Georg-August-Universität Göttingen
Fakultät für Mathematik und Informatik
Mathematik Bachelor of Science

Bachelorarbeit

# Integration von Fahrzeugumlaufplanung und Verspätungsmanagement

vorgelegt von

**Rebecca Nahme**

am 01.08.2012

Betreuerin:
Prof. Dr. Anita Schöbel

Zweitgutachter:
Jun.-Prof. Dr. Stephan Westphal

# Inhaltsverzeichnis

# 1 Motivation

## 1.1 Problemstellung

Verspätungen bei Zügen gehören mittlerweile zum unangenehmen Alltag der Bahnreisenden. Insbesondere Streiks, wie der der Lokführer im Frühjahr 2011 sowie zu Beginn des Jahres 2012, oder Schnee und Eis in den Wintermonaten sorgen regelmäßig für außerplanmäßige Ankunftszeiten von Bahnen im öffentlichen Personennahverkehr und -fernverkehr. Auch andere unvorhersehbare Ereignisse, angefangen bei defekten Signalanlagen bis hin zu bewusst getätigten Anschlägen auf die Gleise, stören immer wieder die Einhaltung der geplanten Abläufe, die besonders darauf ausgelegt sind, den Reisenden ihre Fahrt so komfortabel wie möglich zu gestalten. Dieses Ziel hat bei auftretenden Verspätungen weiterhin enorme Priorität, weshalb durch spontane Umstrukturierungen der eigentlichen Planung größere Unannehmlichkeiten weitmöglichst vermieden werden.

Als besonders kritisch bei Verzögerungen im Betriebsablauf stellen sich sogenannte Zugumläufe an (End-)Bahnhöfen heraus. Bei einem Zugumlauf bedient ein Fahrzeug nach Beendigung der ersten Strecke planmäßig eine neue Strecke. Im einfachsten Fall ist dies zunächst eine Strecke von A nach B gefolgt von der umgekehrten Strecke, B nach A. Dieses Szenario kann an einem Tag natürlich mehrfach wiederholt werden, wie es beispielsweise bei den Fahrzeugen der Eisenbahngesellschaft Cantus zwischen Göttingen und Kassel der Fall ist (1.1).

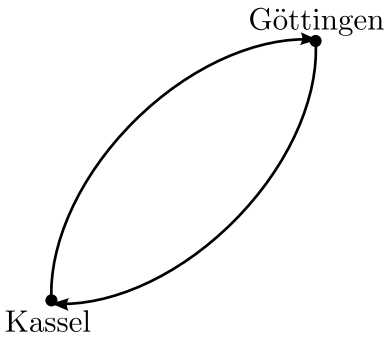



Abb. 1.1: Einfacher Umlauf am Beispiel des Cantus zwischen Göttingen und Kassel

Beendet ein Zug verspätet seine erste Tour, so besteht die Gefahr, dass sich diese

Verspätung auf die Strecke fortpflanzt, die dieser Zug im Anschluss bedient. Dies gilt es bestmöglichst zu vermeiden.

Eine naheliegende Lösung des Problems scheint das Aufbrechen und eine Neuordnung der Umläufe im Betriebsablauf entsprechend der aktuellen Gegenbenheiten zu sein. Wie effektiv dieses Verfahren eingesetzt werden kann, wird zunächst an einem Beispiel demonstriert und im weiteren Verlauf der vorliegenden Arbeit durch Betrachtung einer Reihe von Testdaten genauer untersucht.

Unser Beispiel beschränke sich auf zwei Bahnhöfe und fünf Züge (1.1). Die Züge A, B und D erreichen den jeweiligen Bahnhof auf Grund von außerplanmäßigen Verzögerungen so verspätet, dass sie ihre folgenden Touren nicht pünktlich starten können.

(a) Fahrplan

| | Bahnhof I | | Bahnhof II | |
|---|---|---|---|---|
| Zug | Ankunft | Abfahrt | Ankunft | Abfahrt |
| A | - | - | 11.01 | 11.39 |
| B | 8.05 | 8.30 | - | - |
| C | 8.17 | 8.43 | 11.14 | 11.55 |
| D | 8.31 | 8.59 | - | - |
| E | 8.38 | 9.08 | - | - |

(b) verspätete Ankunftszeiten

| Zug | Ankunft an Bahnhof I | Ankunft an Bahnhof II |
|---|---|---|
| A | - | 12.02 (+61) |
| B | 8.32 (+27) | - |
| C | 8.17 (+0) | 11.14 (+0) |
| D | 9.01 (+30) | - |
| E | 8.42 (+3) | - |

(c) Fahrplan mit angepassten Umläufen

| | Bahnhof I | | Bahnhof II | |
|---|---|---|---|---|
| Zug | Ankunft | Abfahrt | Ankunft | Abfahrt |
| A | - | - | 12.02 | 11.55 |
| B | 8.32 | 8.43 | 11.14 | 11.39 |
| C | 8.17 | 8.30 | - | - |
| D | 9.01 | 9.08 | - | - |
| E | 8.42 | 8.59 | - | - |

Tab. 1.1: Beispiel

Durch einfaches Umsortieren der Umläufe lässt sich an Bahnhof I eine Fortpflanzung der Verspätungen gänzlich verhindern und an Bahnhof II zumindest reduzieren.

