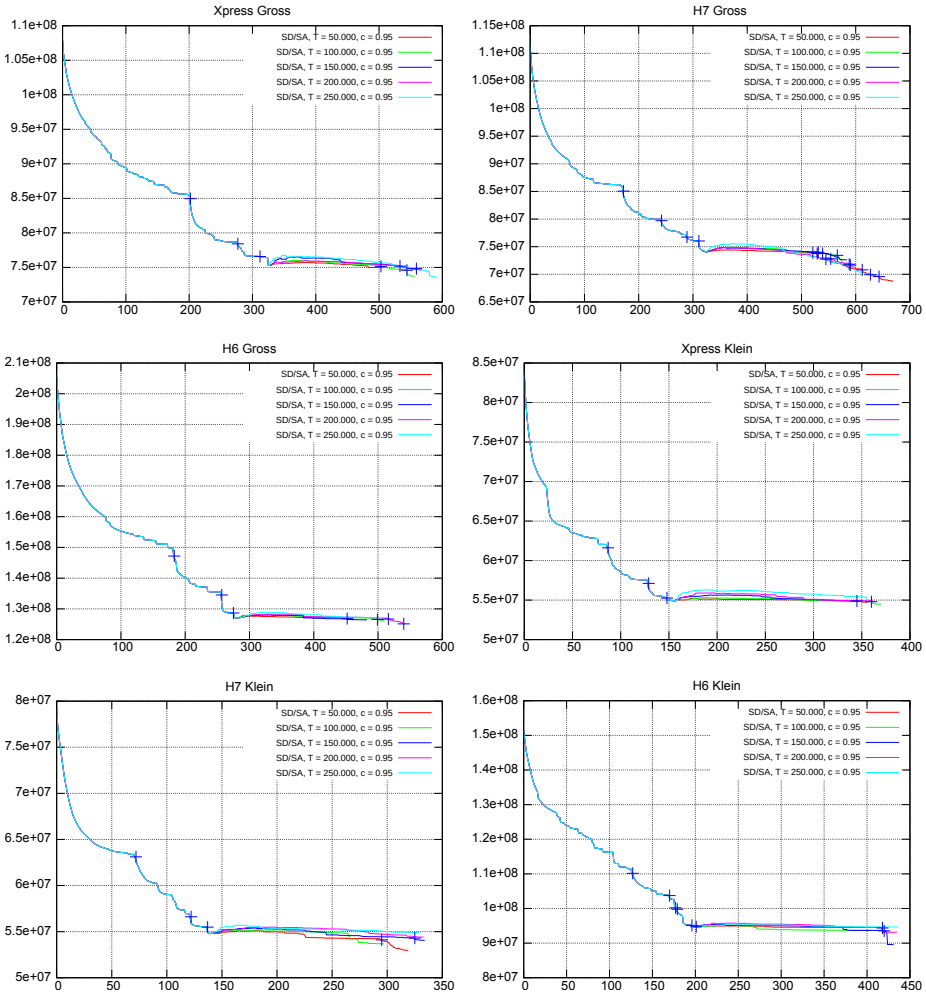


Abbildung 5.8: Ausschnitt aus SD/SA.

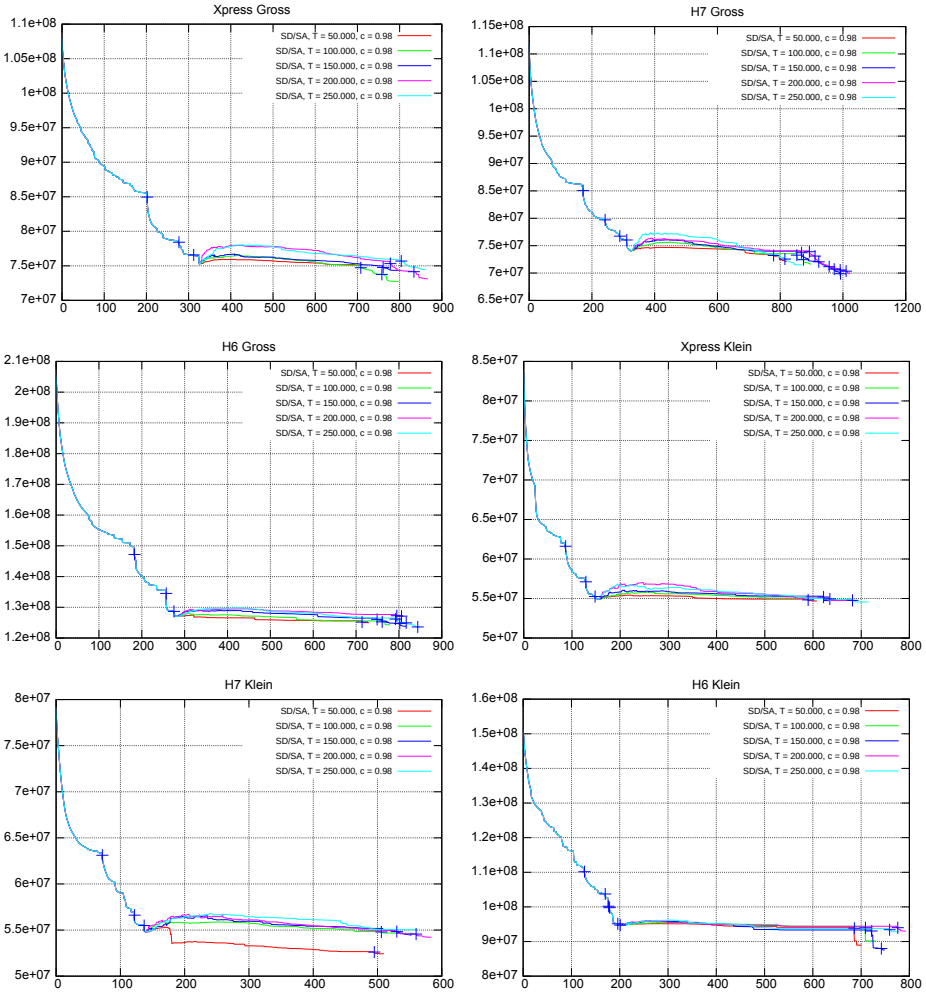
Die Durchführung dieses Verfahrens fand mit denselben Parametern statt, die auch für den gewöhnlichen Simulated Annealing-Modus gewählt wurden. Wie gehabt werden die jeweils besten Vertreter tabellarisch erfasst.

Es lässt sich erkennen, dass es für jede der getesteten Instanzen einen Vertreter dieses Verfahrens gibt, der die Lösung des steilsten Abstiegs weiter verbessern konnte. Im Falle von H6 Groß sogar von 33,6% auf 38,4% und in H6 Klein von 37,5% auf 42,2%, was jeweils Bestmarken aller verwendeten Verfahren darstellen. Diese hohe Qualität wird leider durch Laufzeiten bezahlt, die selbstverständlich über denen des steilsten Abstiegs und auch über denen von Simulated Annealing liegen.



Instanz	Anfangslösung	Endlösung	Verbesserung
Xpress Groß, $T = 100.000$, $c = 0{,}98$	107.928.736	72.721.237	32,6 %
H7 Groß, $T = 50.000$, $c = 0{,}95$	111.583.840	68.768.986	38,4 %
H6 Groß, $T = 250.000$, $c = 0{,}98$	205.388.540	123.496.328	39,9 %
Xpress Klein, $T = 100.000$, $c = 0{,}95$	83.798.525	54.479.510	35,0 %
H7 Klein, $T = 50.000$, $c = 0{,}98$	79.119.703	52.429.526	33,7 %
H6 Klein, $T = 150.000$, $c = 0{,}98$	151.450.954	87.559.448	42,2 %

Tabelle 5.12: Auswahl an Daten zu Modus SD/SA.



Instanz	Anzahl single node cuts	Anzahl Iterationen	Rechenzeit in s	Zeit in s pro Iteration
Xpress Groß $T = 100.000$, $c = 0{,}98$	5	793	19.208	24,2
H7 Groß, $T = 50.000$, $c = 0{,}95$	8	662	18.405	27,8
H6 Groß, $T = 250.000$, $c = 0{,}98$	6	842	23.428	27,8
Xpress Klein, $T = 100.000$, $c = 0{,}95$	4	367	8.684	23,7
H7 Klein, $T = 50.000$, $c = 0{,}98$	4	507	3.684	7,3
H6 Klein, $T = 150.000$, $c = 0{,}98$	8	750	3.764	5,0