## 1.2 LinTim und Vorarbeit

Die Arbeitsgruppe Optimierung um Prof. Anita Schöbel am Instituts für Numerische und Angewandte Mathematik der Georg-August-Universität Göttingen entwickelt die Softwaresammlung LinTim (*Lin*eplanning and *Tim*etabling) für Probleme der Linienplanung und Fahrplangestaltung sowie für das Verspätungsmanagement und Umlaufplanung im öffentlichen Verkehr. Einen allgemeinen Überblick über die Ideen und Grundlagen von LinTim stellt [Sch04] bereit.

Das Delay-Management und die Fahrzeugumlaufplanung stellten dabei bisher voneinander unabhängige und getrennte Verfahren dar. Dass sich aber genau diese Verknüpfung zur weiteren Optimierung der Softwaresammlung LinTim eignen kann, hat die genaue Betrachtung des Einflusses der Umlaufkantenwahl bei der Ausführung des Delay-Managements in [Nah11] gezeigt. Auf dieser Erkenntnis baut die im Folgenden vorgelegte Arbeit auf.

# 2 Fahrzeugumlaufplanung und Verspätungsmanagement

## 2.1 Derzeitige, auf einem EAN basierende Abläufe

Grundlage der betrachteten Abläufe stellt ein *Ereignis-Aktivitäts-Netzwerk* (EAN) dar. In ihm werden die Daten eines Fahrplans derart gespeichert, dass die Kanten in diesem Graphen die Aktivitäten der Züge charakterisieren. Das heißt, es gibt neben Fahrkanten, auch Wartekanten für einen Zug. Die Knoten symbolisieren den Übergang von einer Aktivität in eine neue. Somit gehören zu einem Bahnhof und einem Zug stets zwei Knoten, der erste markiert das Halten des Zuges im Bahnhof, wo er dann stehen bleibt, bis der zweite Knoten seine Abfahrt kennzeichnet.

**Definition 2.1.1** *Ein Ereignis-Aktivitäts-Netzwerk (EAN) ist ein Graph* $\mathcal{N} = (\mathcal{E}, \mathcal{A})$*, dessen Knoten* $\mathcal{E}$ *Ereignisse und dessen Kanten* $\mathcal{A}$ *Aktivitäten im öffentlichen Verkehr darstellen. Dabei setzen sich die Knoten aus Ankunfts- und Abfahrtsereignissen,* $\mathcal{E} = \mathcal{E}_{arr} \cup \mathcal{E}_{dep}$*, und die Kanten aus Fahr-, Warte- und Umsteigeaktivitäten,* $\mathcal{A} = \mathcal{A}_{drive} \cup \mathcal{A}_{wait} \cup \mathcal{A}_{change}$*, zusammen.*

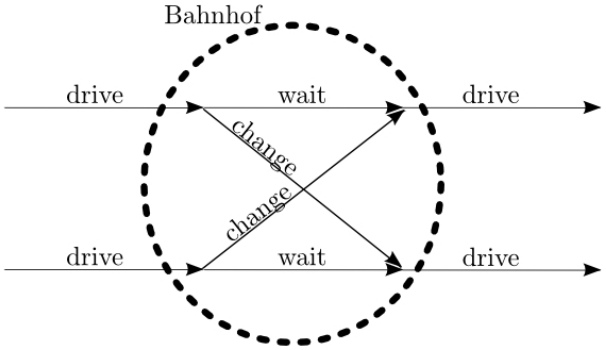



Abb. 2.1: Ausschnitt eines EAN mit einem Bahnhof und zwei Touren

**Definition 2.1.2** *Auf den Knoten- und Kantenmengen des EAN* $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ *seien noch folgende Teilmengen definiert:*

- $\mathcal{E}_{start}$ *... Menge der Startereignisse aller Trips*

- $\mathcal{E}_{end}$ *... Menge der Endereignisse aller Trips*
- $\mathcal{E}_{term}$ *... Menge der fiktiven Ereignisse, die einen Zug ins Depot überführen*
- $\mathcal{E}_{init}$ *... Menge der fiktiven Ereignisse, die einen Zug aus dem Depot holen*
- $\mathcal{A}_{train} = \mathcal{A}_{drive} \cup \mathcal{A}_{wait}$
- $\mathcal{A}_{head}$ *... Menge der Headway-Aktivitäten*
- $\mathcal{A}_{circ}$ *... Menge der Umlaufkanten (*$\mathcal{A}_{circ} \subset (\mathcal{E}_{end} \cup \mathcal{E}_{init}) \times (\mathcal{E}_{start} \cup \mathcal{E}_{term})$*)*

Für weitere Ausführungen die verschiedenen Mengen betreffend sei auf [Sch09] verwiesen.

Nachdem ein Netzplan, mehrere Linien und die zugehörigen Fahrpläne generiert wurden, rollt man diese zu einem aperiodischen EAN aus, das den zeitlichen Zusammenhang der einzelnen Komponenten wiedergibt. Im weiteren Verlauf beschränkt man sich damit lediglich auf einen zeitlich begrenzten, selbst gewählten Abschnitt, auf dem bisher wiederum die Fahrzeugumlaufplanung unabhängig vom Verspätungsmanagement zum Einsatz kommt.

Einen wichtigen Bestandteil der Fahrzeugumlaufplanung stellen *Umlaufkanten* dar. Sie dienen der Verbindung verschiedener Fahrten, die durch den gleichen Zug abgedeckt werden sollen. Das heißt, dass ein Zug nach Beendigung seiner aktuellen Tour, ausgehend vom erreichten Endbahnhof eine neue Fahrt übernehmen kann. Eine Umlaufkante kann sich graphentheoretisch entsprechend so vorgestellt werden, dass sie als Kante innerhalb eines Bahnhofs das Ende einer Tour mit dem Anfang einer anderen Tour verbindet. Sie werden auf Basis der ursprünglichen Ankunfts- und Abfahrtszeiten unabhängig von den erzeugten Verspätungen angelegt.

Das implementierte Vehicle-Scheduling-Verfahren in LinTim basiert auf dem Kanalmodell, welches in [Uff10] genauer betrachtet wird.

In der operativen Phase der Planung wird zudem mit Hilfe des *Delay-Managements* auf unvorhersehbare, aktuelle Störugen der geplanten Abläufe reagiert, so dass Auswirkungen auf die Passagiere minimal bleiben. Hierbei werden ausge¿Xhend von auf dem Fahrplan erzeugten Verspätungen Entscheidungen getroffen; beispielsweise ob also ein Anschluss aufrechterhalten bleiben soll, ob ein Zug auf einen verspäteten wartet, um den Umstieg für die betroffenen Reisenden zu sichern, oder welcher der Züge, die dieselben Gleise nutzen, als erster startet. Dies geschieht mit der Absicht, die Unnannehmlichkeiten der Reisenden bestmöglichst zu minimieren. Für detailliertere Erläuterungen hinsichtlich des verwendeten Delay-Management-Verfahrens kann in [Sch09] nachgelesen werden.

## 2.2 Verbindung von Fahrzeugumlaufplanung und Verspätungsmanagement

Die Arbeit [Nah11] schätzt experimentell auf Grundlage verschiedener Hochgeschwindigkeits-Intercity-Schienennetze Bahn-gross und Bahn-klein den Nutzen einer Integration der Fahrzeugumlaufplanung in das Delay-Management ab. Hierfür wurde betrachtet, welche Resultate sich beim Verspätungsmnagement einerseits ohne und andererseits mit Berücksichtigung von Umlaufkanten ergeben. Dies konnte soweit realisiert werden, dass man die Umlaufkanten bei den Berechnungen im Delay-Management entweder vollständig vernachlässigt oder sie ungeachtet der bekannten Verspätungen als festen Bestandteil der Planung integriert. Durch Evalution der Ergebnisse ergeben sich pro Instanz zwei verschiedene Datensätze. Zwischen den jeweiligen Werten dieser beiden Szenarien zeigte sich ein deutlicher Unterschied, der auf einen lohnenswerten Nutzen der geplanten variablen Integration der Umlaufkanten, also auf eine Verringerung der Fortpflanzung von Verspätungen bei entsprechender Anpassung der Umlaufkanten, schließen lässt. Beispielsweise erhöhte sich mit dem Einbinden der Umlaufkanten in den Ablauf die Gesamtanzahl verspäteter Ereignisse im Netzwerk, wie im Plot 2.2 ersichtlich wird.