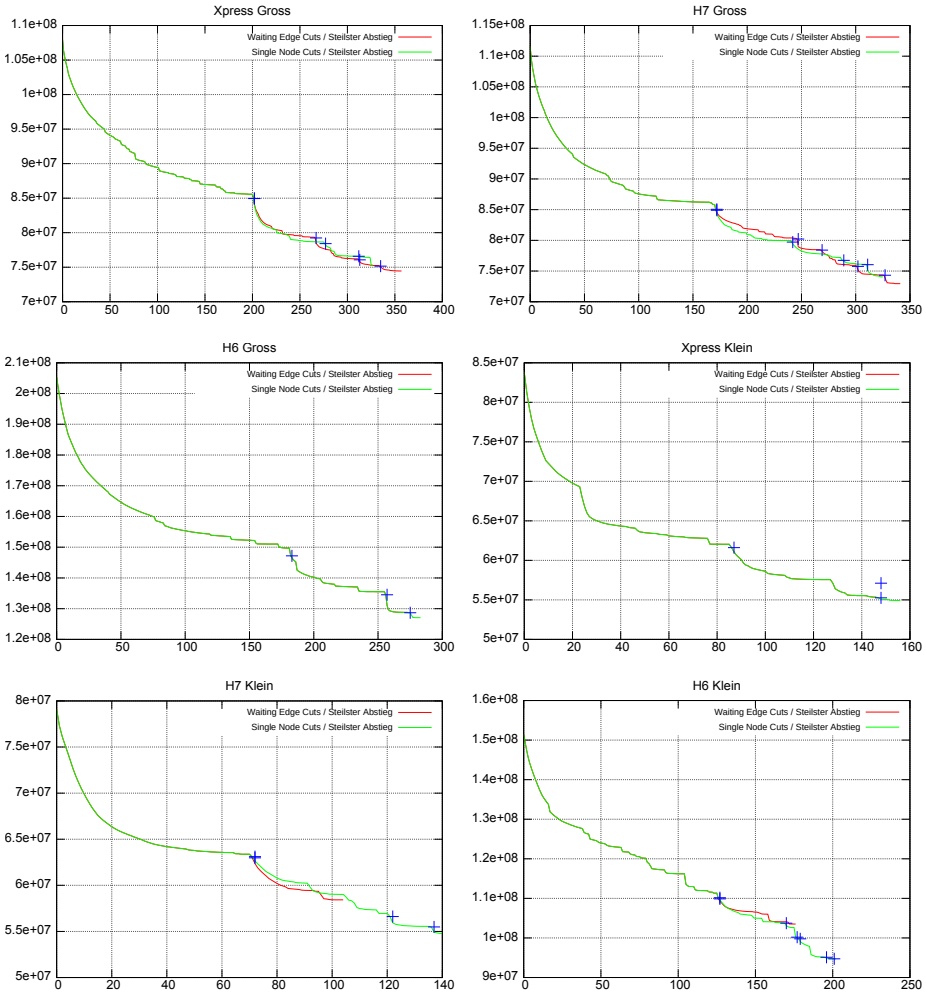
Tabelle 5.13: Auswahl Daten zu Modus SD/SA.

5.3 Lokalschnitt-Variationen

Die bisher präsentierten Verfahren haben alle gemein, dass sie nach *Single Node Cuts* als lokale Verbesserungen suchen. Ausgewählte Repräsentanten des vorherigen Kapitels werden nun mit zwei weiteren Möglichkeiten kombiniert, um aus einem gefundenen lokalen Optimum wieder zu entkommen.

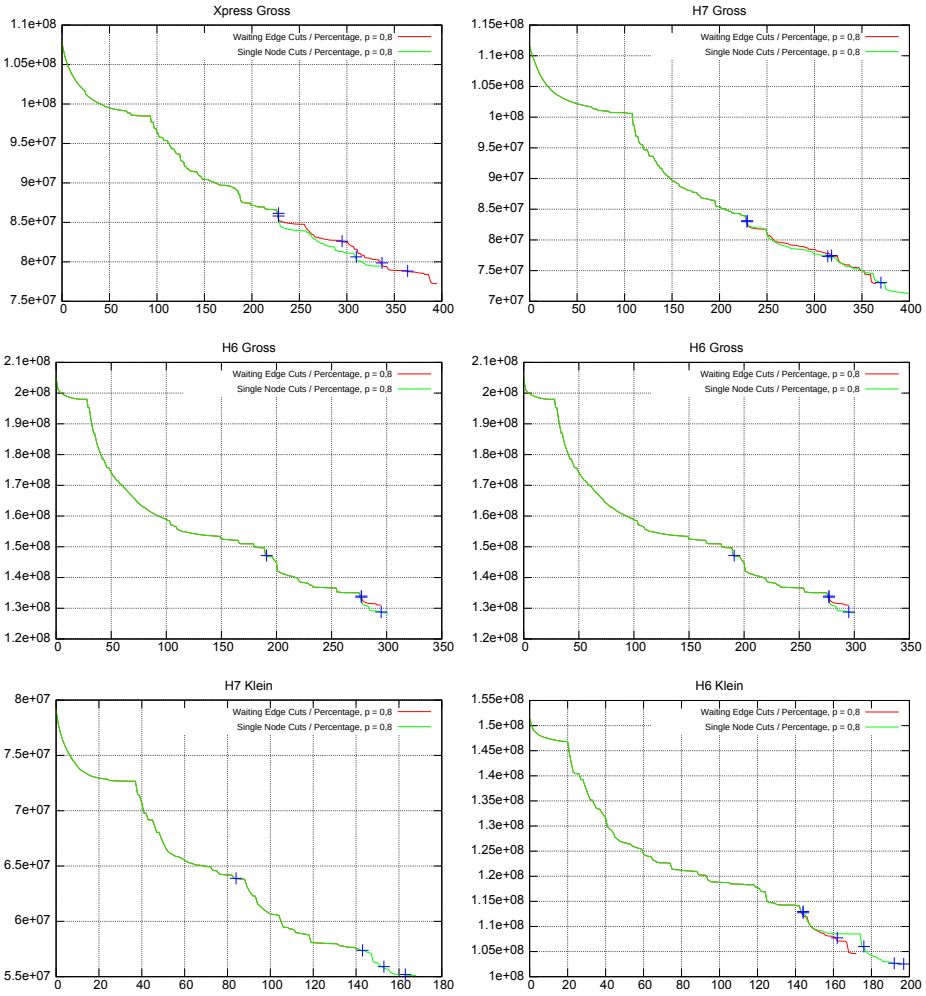
5.3.1 Waiting Edge Cuts

Wie in Abschnitt 3.3.5 erläutert, kann es lohnenswert sein, anstelle von *Single Node Cuts*, die aus allen ein- und ausgehenden Kanten eines einzelnen Knoten bestehen, einen *Waiting Edge Cut* zu verwenden, der die geringen Spielräume für Wartezeiten auslässt. Um die Vergleichbarkeit dieser lokalen Verbesserungsmethoden zu ermöglichen, kann auf kein zufälliges Verfahren zurückgegriffen werden, weshalb die folgenden Diagramme zunächst den steilsten Abstieg und anschließend den Modus Percentage mit $p = 0{,}8$ präsentieren.



In den Tabellen 5.14 und 5.15 werden die erzielten Resultate verglichen.

Es ist auffällig, dass bei Verwendung von Waiting Edge Cuts fast ausschließlich schlechtere Ergebnisse erzielt werden. Grund dafür ist möglicherweise die in den



Instanz	Endlösung Single Node Cut	Endlösung Waiting Edge Cut	Verbesserung der Lösung?
Xpress Groß,	75.301.572	74.448.960	✓
H7 Groß,	74.048.827	72.943.103	✓
H6 Groß,	127.071.960	128.789.748	✗
Xpress Klein,	54.890.924	55.348.472	✗
H7 Klein,	54.780.460	58.427.236	✗
H6 Klein,	94.666.655	103.521.865	✗

Tabelle 5.14: Vergleich Single Node Cut / Waiting Edge Cut für den steilsten Abstieg.

Instanz	Endlösung Single Node Cut	Endlösung Waiting Edge Cut	Verbesserung der Lösung?
Xpress Groß,	79.454.779	77.238.895	✓
H7 Groß,	71.299.287	72.937.005	✗
H6 Groß,	128.488.864	131.027.492	✗
Xpress Klein,	55.001.497	58.781.740	✗
H7 Klein,	55.098.459	57.532.458	✗
H6 Klein,	102.530.560	104.598.086	✗

Tabelle 5.15: Vergleich Single Node Cut / Waiting Edge Cut für den Modus Percentage.

verwendeten EANs grundsätzlich geringe zulässige Intervalllänge jeder Kante, die den Vorteil, dass auf Wartekanten verzichtet wird, abschwächt. Da es zudem weniger Wartekanten als Knoten gibt und die Menge der betrachteten Schnitte dadurch geringer ist, sind Waiting Edge Cuts in diesem Fall sogar weniger effektiv als Single Node Cuts.

5.3.2 Random Node Cuts

Den bisher präsentierten Resultaten ist anzusehen, dass der Zielfunktionswert nach einem lokalen Veränderungsschritt häufig stark weitersinken kann. Ferner ist überraschend oft ein Verfahren, das sich in jeder einzelnen Iteration schlecht verhält, im Resultat gut - der Suchraum ist so groß, dass die Qualität eines Auswahlverfahrens für Basiswechsel nicht voraussagbar ist. Die Kombination dieser beiden Beobachtungen liegt in einem lokalen Schnittverfahren, das gar nicht fordert, dass der gewählte Schnitt verbessernd ist, sondern sich nur auf die Zulässigkeit beschränkt. Um die Endlichkeit dieses Verfahrens zu garantieren, wird dafür die maximale Anzahl, wie oft insgesamt nach lokalen Schnitten gesucht werden darf, begrenzt. Für die hier präsentierten Experimente wurde diese Grenze auf 10 gesetzt.
Ein Single Node Cut, der nur zulässig und nicht zwangsweise verbessernd ist, verändert unter Umständen nicht die Modulo-Parameter einer Lösung. Da bei der anschließenden Berechnung der bezüglich dieser Parameter optimalen Lösung wieder dieselbe Spannender-Baum-Struktur entstünde, werden solche Schnitte ignoriert. Um die Beliebigkeit des gewählten Schnittes zudem zu erhöhen, werden alle Knoten in zufälliger Reihenfolge durchsucht.
Wie im Falle der Waiting Edge Cuts werden jeweils die Verfahren des steilsten Abstiegs und der Modus Percentage mit $p = 0{,}8$ miteinander verglichen.

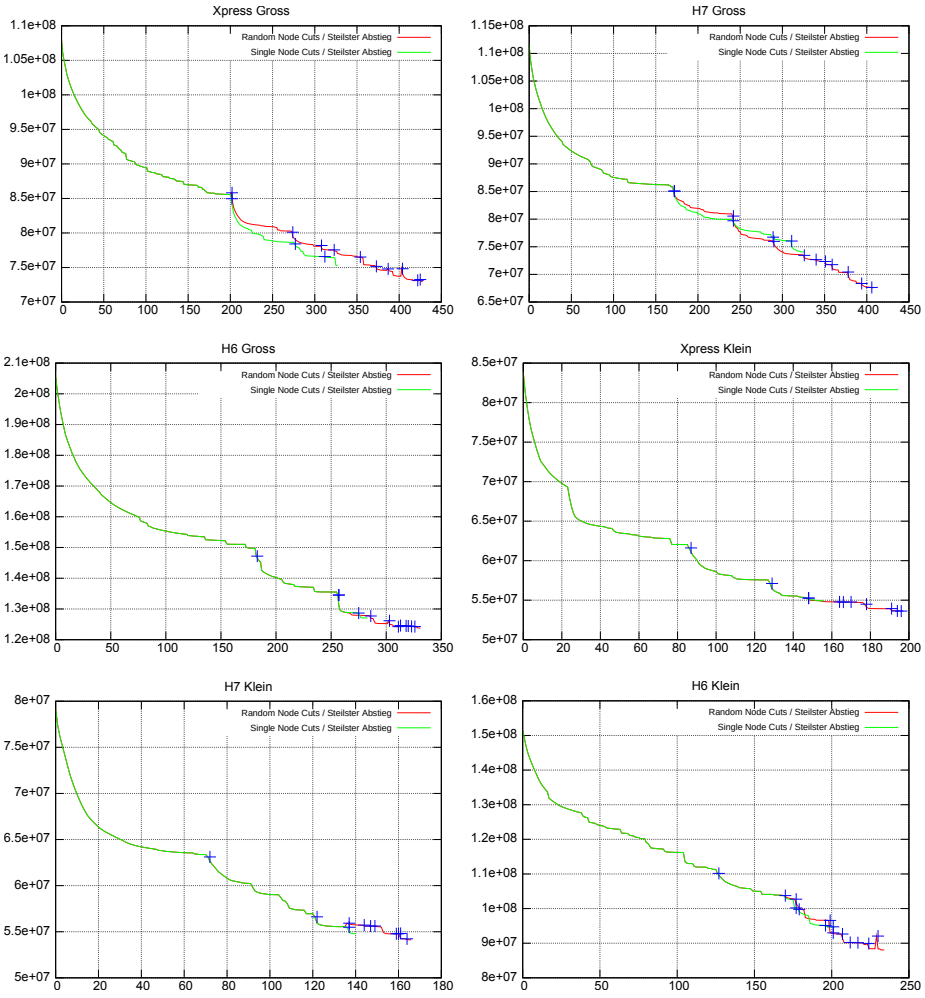
Instanz	Endlösung Single Node Cut	Endlösung Random Node Cut	Verbesserung der Lösung?
Xpress Groß,	75.301.572	72.944.473	✓
H7 Groß,	74.048.827	67.571.558	✓
H6 Groß,	127.071.960	123.777.825	✓
Xpress Klein,	54.890.924	53.617.186	✓
H7 Klein,	54.780.460	54.123.057	✓
H6 Klein,	94.666.655	88.056.446	✓

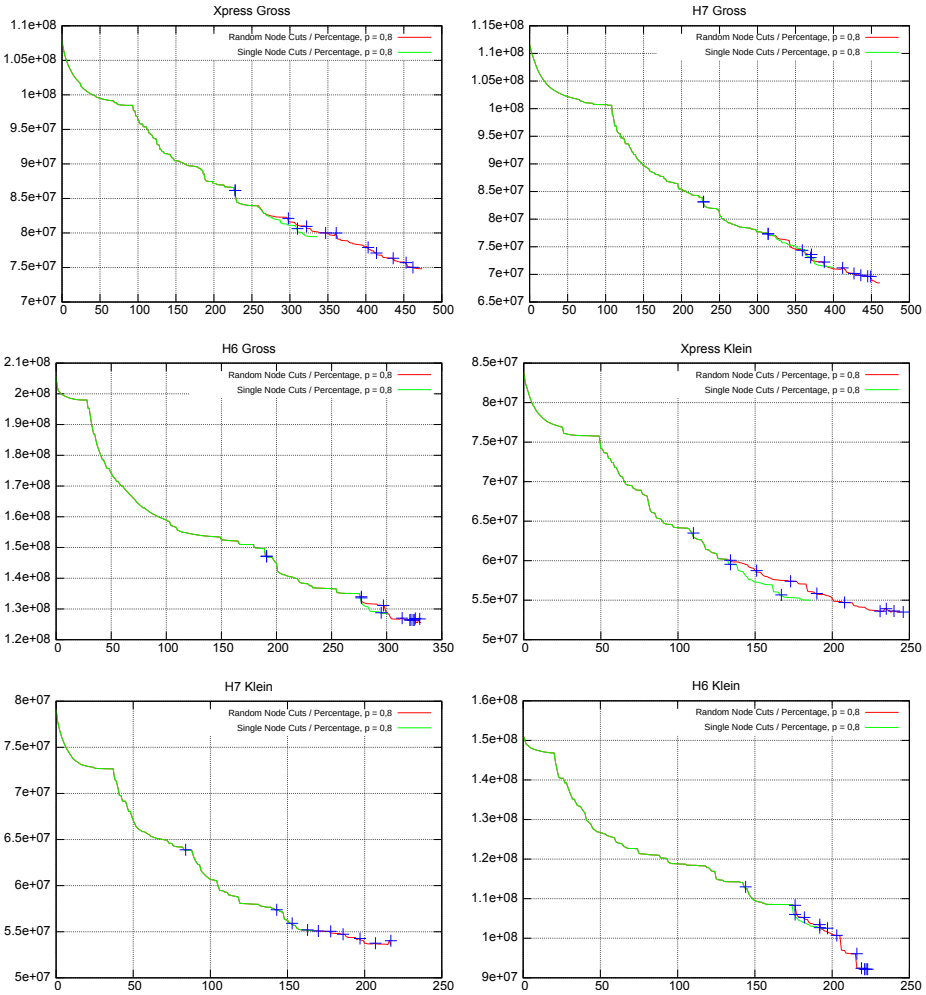
Tabelle 5.16: Vergleich Single Node Cut / Random Node Cut für den steilsten Abstieg.

Instanz	Endlösung Single Node Cut	Endlösung Random Node Cut	Verbesserung der Lösung?
Xpress Groß,	79.454.779	74.799.976	✓
H7 Groß,	71.299.287	68.462.898	✓
H6 Groß,	128.488.864	125.577.374	✓
Xpress Klein,	55.001.497	53.497.559	✓
H7 Klein,	55.098.459	53.609.757	✓
H6 Klein,	102.530.560	92.180.703	✓

Tabelle 5.17: Vergleich Single Node Cut / Random Node Cut für den Modus Percentage.

Es sind in jedem einzelnen Fall bessere Ergebnisse erzielt worden als bei der Verwendung von Single Node Cuts. Diese Tatsache zeigt, dass es im lokalen Ver-


Xpress Gross
H7 Gross
H6 Gross
Xpress Klein
H7 Klein
H6 Klein
Random Node Cuts / Steilster Abstieg
Single Node Cuts / Steilster Abstieg


Xpress Gross
Random Node Cuts / Percentage, p = 0,8
Single Node Cuts / Percentage, p = 0,8
H7 Gross
Random Node Cuts / Percentage, p = 0,8
Single Node Cuts / Percentage, p = 0,8
H6 Gross
Random Node Cuts / Percentage, p = 0,8
Single Node Cuts / Percentage, p = 0,8
Xpress Klein
Random Node Cuts / Percentage, p = 0,8
Single Node Cuts / Percentage, p = 0,8
H7 Klein
Random Node Cuts / Percentage, p = 0,8
Single Node Cuts / Percentage, p = 0,8
H6 Klein
Random Node Cuts / Percentage, p = 0,8
Single Node Cuts / Percentage, p = 0,8

änderungsschnitt wichtiger ist, die momentane Lösung überhaupt zu verschieben, als zwangsweise zu verbessern. Eine auftretende Verschlechterung kann durch die anschließenden Pivotisierungen immer ausgeglichen werden.
Nicht an den Diagrammen ablesen lässt sich, dass der Großteil der random node cuts zwar die Lösung verbesserten, vom Programm aber als die Lösung verschlechternd berechnet wurden. Das liegt daran, dass die lokale Veränderung des Zielfunktionswertes nur *einen* Vertreter aller Lösungen mit den neu gewonnen Modulo-Parametern darstellt und daher eigentlich eine *obere Schranke* ist.

Kapitel 6

Fazit

6.1 Rückblick

Die Vergleichbarkeit der präsentierten Verfahren wird einerseits durch eine große Menge möglicher Parameter, die niemals erschöpfend getestet werden können, und der Verwendung von Zufallsmechanismen erschwert. Doch sind die Differenzen bezüglich der Qualität der erzielten Lösungen und der Laufzeiten so erheblich, dass ein eindeutiges Urteil möglich ist. Die Tabellen 6.1 und 6.2 geben eine Übersicht der Ergebnisse bei Verwendung von Single Node Cuts. Fett geschrieben sind die besten Einträge einer Zeile.