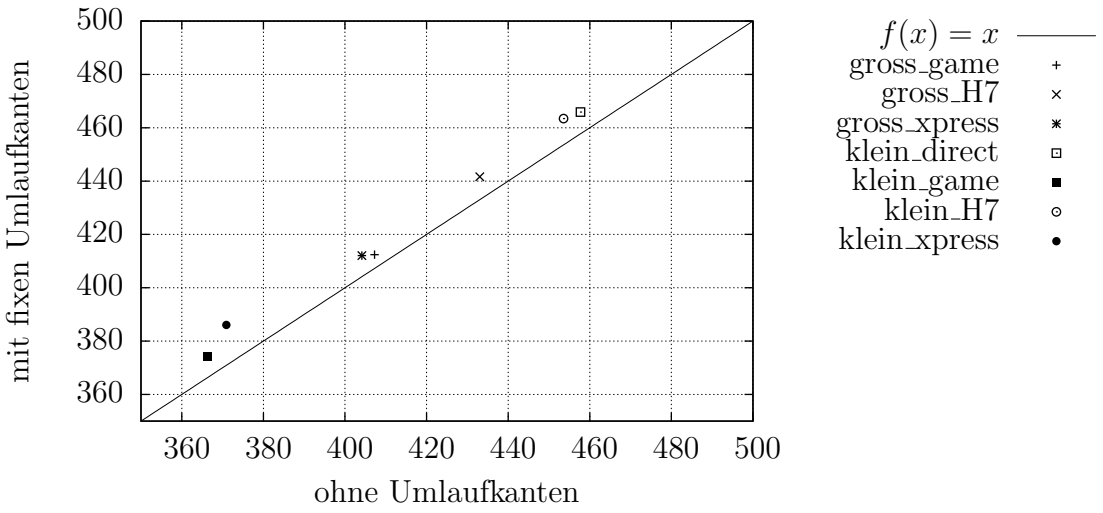



Abb. 2.2: Anzahl der verspäteten Ereignisse ohne sowie mit integrierten Umläufen im Vergleich (Quelle: [Nah11])

Die eigentliche Verbindung von Fahrzeugumlaufplanung und Verspätungsmanagement wird als lokaler Prozess an den entsprechenden Bahnhöfen realisiert, wobei sich jede Neuordnung an einem Bahnhof auf die sich zeitlich anschließenden Züge, Fahrtzeiten etc. auswirkt.

Mathematisch betrachtet führt das Problem der variablen Umlaufkanten an einem Bahnhof zum sogenannten *Matching-Problem*, welches ein Bestandteil der Graphentheorie ist. Im Folgenden werden zunächst einige theoretische Grundlagen vorgestellt, um mit deren Hilfe im weiteren Verlauf der Arbeit eine Lösung für das gegebene Matching-Problem zu entwickeln.

# 3 Das Matching-Problem

## 3.1 Theoretische Grundlagen

### 3.1.1 Grundbegriffe der Graphentheorie

Für eine fachgerechte Betrachtung des gestellten Problems bedarf es einiger graphentheoretischer Grundlagen, die in diesem Kapitel dargestellt werden. Dabei dient neben dem Buch [Bü10], auch die Vorlesung "Netzwerkflüsse"[Wes11] als Grundlage.

Ausgangspunkt von definierten Begriffen, angeführten Folgerungen sowie Sätzen stellt ein ungerichteter Graph $G$ dar, bestehend aus einer Knotenmenge $V$ und einer Kantenmenge $E$. Man schreibt auch $G = (V, E)$.

**Definition 3.1.1** *Sei $G = (V, E)$.*

1. *Mit $\gamma(e)$ für eine Kante $e \in E$ bezeichnet man die Menge der Knoten, die $e$ miteinander verbindet.*

2. *Einen Knoten $v \in V$ und eine Kante $e \in E$ nennt man inzident, wenn $v \in \gamma(e)$.*

3. *Zwei Kanten heißen adjazent, wenn ein Knoten $v \in V$ existiert mit $v \in \gamma(e_1) \cap \gamma(e_2)$.*

Als Folge dieser Beziehungen lassen sich Kanten in einem Graphen durch die zu ihnen inzidenten Knoten darstellen. Dabei ist diese Identifizierung in einem ungerichteten Graphen nur dann eindeutig, wenn es keine zwei Kanten $e_1$ und $e_2$ gibt, für die gilt $\gamma(e_1) = \gamma(e_2)$.

**Definition 3.1.2** *Ein ungerichteter Graph $G$ heißt einfach, wenn keine zwei Kanten zu den gleichen Knoten inzident sind, d.h. wenn $\gamma(e_1) \neq \gamma(e_2)\ \forall e_1 \neq e_2 \in V$.*

**Bemerkung 3.1.3** *Eine Kante $e$ eines einfachen, ungerichteten Graphen $G$ kann auch durch die zu ihr inzidenten Knoten $v_1,\ v_2$ eindeutig identifiziert werden und man schreibt $e = (v_1, v_2)$.*

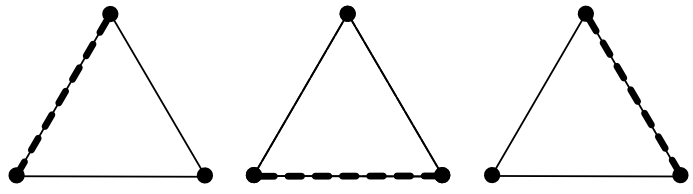
**Definition 3.1.4** *Einen ungerichteteten Graphen $G$ nennt man vollständig, wenn zu allen Knotenpaaren $(u, v) \in V \times V$ eine Kante $e \in E$ existiert, die diese beiden Knoten miteinander verbindet, d.h. wenn alle Knoten zueinander adjazent sind.*

Die eingeführten Begriffe sollen nun auf das Ausgangsproblem, die Zuordnung von Zügen und Abfahrtszeiten an einem Bahnhof, übertragen werden. Hierfür wird nun ein graphentheoretischer Rahmen gelegt, um dieses Problem mathematisch zu formulieren.

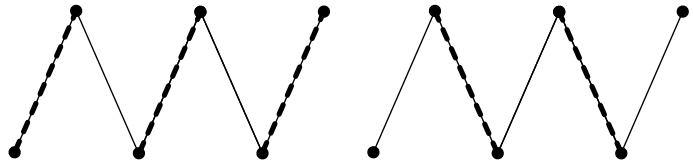
**Definition 3.1.5** *Ein Matching (oder auch eine Paarung) in einem Graphen $G$ ist eine Kantenmenge $M \subseteq E$, für die gilt, dass es keine adjazenten Kanten in $M$ gibt.*

**Definition 3.1.6** *Ein Matching $M$ eines Graphen $G$ heißt maximal, wenn $M$ um keine weitere Kante erweitert werden kann, ohne dass die Matching-Eigenschaft verloren geht.*

**Bemerkung 3.1.7** *Generell muss ein maximales Matching in einem Graphen nicht eindeutig sein, weder bei der Kantenmenge, noch in der Kantenanzahl (Abb. 3.1).*



(a) unterschiedliche Kantenmengen bei gleicher Mächtigkeit



(b) unterschiedlicher Kantenanzahl

Abb. 3.1: Graphen mit verschiedenen maximalen Matchings (dargestellt durch gestrichelte Linien)

**Definition 3.1.8** *Ein Matching $M$ nennt man perfekt, wenn jeder Knoten des Graphen $G$ zu genau einer Kante des Matchings $M$ inzident ist.*

Bei erneuter Betrachtung des Ausgangsproblems fällt auf, dass dieses eine Besonderheit aufweist. Es gilt, jedem Zug eine Abfahrtszeit zuzuordnen, wobei keine Matchings zwischen zwei Zügen oder zwei Abfahrtszeiten erwünscht sind. Damit vereinfacht sich die Problematik der Aufgabenstellung auf zwei Mengen, eine bestehend aus den verfügbaren Zügen, die andere aus den planmäßigen Abfahrszeiten, die jeweils innerhalb ihrer Elemente keine Verbindungen zulassen. Dies macht die genauere Untersuchung von bipartiten Graphen interessant.

**Definition 3.1.9** *Ein Graph $G = (V, E)$ heißt bipartit, falls man seine Knotenmenge in zwei disjunkte Mengen $A$, $D$ mit $A \cap D = \varnothing$ und $A \dot{\cup} D = V$ teilen kann, so dass jeweils zwischen Knoten innerhalb einer dieser Mengen keine Kanten verlaufen.*

**Definition 3.1.10** *Einen bipartiten Graphen $G$ nennt man zusätzlich regulär, wenn die disjunkten Mengen $A$ und $D$ mit $A \dot{\cup} D = V$ gleiche Mächtigkeit aufweisen, also $|A| = |D|$.*

In den folgenden Kapiteln sollen weitere Graphenstrukturen vorgestellt werden, die es ermöglichen, mit Hilfe eines Graphen beispielsweise im Alltag auftretende Transportprobleme abzubilden und zu lösen.

### 3.1.2 Kürzeste Wege

Innerhalb eines Graphen $G = (V, E)$ kann es nun interessant sein, ob es möglich ist, einen Knoten $t \in V$ ausgehend von einem anderen Knoten $s \in V$ des Graphen über existierende Kanten zu erreichen. In diesem Fall wird ein zulässiger ($s$-$t$)-Weg gesucht.