Instanz	SD	F	P	TS	SA	SD/SA
Xpress Groß	30,2	28,4	30,8	30,4	31,9	**32,6**
H7 Groß	33,6	34,8	36,1	35,7	37,2	**38,4**
H6 Groß	38,1	38,7	37,4	38,1	**40,0**	39,9
Xpress Klein	34,5	31,3	34,4	34,5	33,8	**35,0**
H7 Klein	30,8	29,2	29,4	32,1	32,2	**33,7**
H6 Klein	37,5	35,7	32,3	37,5	39,7	**42,2**

Tabelle 6.1: Prozentuale Verbesserung der Anfangslösung der jeweils erfolgreichsten Repräsentanten bei Verwendung von Single Node Cuts.

Instanz	SD	F	P	TS	SA	SD/SA
Xpress Groß	14.918	4.626	**4.194**	57.872	8.477	19.208
H7 Groß	13.974	5.398	**4.679**	70.156	10.975	18.405
H6 Groß	9.770	10.910	**10.800**	37.777	18.500	23.428
Xpress Klein	7.920	**884**	1.214	14.915	1.316	8.684
H7 Klein	2.326	**791**	1.873	7.614	1.286	3.684
H6 Klein	1.657	**898**	1.562	7.439	1.455	3.764

Tabelle 6.2: Laufzeiten in s der jeweils erfolgreichsten Repräsentanten bei Verwendung von Single Node Cuts.

Die Modi Fastest und Percentage sind, was ganz ihrem ursprünglichen Zweck

entspricht, die schnellsten hier betrachteten Verfahren. Nun ist Geschwindigkeit allein ein irreführendes Kriterium zur Beurteilung eines Verfahrens, da es in diesem Fall am „besten“ wäre, ohne eine Iteration die Anfangslösung zu behalten. Doch auch die Qualität der erzielten Lösungen ist mit dem Referenzverfahren des steilsten Abstiegs vergleichbar, teilweise sogar besser.
Liegt der gewünschte Schwerpunkt weniger auf der benötigten Rechenzeit und mehr auf dem Zielfunktionswert, so sind die beiden Simulated Annealing-Verfahren die beste Wahl. Leider lässt sich nicht ermitteln, welche Parameter besonders geeignet wären, da diese einerseits vom EAN abhängen und andererseits vom Zufall überdeckt werden. Tabu Search ist durch die große Menge möglicher Basiswechsel von wenig Nutzen und kann unter sehr hohen Laufzeitkosten nur wenig bessere Ergebnisse erzielen als durch den steilsten Abstieg möglich sind.
Diagramm 6.1 stellt für das EAN „Xpress Groß“ den jeweils besten und schlechtesten Vertreter eines Verfahrens vergleichend dar. Die größte Differenz innerhalb eines Verfahrens liegt offenbar beim Simulated Annealing, wie auch zu erwarten ist, da hier der Einfluss des Zufalls am größten ist.

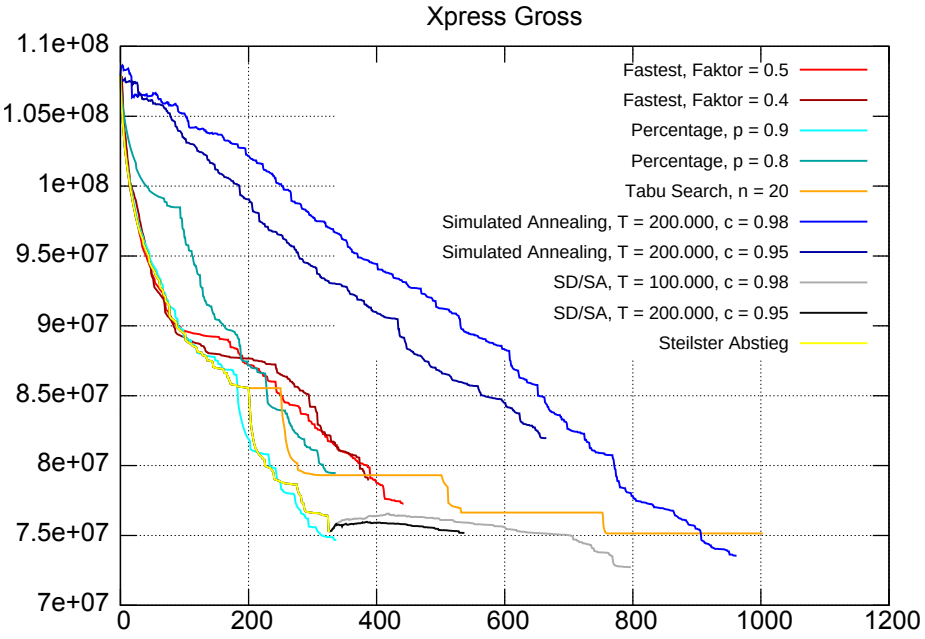


Abbildung 6.1: Vergleich der besten und schlechtesten Verfahrensvertreter.

Mit Hinblick auf die drei getesteten lokalen Schnittverfahren Single Node Cuts, Waiting Edge Cuts und random node cuts kann im Rahmen der durchgeführten Versuche festgestellt werden, dass die rein zufällige Veränderung der bestehenden Lösung durch random node cuts trotz vorübergehender Verschlechterung des Zielfunktionswertes, die durch die folgenden Pivotschritte leicht wieder gut gemacht werden kann, der Auswahl von verbessernden Single Node Cuts überlegen ist. Von letzteren werden zu wenig gefunden, so dass die Verfahren zu früh abbrechen. Die Verwendung von Waiting Edge Cuts ist in den meisten Fällen noch weniger gewinnbringend, da davon weniger zur Verfügung stehen.
Insgesamt schneiden gerade diejenigen Verfahren gut ab, die sich auf den Zufall stützen. Die Zielfunktion ist derart „unstetig“ auf der Menge der zulässigen

Spannender-Baum-Strukturen, dass eine besonders „gute“ Auswahl des Nachbarn in jeder Iteration kaum Einfluss auf die am Ende erzielte Lösung besitzt. Anstatt also Rechenzeit in den Vergleich der Zielfunktion aller Nachbarn einer Lösung zu legen, ist es sinnvoller, einen möglichst großen Bereich besucht zu haben. Aus mathematischer Sicht sind die beschriebenen Verfahren daher leider kaum vorhersagbar und auch der Intuition wird ein Streich gespielt. Es bleibt weiterhin eine offene und spannende Frage, wie der Lösungsraum der periodischen Fahrplangestaltung effektiv durchsucht werden kann.

6.2 Ausblick

Werkzeuge werden mit dem Ziel entwickelt, bei einer bestimmten Aufgabe behilflich zu sein. Kein Handwerker käme auf die Idee, ein neuartiges Gerät zu kaufen, das noch keinen sichtbaren Zweck erfüllt, mit der Hoffnung, es könnte sich irgendwann für ein Problem nützlich erweisen, das noch gar nicht existiert. Auch eine mathematische Theorie wird, sofern sie nicht dem Verständnis entspringt, Mathematik ihres Selbstzwecks wegen zu betreiben, mit dem Ziel entwickelt, ein Problem zu lösen - und sei dieses auch wieder mathematischer Natur. Die derart entwickelten Werkzeuge sind aber in so hohem Grade abstrakt, dass es gerade ihre Stärke ist, in einem anderen als den ursprünglichen Zusammenhang wiederverwendet zu werden. Daher kann die Qualität einer mathematischen Theorie nur begrenzt danach bewertet werden, inwiefern sie bei der Lösung des Zieles behilflich ist, für das sie entwickelt wurde.
Gerade in der angewandten Mathematik dagegen lässt sich der Erfolg einer mathematischen Lösung danach bemessen, wie gut die Ergebnisse sind, die sich auch praktisch umsetzen lassen. Im Falle der periodischen Fahrplangestaltung bedeutet das:

- Das Problem ist sehr gut modelliert und mathematisch erschlossen worden, denn es sind bereits Lösungen im realen Umfeld eingesetzt worden.
- Die dafür verwendete Mathematik ist hinreichend abstrakt, um auch in anderen Gebieten einsetzbar zu sein. Damit ist auch der „mathematische Werkzeugkoffer“ erweitert worden.
- Instanzen realistischer Größe sind, wie in dieser Arbeit gezeigt, bereits in einem Zeitrahmen lösbar, der weit unter den Planungslängen liegt, die für die Einführung eines neuen Fahrplans veranschlagt werden. Das Problem lässt sich inzwischen also an den Umständen gemessen effizient lösen.

Die jetzt folgende, noch offene Aufgabe aus mathematischer Sicht liegt also weniger darin, wie überhaupt eine gute Lösung in annehmbar kurzer Zeit gefunden werden kann, sondern eher, wie innerhalb eines vernünftigen Zeitrahmens die beste verfügbare Lösung ermittelt werden kann. Gerade im Bereich der Fahrplangestaltung ist es sinnvoll, einen Algorithmus einen Tag länger rechnen zu lassen, wenn dadurch über Jahre hinweg tagtäglich tausende Passagiere zufriedener und tausende Euros eingespart werden können.
Der praktisch gesehen weitaus größere Aufwand dürfte inzwischen aber in der Modellierung des Problems liegen. Vom Aufwand abgesehen, die umfassenden Daten eines realen Netzwerkes mitsamt unzähliger Headway- und Umlaufaktivitäten in die geschliffene Form eines Ereignis-Aktivität-Netzwerkes zu bringen, bemisst sich der Nutzen eines Fahrplans an mehr Faktoren als die gewichtete Fahrzeit. Tatsächlich ist das Ziel, Fahrzeiten zu minimieren, demjenigen sogar

entgegengesetzt, einen Fahrplan zu schaffen, der auf Verspätungen möglichst unempfindlich reagiert. Entsprechend wurde im bisher spektakulärsten Einsatz, die Einführung des niederländischen Fahrplans, sogar auf die hier beschriebene Zielfunktion gänzlich verzichtet. Robustheit mit in die aperiodische Fahrplangestaltung zu integrieren, ist ein offenes Feld und nicht mit den hier beschriebenen Algorithmen lösbar. Man kann sogar sagen: Das in dieser Arbeit verwendete Modell der Fahrplangestaltung ist gerade deshalb inzwischen gut mathematisch lösbar, weil es auch entsprechend gestaltet wurde. Der weitere Weg wird über kurz oder lang zu anderen Modellen führen müssen.
Das Problem erinnert an Kants „empirischen Maschinisten“ aus dem Zitat, das zu Beginn des experimentellen Teils widergegeben wurde. Glücklicherweise ist das verwendete Modell hinreichend gut entwickelt, als dass der Mathematiker fürchten müsste, „belacht“ zu werden. Doch um in der Praxis noch wirksamer zu sein, ist jetzt eines nötig: Zusätzliche Theorie.

Anhang A

Genetische Verfahren

Für diese Diplomarbeit wurde auch ein genetisches Verfahren implementiert. Es wird die Funktionsweise hier dargestellt und erläutert, warum es nicht über die Lösung der „Spiel"-EANs hinaus gelangte und daher nicht Teil der Auswertung wurde.

Mitte des 20. Jahrhunderts trat Computerwissenschaftlern zunehmend die Evolution als Vorbild für Lösungsstrategien in den Blickpunkt. Während diese neu gewonnene Perspektive nicht nur die Phantasie der Science-Fiction Autoren beflügelte (sogar Manfred Eigen fragte: „Wird der Mensch in einer unbegrenzten technischen Evolution das Steuer in der Hand behalten können? Oder wird er einmal zur mehr oder weniger bedeutungslosen Zelle eines gigantischen sich selbst fortpflanzenden und ständig optimierenden Automaten absinken?" (Rec73)), konnten unter anderem mit dem John Holland zugeschriebenen genetischen Algorithmus völlig neuartige Lösungen berechnet werden.
In Anlehnung an die bekannten Begriffe aus der Biologie wird eine zulässige Lösung ein *Phänotyp* und die sie erzeugende Codierung der *Genotyp* genannt. Eine Menge an Genotypen bildet eine Population. Der genetische Algorithmus verwendet iterativ eine bestehende Population, um die folgende Generation durch Kreuzung, Mutation und Selektion unter Verwendung einer Fitness-Funktion zu erzeugen. Da die genaue Art und Weise, wie dies geschehen soll, variabel ist, wird das Verfahren zu den Meta-Heuristiken gezählt.
Im Rahmen dieser Diplomarbeit wurde auch ein genetisches Verfahren zur Lösung des periodischen Fahrplanproblems implementiert. Dabei wurde auf die als Freeware verfügbare LibGA (CW93) zurückgegriffen, die eine breite Auswahl an Kreuzungs- und Mutationsverfahren für verschiedenste Genotypen bereitstellt.
Die grundlegende Idee, eine zulässige Lösung zu codieren, gibt die Beobachtung, dass für fixierte Modulo-Parameter das Problem mithilfe des klassischen Netzwerk-Simplex Algorithmus effizient lösbar ist. Daher kann der Suchraum als die Menge aller Modulo-Parameter aufgefasst werden und ein Phänotyp wird durch seine Modulo-Parameter codiert. Da diese sowohl von genetischen Verfahren besonders leicht handhabbar sind, als auch von Solvern besonders effektiv berechenbar sind, wurde der Suchraum zunächst auf *binäre* Modulo-Parameter beschränkt.
Problematisch erwies sich allerdings die Konstruktion einer Anfangsgeneration. Für EANs, die über die Größe eines Spielbeispiels hinaus gingen, konnte eine zufällige Aussaat in keinem einzigen der durchgeführten Experimente eine zulässige Lösung erzeugen. Auch das im Abschnitt 4.3.2 beschriebene Verfahren, Umstei-

gekanten zunächst auszulassen und das verkleinerte Problem vom Solver lösen zu lassen, ist nicht einsetzbar, da auf diese Weise beliebig große Modulo-Parameter entstehen können. Daher konnten die eigentlichen genetischen Algorithmen leider nicht getestet werden.
Allerdings zeigen die in dieser Arbeit ausgewerteten Versuche, dass ein derart gestaltetes Verfahren wahrscheinlich gar nicht effektiv wäre. Genetische Algorithmen stützen sich wesentlich darauf, dass Nachkommen besonders „guter“ Phänotypen mit hoher Wahrscheinlichkeit wieder „gute“ Lösungen ergeben. Diese Annahme ist zumindest im Falle der Codierung durch Spannender-Baum-Strukturen nicht zulässig, da sich schon durch die „kleinen“ Änderungen einer Pivotoperation die Qualität einer Lösung völlig verändern kann. Eine „Kreuzung“ zweier spannender Bäume ist im Resultat nicht abschätzbar, da die Zielfunktion vor allem von denjenigen Kanten beeinflusst wird, die sich *nicht* im Baum befinden.
Insgesamt konnte also das „Spiel-EAN“ unter Verwendung eines genetischen Algorithmus gelöst werden, die deutlich größeren, realistischen Netzwerke konnten auf diese Weise nicht angegangen werden.

Anhang B

Code

Im Verlauf der Arbeit wurden bereits einige Teile des verwendeten Codes in teilweise gekürzter Fassung vorgestellt. Die folgenden Seiten versuchen, eine detailliertere Übersicht zu geben, doch stellen sie aufgrund des großen Umfangs nach wie vor eine Auswahl dar.

```
/*
 * main.cpp
 *
 *  Created on: 29.03.2009
 *  Author: Marc Goerigk
 */
#include "modulo_network_simplex.h"

using namespace std;
using namespace modulosimplex;

int main(int argc, char** argv)
{

        simplex solver;

        solver.init(argv[1], argv[2], 60);

        switch(atoi(argv[3]))
        {
                case 1: //single node cuts
                        solver.set_loc_improvement(SINGLE_NODE_CUT);
                        break;
                case 2: //multi node cuts
                        solver.set_loc_improvement(RANDOM_CUT);
                        break;
                case 3: //waiting edge cuts
                        solver.set_loc_improvement(WAITING_CUT);
                        break;
                default: exit(0); break;
        }

        switch(atoi(argv[4]))
        {
                case 1: //steepest decent
                        solver.set_tab_search(TAB_FULL);
                        break;
                case 2: //tabu search
                        solver.set_tab_search(TAB_SIMPLE_TABU_SEARCH);
                        solver.set_ts_memory(atoi(argv[5]));
                        solver.set_ts_max_iterations(atoi(argv[6]));
                        break;
                case 3: //simulated annealing
                        solver.set_tab_search(TAB_SIMULATED_ANNEALING);
                        solver.set_sa_init_temperature(atoi(argv[5]));
                        solver.set_sa_coolness_factor(atof(argv[6]));
                        break;
                case 4: //simulated annealing steepest descent hybrid
                        solver.set_tab_search(TAB_STEEPEST_SA_HYBRID);
                        solver.set_sa_init_temperature(atoi(argv[5]));
                        solver.set_sa_coolness_factor(atof(argv[6]));
                        break;
                case 5: //percentage
                        solver.set_tab_search(TAB_PERCENTAGE);
                        solver.set_percentage_improvement(atoi(argv[5]));
                        break;
                case 6: //fastest
                        solver.set_tab_search(TAB_FASTEST);
                        solver.set_min_pivot_improvement(atof(argv[5]));
                        solver.set_dynamic_pivot_factor(atof(argv[6]));
                        solver.set_tab_min_improvement(DYNAMIC);
                        break;
```



```
                default: exit(0); break;
        }

        solver.solve();
        solver.write_result("timetable.tim");

        return 0;

}
```

Aug 01, 09 12:58 **modulo_network_simplex.h** Seite 1/4

```
/*
 * modulo_network_simplex.h
 *
 *  Created on: 29.03.2009
 *  Author: Marc Goerigk
 */

#ifndef MODULO_NETWORK_SIMPLEX_H_
#define MODULO_NETWORK_SIMPLEX_H_

#include <vector>
#include <queue>
#include <map>

#include "boost/utility.hpp"
#include "boost/config.hpp"
#include "boost/property_map.hpp"
#include "boost/graph/adjacency_list.hpp"

#include "boost/bimap.hpp"

enum edge_mintime_t { edge_mintime };
enum edge_maxtime_t { edge_maxtime };

namespace boost
{
BOOST_INSTALL_PROPERTY(edge, mintime);
BOOST_INSTALL_PROPERTY(edge, maxtime);
}

typedef boost::property<edge_maxtime_t, int> Maxtime;
typedef boost::property<edge_mintime_t, int, Maxtime> Mintime;
typedef boost::property<boost::edge_weight_t, int, Mintime> Edge_props;
typedef boost::adjacency_list<boost::vecS, boost::vecS, boost::bidirectionalS,bo
ost::no_property,Edge_props> Graph;
typedef boost::adjacency_list<boost::vecS, boost::vecS, boost::undirectedS,boost
::no_property,boost::no_property> Tree;
typedef boost::graph_traits<Graph>::edge_descriptor Edge;
typedef boost::graph_traits<Graph>::vertex_descriptor Vertex;
typedef boost::graph_traits<Graph>::vertex_iterator Vertex_Iterator;

namespace modulosimplex
{

        //These are the possible search methods for the fundamental cuts.
        enum TABLEAU_SEARCH
        {
                //Takes the best of all neighbours.
                TAB_FULL = 1,

                //Sorts the columns so that the shortest will be used first
                //in the search. Takes the first fundamental cut that is
                //good enough according to the min_pivot_improvement percentage.
                TAB_FASTEST = 2,

                //Sorts the columns so that the shortest will be used first
                //in the search. Takes the best fundumental cut out of the
                //first percentage_improvement percentage.
                TAB_PERCENTAGE = 3,

                //Uses simulated annealing to find the next pivot step. Paramete
rs
```