**Definition 3.1.11** *Ein ($s$-$t$)-Weg eines Graphen $G = (V, E)$ ist eine Abfolge von Knoten und Kanten $(s = v_0, e_0, v_1, ..., e_{m-1}, v_m = t)$, so dass $e_i = (v_i, v_{i+1}) \in E$ für $i = 0, ..., m-1$. Ein solcher ($s$-$t$)-Weg heißt zusätzlich Kreis, wenn $s = t$ gilt. Dagegen liegt ein einfacher ($s$-$t$)-Weg $P$ vor, wenn $P$ keinen Knoten doppelt durchläuft. $P_{[u,v]}$ bezeichnet den Teilweg zwischen zwei Knoten $u$ und $v$ von $P$.*

**Satz 3.1.12** *Sei $G = (V, E)$ ein ungerichteter Graph mit $|V| = n$ und sei $P$ ein einfacher Weg in $G$. Dann umfasst $P$ höchstens $n - 1$ Kanten.*

**Beweis:** *Weil $P$ als einfach vorausgesetzt ist, enthält er laut Definition keinen Knoten doppelt. Daher gilt: $|V(P)| \leq n$. Somit folgt aber mit einem Blick auf die Struktur eines Weges $P = (v_0, e_0, v_1, ..., e_{n-1}, v_n)$ direkt, dass $E(P) \leq n - 1$.*

□

Möchte man nun weiterhin die Wege in einem Graphen gewichten, wird eine Instanz zur Bewertung der Distanz zweier Knoten benötigt.

**Definition 3.1.13** *Mit $c : E \to \mathbb{R}$ werden die Kosten einer jeden Kante eines Graphen $G = (V, E)$ charakterisiert. Man nennt sie zudem nichtnegativ, wenn $c : E \to \mathbb{R}_{\geq 0}$, und konservativ, wenn es keinen Kreis in $G$ gibt mit negativen Kosten. Die Kosten eines ($s$-$t$)-Weges $P = (s = v_0, e_0, v_1, ..., e_{m-1}, v_m)$ sind als die Summe der Kosten seiner einzelnen Kanten definiert,*

$$c(P) = \sum_{i=0}^{m-1} c(e_i).$$

*Ein kürzester (s-t)-Weg liegt vor, wenn für alle anderen (s-t)-Wege $P'$ aus $G$ gilt:* $c(P) \leq c(P')$.

**Bemerkung 3.1.14** *Bei der Bewertung von Kanten durch eine Kostenfunktion* $c : E \to \mathbb{R}$ *gelte die Konvention:* $c(u, v) = \infty$ *für* $u, v \in V$, *aber* $(u, v) \notin E$.

Für das Auffinden eines kürzesten Weges in einem Graphen kann man beispielsweise den Moore-Bellman-Ford-Algorithmus (3.2.2) verwenden.

### 3.1.3 Maximale Flüsse

Von nun an soll ein gerichteter Graph, auch als Digraph bezeichnet, den Ausgangspunkt der Betrachtungen darstellen. Das heißt, dass jeder Kante des Graphen eine Richtung zugewiesen bekommt. Bildlich gesprochen zeigt sie damit von einem der zu ihr inzidenten Knoten weg und zum anderen hin.

**Definition 3.1.15** *Seien* $G = (V, E)$ *und* $v \in V$.

1. $\delta^-(v)$ *ist die Menge der eingehenden Kanten in* $v$.
2. $\delta^+(v)$ *umfasst die ausgehenden Kanten von* $v$.

**Definition 3.1.16** *Man nennt (G,u,s,t) ein Netzwerk, wenn in* $G = (V, E)$ *ein gerichteter Graph, in* $u : E \to \mathbb{N}$ *eine obere Kapazitätsschranke für jede Kante aus* $G$ *und mit* $s, t$ *zwei Knoten aus* $V$ *vorliegen. Eine Kantenbewertung* $f : E \to \mathbb{R}_{\geq 0}$ *heißt (s,t)-Fluss, wenn für alle Kanten* $e \in E$ *gilt* $f(e) \leq u(e)$ *und wenn* $f$ *an allen Knoten* $v \in V \backslash \{s, t\}$ *die Flusserhaltung erfüllt, wobei die Flusserhaltungsgleichung bestimmt ist durch*

$$\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = 0.$$

*Weiterhin ist* $f$ *maximal, wenn für jeden anderen (s-t)-Fluss* $g$ *gilt* $\text{value}(f) \geq \text{value}(g)$, *wobei*

$$\text{value}(f) = \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e)$$

*den Wert des Flusses angibt.*

**Definition 3.1.17** *Seien* $G = (V, E)$ *und* $v \in V$. *Der b-Fluss eines Knoten* $v$ *ist definiert als*

$$b(v) = \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e)$$

1. *Gilt $b(v) > 0$, dann nennt man $v$ auch Quelle.*

2. *Gilt dagegen $b(v) < 0$, so bezeichnet man $v$ als Senke.*

Im Folgenden gelte für die Knoten $s, t \in G$ stets $\delta^+(s) = \varnothing$ bzw. $\delta^-(t) = \varnothing$. D.h. dass aus $s$ nur Kanten hinausführen und in $t$ wiederum nur hinein.