Aug 01, 09 12:58 **modulo_network_simplex.h** Seite 2/4

```
                //are sa_temperature and sa_cooling_factor.
                TAB_SIMULATED_ANNEALING = 4,

                //Tabu search.
                TAB_SIMPLE_TABU_SEARCH = 5,

                //Uses steepest descend until it arrives at the local optimum,
                //then switches to simulated annealing.
                TAB_STEEPEST_SA_HYBRID = 6
        };

        //These are the possible ways the minimum improvement of a pivot step ch
anges when none is found.
        enum TABLEAU_MIN_IMPROVEMENT
        {
                //The criterion stays fixed.
                FIXED = 1,

                //The criterion becomes smaller.
                DYNAMIC = 2
        };

        //These are the available local seach algorithms.
        enum LOCAL_IMPROVEMENT
        {
                //Single node cuts.
                SINGLE_NODE_CUT = 1,

                //Random node cuts.
                RANDOM_CUT = 2,

                //Waiting edge cuts.
                WAITING_CUT = 3
        };


class simplex
{
public:
        //Constructor and destructor.
        simplex();
        ~simplex();

        //Setter-methods.
        void set_tab_search(TABLEAU_SEARCH tab_search);
        void set_min_pivot_improvement(double min_percentage);
        void set_min_cut_improvement(double min_percentage);
        void set_tab_min_improvement(TABLEAU_MIN_IMPROVEMENT tab_min);
        void set_percentage_improvement(double percentage);
        void set_sa_init_temperature(double temp);
        void set_sa_coolness_factor(double fac);
        void set_ts_memory(int length);
        void set_ts_max_iterations(int number);
        void set_loc_improvement(LOCAL_IMPROVEMENT loc);
        void set_loc_number_of_nodes(int number);
        void set_loc_number_of_tries(int number);
        void set_dynamic_pivot_factor(double factor);

        //Initialises the algorithm.
        void init(std::string activities_file, std::string events_file, int give
n_period);
```

```
        void improvable();
        void transform();
        void non_periodic();
        bool pivot();

        //Starts the algorithm.
        void solve();

        //Writes the found timetable to the given file.
        void write_result(std::string filename);

private:

        struct ts_memory
        {
                std::set<unsigned short> lower_tree;
                std::set<unsigned short> upper_tree;

                uint lower_checksum;
                uint upper_checksum;
        };

        struct eta
        {
                 int where;
                 int delta;
                 bool active;
                 std::vector<int> directions;
        };

        void set_time();
        void set_time(Vertex where);
        int get_objective();
        int get_obj_change(int in_edge, int out_edge, bool as_lower);
        void set_coeff(int out_edge, int in_edge, int value);
        int get_coeff(int out_edge, int in_edge);
        void set_coeff_temp(int out_edge, int in_edge, int value);
        void update_coeffs();

        void swap_edges(int into_edge, int out_edge, bool as_lower);
        void build();

        void find_modulo();
        void find_feasible_periodic();

        std::vector<int> m_rhs;
        std::vector<int> m_slack;
        std::vector<short> modulo_param;
        std::map<std::pair<unsigned short,unsigned short>,bool> coeff;
        std::map<std::pair<unsigned short,unsigned short>,bool> coeff_temp;
        std::vector<std::set<unsigned short> > columns;
        Graph* g;
        Tree* spanning_tree;
        std::vector<Edge> span;
        std::vector<unsigned short> in_tree_edges;
        std::vector<unsigned short> out_tree_edges;
        boost::bimap<unsigned short,Edge> graph_edges;
        boost::bimap<unsigned short,Vertex> graph_vertices;
        std::map<unsigned short,int> m_pi;
        std::vector<bool> lower_tree_edges;
        std::vector<std::string> edge_types;
```



```
        int nr_of_edges;
        int nr_of_vertices;
        int period;
        eta cut;

        static const bool verbose = false;
        static const bool presentation = false;
        static const bool countpercentages = false;

        std::vector<uint> distribution;

        TABLEAU_SEARCH search;
        TABLEAU_MIN_IMPROVEMENT search_impro;
        LOCAL_IMPROVEMENT local_search;
        //min_pivot_improvement = 0 means no aborting criteria
        double min_pivot_improvement;
        double min_cut_improvement;
        double percentage_improvement;
        double dynamic_pivot_factor;

        int loc_number_of_nodes;
        int loc_number_of_tries;
        int loc_current_tries;

        double sa_temperature;
        double sa_cooling_factor;

        int ts_memory_length;
        int ts_max_iterations;
        ts_memory best_solution;
        int best_objective;
        std::deque< ts_memory > ts_memory_deque;
        void ts_recreate_best();

        bool improved_last_time;

        bool sa_hybrid_active;
};
}//namespace

#endif /* SIMPLEX_H_ */
```

```
/*
 * modulo_network_simplex.cpp
 *
 *  Created on: 29.03.2009
 *  Author: Marc Goerigk
 */

using namespace boost;
using namespace modulosimplex;

static goblinController *CT_NW_Simplex;

void simplex::set_time()
{
        m_pi.clear();

        Vertex start =  source(span.front(),*g);
        m_pi[graph_vertices.right.at(start)]=0;
        set_time(start);
}

void simplex::set_time(Vertex where)
{
        property_map<Graph,edge_mintime_t>::type min = get(edge_mintime, *g);
        property_map<Graph,edge_maxtime_t>::type max = get(edge_maxtime, *g);


        for (uint i=0; i<in_tree_edges.size(); ++i)
        {
                Edge e = graph_edges.left.at(in_tree_edges[i]);
                if (source(e,*g) == where)
                {
                        if (m_pi.find(graph_vertices.right.at(target(e, *g))) ==
 m_pi.end())
                        {
                                if (lower_tree_edges[i])
                                {
                                        m_pi[graph_vertices.right.at(target(e,*g
))] = m_pi[graph_vertices.right.at(where)] + min[e];
                                }
                                else
                                {
                                        m_pi[graph_vertices.right.at(target(e,*g
))] = m_pi[graph_vertices.right.at(where)] + max[e];
                                }
                                set_time(target(e,*g));
                        }
                }

                if (target(e,*g) == where)
                {
                        if (m_pi.find(graph_vertices.right.at(source(e, *g))) ==
 m_pi.end())
                        {
                                if (lower_tree_edges[i])
                                {
                                        m_pi[graph_vertices.right.at(source(e,*g
))] = m_pi[graph_vertices.right.at(where)] - min[e];
                                }
                                else
                                {
                                        m_pi[graph_vertices.right.at(source(e,*g
```



```
))] = m_pi[graph_vertices.right.at(where)] - max[e];
                                }
                                set_time(source(e,*g));
                        }
                }
        }
}

void simplex::improvable()
{

        graph_traits<Graph>::out_edge_iterator out_i, out_end;
        graph_traits<Graph>::in_edge_iterator in_i, in_end;

        property_map<Graph,edge_maxtime_t>::type max = get(edge_maxtime, *g);
        property_map<Graph,edge_mintime_t>::type min = get(edge_mintime, *g);
        property_map<Graph,vertex_index_t>::type index = get(vertex_index, *g);
        property_map<Graph,edge_weight_t>::type w = get(edge_weight, *g);

        bool found = false;
        cut.active = false;
        cut.directions.reserve(num_edges(*g));

        if (local_search == SINGLE_NODE_CUT)
        {
                ...
        }
        else if (local_search == RANDOM_CUT)
        {
                ...
        }
        else if (local_search == WAITING_CUT)
        {
                ...
        }

        if (verbose && !(cut.active))
                std::cout<<"No cut found.\n"<<std::flush;
}

bool simplex::pivot()
{
        int max_change = 0;
        bool impro = false;
        int current_objective = get_objective();

        int change_in_edge=0;
        int change_out_edge=0;
        bool as_lower = true;

        uint loop_percentage=0;
        int loop_counter=0;

        if (search == TAB_SIMPLE_TABU_SEARCH && (best_objective == 0 || current_
objective < best_objective) )
        {
                ts_memory q_member;
                q_member.lower_checksum = 0;
                q_member.upper_checksum = 0;

                for (uint i=0; i<in_tree_edges.size(); ++i)
                {
```

```
                        if (lower_tree_edges[i])
                        {
                                q_member.lower_tree.insert(in_tree_edges[i]);
                                q_member.lower_checksum+= in_tree_edges[i];
                        }
                        else
                        {
                                q_member.upper_tree.insert(in_tree_edges[i]);
                                q_member.upper_checksum+=in_tree_edges[i];
                        }
                }

                if (best_objective != 0)
                {
                        ts_memory_deque.push_back(q_member);

                        if (ts_memory_deque.size() > (uint) ts_memory_length)
                                ts_memory_deque.pop_front();
                }

                best_objective = current_objective;
                best_solution = q_member;
        }

        if (search != TAB_SIMULATED_ANNEALING && verbose)
                std::cout<<"Searched 0%"<<std::flush;

        if (search == TAB_FULL || (search == TAB_STEEPEST_SA_HYBRID && sa_hybrid
_active == false))
        {
                for (std::map<std::pair<unsigned short, unsigned short>,bool>::i
terator it = coeff.begin(); it!=coeff.end(); ++it)
                        for (int temp_lower_int = 0; temp_lower_int<=1; temp_low
er_int++)
                        {
                                ...
                        }
        }
        else if (search == TAB_FASTEST)
        {
                std::vector<std::pair<int,unsigned short> > column_sizes(in_tree
_edges.size());
                std::map<unsigned short, std::set<unsigned short> > columns;

                for (std::map<std::pair<unsigned short, unsigned short>,bool>::i
terator it = coeff.begin(); it!=coeff.end(); ++it)
                {
                        columns[((*it).first).second].insert(((*it).first).first
);
                }

                for (uint i=0; i<in_tree_edges.size(); ++i)
                        column_sizes[i] = std::make_pair(columns[in_tree_edges[i
]].size(),in_tree_edges[i]);

                std::sort(column_sizes.begin(), column_sizes.end());
                int tempchange = 0;

                uint i=0;
                while(current_objective*min_pivot_improvement > -max_change && i
<column_sizes.size())
                {
```



```
                        ...
                }
        }
        else if (search == TAB_PERCENTAGE)
        {
                std::vector<std::pair<int,unsigned short> > column_sizes(in_tree
_edges.size());
                std::map<unsigned short, std::set<unsigned short> > columns;

                for (std::map<std::pair<unsigned short, unsigned short>,bool>::i
terator it = coeff.begin(); it!=coeff.end(); ++it)
                {
                        columns[((*it).first).second].insert(((*it).first).first
);
                }

                for (uint i=0; i<in_tree_edges.size(); ++i)
                        column_sizes[i] = std::make_pair(columns[in_tree_edges[i
]].size(),in_tree_edges[i]);

                std::sort(column_sizes.begin(), column_sizes.end());
                int tempchange = 0;

                uint i=0;
                while(loop_percentage < percentage_improvement)
                {
                        ...
                }

                if (!impro)
                {
                        while(i < in_tree_edges.size())
                        {
                                ...
                        }
                }

        }
        else if (search == TAB_SIMULATED_ANNEALING || (search == TAB_STEEPEST_SA
_HYBRID && sa_hybrid_active == true))
        {
                if (countpercentages)
                {
                        std::map<unsigned short, std::set<unsigned short> > colu
mns;

                        for (std::map<std::pair<unsigned short, unsigned short>,
bool>::iterator it = coeff.begin(); it!=coeff.end(); ++it)
                        {
                                columns[((*it).first).second].insert(((*it).firs
t).first);
                        }

                        for (uint i=0; i<in_tree_edges.size(); ++i)
                                ++distribution[columns[in_tree_edges[i]].size()]
;
                }



                if (verbose) std::cout<<"\nSearching.\n"<<std::flush;
                typedef std::pair< std::pair<unsigned short, unsigned short>, bo
```

```
ol> tab_value;
                std::vector< tab_value > possibilities;
                possibilities.reserve(coeff.size());

                for (std::map<std::pair<unsigned short, unsigned short>,bool>::i
terator it = coeff.begin(); it!=coeff.end(); ++it)
                {
                        possibilities.push_back(std::make_pair(std::make_pair(((
*it).first).first, ((*it).first).second),true));
                        possibilities.push_back(std::make_pair(std::make_pair(((
*it).first).first, ((*it).first).second),false));
                }

                int left_size = possibilities.size();

                while (!impro && left_size != 0)
                {
                        int position = rand()%left_size;
                        tab_value it = possibilities[position];
                        int tempchange = get_obj_change((it.first).first, (it.fi
rst).second, it.second);
                        if (tempchange == coeff_not_feasible)
                        {
                                tab_value helper = it;
                                possibilities[position] = possibilities[left_siz
e-1];
                                possibilities[left_size-1] = helper;
                                --left_size;
                        }
                        else if (tempchange < 0)
                        {
                                max_change = tempchange;
                                change_in_edge = (it.first).first;
                                change_out_edge = (it.first).second;
                                impro = true;
                                as_lower = it.second;
                                sa_temperature*=sa_cooling_factor;
                                if (verbose) std::cout<<"Current temperature: "<<sa_te
mperature<<"\n"<<std::flush;
                        }
                        else
                        {
                                double p;
                                tempchange == 0 ? p = 0 : p = exp(-tempchange/sa
_temperature);
                                double rval = double(rand())/RAND_MAX;
                                if (rval < p)
                                {
                                        ...
                                }
                                else
                                {
                                        ...
                                }
                        }
                }

        }
        else if (search == TAB_SIMPLE_TABU_SEARCH)
        {
                max_change = std::numeric_limits<int>::max();
                std::set<unsigned short> upper_tree_set;
```



```
                std::set<unsigned short> lower_tree_set;

                uint upper_tree_checksum = 0;
                uint lower_tree_checksum = 0;

                for (uint i=0; i<in_tree_edges.size(); ++i)
                {
                        if (lower_tree_edges[i])
                        {
                                lower_tree_set.insert(in_tree_edges[i]);
                                lower_tree_checksum+=in_tree_edges[i];
                        }
                        else
                        {
                                upper_tree_set.insert(in_tree_edges[i]);
                                upper_tree_checksum+=in_tree_edges[i];
                        }
                }

                for (std::map<std::pair<unsigned short, unsigned short>,bool>::i
terator it = coeff.begin(); it!=coeff.end(); ++it)
                        for (int temp_lower_int = 0; temp_lower_int<=1; temp_low
er_int++)
                        {
                                ...
                        }
        }

        if (search != TAB_SIMULATED_ANNEALING && verbose)
                std::cout<<"\n";

        if (impro && (min_pivot_improvement == 0 || (min_pivot_improvement != 0
&& min_pivot_improvement*current_objective <= -max_change)))
        {
                if (verbose)
                {
                        if (as_lower)
                                std::cout<<"Pivoting "<<change_in_edge<<" <--> "<<cha
nge_out_edge<<" as lower.\n"<<std::flush;
                        else
                                std::cout<<"Pivoting "<<change_in_edge<<" <--> "<<cha
nge_out_edge<<" as upper.\n"<<std::flush;
                        std::cout<<"Old objective value: "<<current_objective<<".\n"<<st
d::flush;
                        std::cout<<"New objective value: "<<current_objective+max_chan
ge<<".\n"<<std::flush;
                }
                else
                {
                        std::cout<<current_objective+max_change<<"\n"<<std::flus
h;
                }

                if (search == TAB_SIMPLE_TABU_SEARCH)
                {
                        //make this one forbidden
                        ...

                        if (ts_memory_deque.size() > (uint) ts_memory_length)
                                ts_memory_deque.pop_front();

                        if (best_objective > current_objective+max_change)
```

```
                        {
                                best_objective = current_objective+max_change;
                                best_solution = q_member;
                        }
                }


                swap_edges(change_in_edge, change_out_edge, as_lower);
                improved_last_time = true;

                if (presentation) getchar();

                return true;
        }
        else
        {
                if (verbose)
                        std::cout<<"No improving pivot found.\n"<<std::flush;

                if (search == TAB_SIMPLE_TABU_SEARCH)
                {
                        ts_recreate_best();
                }
                else if (search_impro == DYNAMIC && min_pivot_improvement != 0)
                {
                        if (verbose) std::cout<<"Adjusting minimum pivot improvement to "<
<min_pivot_improvement*dynamic_pivot_factor<<".\n";
                        min_pivot_improvement*=dynamic_pivot_factor;
                        if (improved_last_time)
                        {
                                improved_last_time = false;
                                return pivot();
                        }
                }
        }

        if (presentation) getchar();
        return false;

}

void simplex::non_periodic()
{
        property_map<Graph,edge_maxtime_t>::type max = get(edge_maxtime, *g);
        property_map<Graph,edge_mintime_t>::type min = get(edge_mintime, *g);
        property_map<Graph,edge_weight_t>::type w = get(edge_weight, *g);

        if (verbose) std::cout<<"Creating Goblin Controller.\n"<<std::flush;
        CT_NW_Simplex = new goblinController();
    CT_NW_Simplex -> traceLevel = 0;
    CT_NW_Simplex -> methMCF = abstractMixedGraph::MCF_BF_SIMPLEX;

    //CT->logEventHandler = &myLogEventHandler;

    if (verbose) std::cout<<"Creating dual problem instance.\n"<<std::flush;

    sparseDiGraph *goblin_graph = new sparseDiGraph((TNode)0, *CT_NW_Simplex);
    sparseRepresentation* goblin_rep = static_cast<sparseRepresentation*> (gobli
n_graph->Representation());

    graph_traits<Graph>::out_edge_iterator out_i, out_end;
    graph_traits<Graph>::in_edge_iterator in_i, in_end;
```



```
    int count_edges=0;

    for (uint i=0; i<num_vertices(*g); i++)
    {
                goblin_graph->InsertNode();
                int demand=0;
                for (tie(out_i, out_end) = out_edges(graph_vertices.left.at(i) ,
 *g); out_i != out_end; ++out_i)
                        demand-=w[*out_i];
                for (tie(in_i, in_end) = in_edges(graph_vertices.left.at(i) , *g
); in_i != in_end; ++in_i)
                        demand+=w[*in_i];

                goblin_rep->SetDemand(i,demand);
    }

    for (uint i=0; i<num_edges(*g); i++)
    {
        Edge e = graph_edges.left.at(i);
        goblin_graph->InsertArc(graph_vertices.right.at(source(e,*g)), graph_ver
tices.right.at(target(e,*g)),InfCap,-(min[e]-modulo_param[graph_edges.right.at(e
)]*period),0);
        goblin_graph->InsertArc(graph_vertices.right.at(target(e,*g)), graph_ver
tices.right.at(source(e,*g)), InfCap,max[e]-modulo_param[graph_edges.right.at(e)
]*period,0);
    }


    if (verbose) std::cout<<"Invoking goblin solver.\n"<<std::flush;
    double return_val = goblin_graph->MinCostBFlow();
    if (verbose) std::cout<<"Objective value: "<<return_val<<"\n"<<std::flush;


        ...
}

void simplex::solve()
{
        ...

        bool sa_hybrid_return = false;

        cut.active = false;
        do
        {
                if (cut.active)
                {
                        if (verbose) std::cout<<"\n*** Cut found, applying it. ***\n\n"<<std
::flush;
                        transform();
                        if (presentation) getchar();


                        if (verbose) std::cout<<"\n*** Solving non-periodic flow. ***\n\n"<
<std::flush;
                        non_periodic();


                        if (verbose) std::cout<<"\n*** Building simplex tableau. ***\n\n"<<s
td::flush;
                        build();
                        if (presentation) getchar();
                }
                else if(sa_hybrid_return && !sa_hybrid_active)
```

```
                {
                        sa_hybrid_return = false;
                        sa_hybrid_active = true;
                }

                if (verbose) std::cout<<"\n*** Pivoting. ***\n\n"<<std::flush;
                int pivot_count = 1;
                if (!verbose) std::cout<<get_objective()<<"\n";
                while(pivot() && (search != TAB_SIMPLE_TABU_SEARCH || pivot_coun
t < ts_max_iterations) && (!countpercentages || pivot_count<100))
                {
                        if (verbose) std::cout<<"*** "<<pivot_count<<" Pivot. ***\n"<
<std::flush;
                        ++pivot_count;
                };

                if (search == TAB_SIMPLE_TABU_SEARCH && pivot_count == ts_max_it
erations)
                        ts_recreate_best();

                ...

                std::cout<<"\n*** Searching for an improving cut. ***\n\n"<<std::flush;
                improvable();
                if (presentation) getchar();

                if (cut.active == false && (search == TAB_STEEPEST_SA_HYBRID &&
!sa_hybrid_active))
                        sa_hybrid_return = true;
        }
        while (cut.active || sa_hybrid_return);

        ...
}

void simplex::build()
{
        find_modulo();

        graph_traits<Graph>::edge_iterator ei;
        property_map<Graph,edge_maxtime_t>::type max = get(edge_maxtime, *g);
        property_map<Graph,edge_mintime_t>::type min = get(edge_mintime, *g);
        property_map<Graph,vertex_index_t>::type index = get(vertex_index, *g);
        property_map<Graph,edge_weight_t>::type w = get(edge_weight, *g);

        if (verbose)
                std::cout<<"Calculating slack for tree edges.\n"<<std::flush;
        //calculate slack: in edges
        for (uint k = 0; k<in_tree_edges.size(); k++)
                if (lower_tree_edges[k])
                        m_slack[in_tree_edges[k]] = 0;
                else
                        m_slack[in_tree_edges[k]] = max[graph_edges.left.at(in_t
ree_edges[k])] - min[graph_edges.left.at(in_tree_edges[k])];

        if (verbose)
                std::cout<<"Calculating slack for non-tree edges.\n"<<std::flush;
        //calculate slack: out edges
        for (uint k = 0; k<out_tree_edges.size(); k++)
        {
                int slack = m_pi[graph_vertices.right.at(target(graph_edges.left
.at(out_tree_edges[k]), *g))] - m_pi[graph_vertices.right.at(source(graph_edges.
```



```
left.at(out_tree_edges[k]), *g))];
                while (slack > max[graph_edges.left.at(out_tree_edges[k])])
                        slack-=period;
                while (slack < min[graph_edges.left.at(out_tree_edges[k])])
                        slack+=period;
                slack-=min[graph_edges.left.at(out_tree_edges[k])];
                m_slack[out_tree_edges[k]] = slack;
        }

        coeff.clear();
        for (uint i=0; i<num_edges(*g); i++)
                columns[i].clear();
        columns.clear();

        for (uint i = 0; i<num_edges(*g); i++)
                m_rhs[i]=m_slack[i];

        if (verbose)
                std::cout<<"Calculating tableau coefficients.\n"<<std::flush;

        for (uint i=0; i<out_tree_edges.size(); i++)
        {
                ...
        }
}
```