Auf Grundlage dieser Eigenschaften eines Graphen lässt sich nun das folgende Problem formulieren:

**Maximales Fluss-Problem 3.1.18** *Sei $(G, u, s, t)$ ein Netzwerk. Dann ist das Maximale Fluss-Problem gegeben durch*

$$\max \sum_{e \in \delta^-(s)} f(e), \text{ so dass}$$

$$\begin{array}{rcll} \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) & = & 0 & \forall v \in V \backslash \{s, t\} \\ f(e) & \leq & u(e) & \forall e \in E \end{array}$$

Aufbauend auf Kosten, Kapazität und Flusswert als Kantenbewertungen eines Graphen ergibt sich direkt eine neue Aufgabenstellung. Deren Ziel ist es, die Kosten eines maximalen Flusses eines Graphen zu minimieren.

**Minimale-Kosten-Fluss-Problem 3.1.19** *Seien $G = (V, E)$ ein gerichteter Graph, $(G, u, s, t)$ ein Netzwerk und $f : E \to \mathbb{R}_{\geq}$ ein maximaler Fluss in $G$. Das zugehörige Minimale-Kosten-Fluss-Problem ist wie folgt definiert*

$$\min \sum_{e \in E} c(e) \cdot f(e), \text{ so dass}$$

$$\begin{array}{rcll} \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) & = & 0 & \forall v \in V \backslash \{s, t\} \\ f(e) & \leq & u(e) & \forall e \in E \end{array}$$

Dieses Minimierungsproblem wird später als Grundlage dienen, das Zuordnungsproblem von Zügen und Abfahrtszeiten an einem Bahnhof graphentheoretisch abzubilden (4.1) und letztlich auch zu lösen. Hierfür wird im Kapitel 3.2.2 ein Algorithmus vorgestellt.

**Definition 3.1.20** *Sei $(G, u, s, t)$ ein Netzwerk mit einem $(s$-$t)$-Fluss, $G = (V, E)$ und $e = (u, v) \in E$. $\overleftarrow{e} = (v, u)$ symbolisiert die Rückwärtskante zu $e$, die in diesem*

*Zusammenhang wiederum auch als Vorwärtskante bezeichnet wird. Die Residualkapazitäten $u_f$ sind definiert als*

$$\begin{aligned} u_f(e) &= u(e) - f(e) \\ u_f(\overleftarrow{e}) &= f(e) \qquad\qquad \forall e \in E \end{aligned}$$

*Weiterhin stellt $G_f = (V, \{e \in E \mid u_f(e) \neq 0\} \cup \{\overleftarrow{e} \mid e \in E,\ u_f(\overleftarrow{e}) \neq 0\})$ den Residualgraphen dar.*

**Satz 3.1.21** *Sei $(G, u, s, t)$ ein Netzwerk und $f$ ein $(s$-$t)$-Fluss. Es existiert genau dann kein $(s$-$t)$-Weg im zugehörigen Residualgraphen $G_f$, wenn $f$ maximal ist.*

**Beweis:** $\underline{\Rightarrow}$**:** *Es gebe keinen $(s$-$t)$-Weg in $G_f$. Sei $R = \{v \in V \mid$ es gibt einen $(s$-$v)$-Weg in $G_f\} \Rightarrow s \in G_f,\ t \notin G_f$. Betrachte eine $R$ verlassende Kante $(u, v) \in \delta^+(R)$. Dann gilt:*
*$u \in R,\ v \notin R$*
*$\Rightarrow (u, v) \notin E(G_f)$, sonst wäre $v$ von $s$ aus erreichbar*
*$\Rightarrow f(u, v) = u(u, v)$. Sei nun $(u, v) \in \delta^-(R) \Rightarrow u \notin R,\ v \in R. \Rightarrow f(u, v) = 0$, ansonsten $\overleftarrow{(u, v)} \in G_f$ und somit wäre $u$ von $s$ aus erreichbar. Insgesamt folgt dann:*

$$\begin{aligned} \text{value}(f) &= \sum_{e \in \delta^+(R)} f(e) + \sum_{e \in \delta^-(R)} f(e) \\ &= \sum_{e \in \delta^+(R)} u(e) + \sum_{e \in \delta^-(R)} 0 \end{aligned}$$