Literaturverzeichnis

AMO93 Ahuja, Ravindra ; Magnanti, Thomas ; Orlin, James: *Network flows: theory, algorithms, and applications*. Prentice-Hall, 1993

Boo09 *BOOST C++ Libraries*. 07.2009. – http://www.boost.org/

BS80 Bronstein, I. N. ; Semendjajew, K.A.: *Taschenbuch der Mathematik*. 19. Verlag Harri Deutsch, 1980

Che76 Chen, Wai-Kai: *Applied Graph Theory*. 2. North-Holland Publishing Company, 1976

Com09 *Der „Complexity Zoo“ im Quantenphysik-Wiki*. 07.2009. – http://qwiki.stanford.edu/index.php?title=Complexity_Zoo&oldid=12051

CW93 Corcoran, Arthur L. ; Wainwright, Roger L.: LibGA: a user-friendly workbench for order-based genetic algorithm research. In: *SAC '93: Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing*, ACM, 1993

DS84 Doyle, Peter G. ; Snell, J. L.: *Random Walks and Electric Networks*. The Mathematical Association of America, 1984 (The Carus Mathematical Monographs 22)

Fuh08 Fuhse, Rasmus: *Heuristiken zur Erstellung von Linienkonzepten*, Universität Göttingen, Diplomarbeit, 2008

Glo89 Glover, Fred: Tabu Search - Part I. In: *ORSA Journal on Computing* 1 (1989), Nr. 3

Glo90 Glover, Fred: Tabu Search - Part II. In: *ORSA Journal on Computing* 2 (1990), Nr. 1

GLP09 *GNU Linear Programming Kit*. 07.2009. – http://www.gnu.org/software/glpk/

Gob09 *GOBLIN Graph library*. 07.2009. – http://goblin2.sourceforge.net/

Hus05 Huschke, Reinhard: Schneller umsteigen. In: *Berliner Zeitung* (11.2005). – www.berlinonline.de/berliner-zeitung/archiv/.bin/dump.fcgi/2005/1109/wissenschaft/0002/index.html

KHA+09 Kroon, Leon ; Huisman, Dennis ; Abbink, Erwin ; Fioole, Pieter-Jan ; Fischetti, Matteo ; Maróti, Gábor ; Schrijver, Alexander ; Steenbeek, Adri ; Ybema, Roelof: The New Dutch Timetable: The OR Revolution. In: *Interfaces* 39 (2009), Nr. 1, S. 6–17

KJV83 Kirkpatrick, S. ; Jr., C. D. G. ; Vecchi, M. P.: Optimization by Simulated Annealing. In: *Science* 220 (1983), Nr. 4598

Lie08 Liebchen, Christian: The First Optimized Railway Timetable in Practice. In: *Transportation Science* 42 (2008), Nr. 4, S. 420–435

McK81 McKay, Brendan: Practical Graph Isomorphism. In: *Congressus Numerantium* 30 (1981), S. 45–87

Nac98 Nachtigall, Karl: Periodic Network Optimization and Fixed Interval Timetables. (1998). – Habilitationsschrift

NO08 Nachtigall, Karl ; Opitz, Jens: *Solving Periodic Timetable Optimisation Problems by Modulo Simplex Calculations*. 2008. – ATMOS 2008
8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2008/1588

NW88 Nemhauser, George L. ; Wolsey, Laurence A.: *Integer and combinatorial optimization*. Wiley, 1988

Rec73 Rechenberg, Ingo: *Evolutionsstrategie*. Friedrich Frommann Verlag, 1973 (problemata frommann-holzboog)

Sch07 Schöbel, Anita: *Vorlesungsskript „Einfuehrung in die Optimierung“*. 2007

Sch09 Schöbel, Anita: *Skript „Optimization Models in Public Transportation“*. 2009

SS09 Schachtebeck, Michael ; Schöbel, Anita: LinTim - A Toolbox for the Experimental Evaluation of the Interaction of Different Planning Stages in Public Transportation / ARRIVAL - Algorithms for Robust and online Railway optimization: Improving the Validity and reliAbility of Large scale systems. 2009 (0206). – Forschungsbericht

SU89 Serafini, Paolo ; Ukovich, Walter: A Mathematical Model for Periodic Scheduling Problems. In: *SIAM J. Disc. Math.* 2 (1989), Nr. 4, S. 550–581

Č85 Černý, V.: Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. In: *Journal of Optimization Theory and Applications* 45 (1985), January, Nr. 1

Abbildungsverzeichnis