*Damit reizt $f$ bereits alle Kapazitäten der Kanten aus, die $R$ verlassen, und lässt sich somit nicht mehr erhöhen. Da nun aber nach Definition $s \in R$, folgt, dass $f$ maximal ist.*

$\underline{\Leftarrow}$**:** *Sei $f$ maximal. Annahme: Es gibt einen Weg $P$ in $G_f$.*
*Es gilt $u_f(P) = \min\limits_{e \in P} u_f(e) > 0$. Sei weiterhin*

$$g(e) = \begin{cases} f(e) + u_f(P) & , \ e \in P \\ f(e) - u_f(P) & , \ \overleftarrow{e} \in P \\ f(e) & , \ \textit{sonst} \end{cases}$$

*wodurch ein neuer $(s$-$t)$-Fluss definiert wurde, da die Kapazitätsbeschräkungen jeder Kante $e \in E$ sowie die Flusserhaltung an jedem $v \in V \backslash \{s, t\}$ eingehalten werden, was im Folgenden zu prüfen ist:*

$$\begin{aligned} e \in P: \quad 0 \leq f(e) &< \underbrace{f(e) + u_f(P)}_{g(e)} \\ &\leq f(e) + u_f(e) = f(e) + u(e) - f(e) = u(e) \end{aligned}$$

$$\overleftarrow{e} \in P: \quad u(e) \geq f(e) \quad > \quad \underbrace{f(e) - u_f(P)}_{g(e)}$$
$$\geq \quad f(e) - u_f(\overleftarrow{e}) = f(e) - f(e) = 0$$

$v \in V \backslash \{s,t\}$ : *Relevant sind lediglich $v \in P$, da sich nur hier der Fluss verändert. Gleichzeitig nimmt der Flusswert aber sowohl auf der Kante $e \in \delta^+(v)$ als auch auf $f \in \delta^-(v)$ um $u_f(P)$ zu, womit die Flusserhaltungsgleichung erfüllt bleibt.*

*Dann folgt aber nach Definition von $g$:* $\text{value}(f) < \text{value}(g)$. *Dies ist ein Widerspruch zur vorausgesetzten Maximalität von $f$. D.h. es existiert kein $(s$-$t)$-Weg $P$ in $G_f$.*

□

Der Satz 3.1.21 liefert ein Optimalitätskriterium für maximale Flüsse. Von diesem wird später im Successive-Shortest-Path-Algorithm (3.2.2) noch Gebrauch gemacht.

### 3.1.4 Matchings in bipartiten Graphen

Im vorliegenden Kapitel werden spezielle Eigenschaften von Matchings in bipartiten Graphen untersucht. Dies ist gerade deshalb von besonderem Interesse, da sich einige Problemstellungen unter diesen Voraussetzungen vereinfachen lassen.

**Satz 3.1.22 (Satz von Hall)** *Sei $G = (V, E)$ ein bipartiter Graph mit $A \dot\cup D = V$. So besitzt $G$ ein $A$ überdeckendes Matching genau dann, wenn für jedes $X \subseteq A$ gilt*

$$|X| \leq |\{v \in D \mid \exists u \in X: \ (u,v) \in E\}|.$$

**Bemerkung 3.1.23** *Ein recht populärer Spezialfall des Satzes von Hall ist auch unter dem Namen Heiratsproblem bekannt, bei dem zusätzlich $|A| = |D|$ gefordert und nach einem perfekten Matching gesucht wird.*

Im Gegensatz zu Matchings in allgemeinen Graphen kann dieses Zuordnungsproblem bei vorhandener Bipartitheit des betrachteten Graphen auf das Maximale Fluss-Problem (3.1.18) zurückgeführt werden.

Sei $G = (V, E)$ ein bipartiter Graph mit $A \dot\cup D = V$ und jeder Knoten aus $A$ sei zu jedem Knoten aus $D$ adjazent. Man füge dem Graphen nun zwei weitere Knoten hinzu, $s$ und $t$, wobei man alle Knoten von $A$ mit $s$ und die aus $D$ wiederum mit $t$ verbinde, wie in Abb. 3.2 veranschaulicht. In dem so entstanden neuen Graphen $G' = (V', E')$ bewerte man alle Kanten mit einer Kapazität $u$ von 1. Auf diese

Weise ist gesichert, dass zwischen zwei Knoten des Graphen keine zwei Flusseinheiten gleichzeitig transportiert werden. Wenn nach diesen Schritten im Netzwerk $(G', u, s, t)$ ein maximaler Fluss gesucht wird, stellen die von diesem Fluss belegten Kanten zwischen den Mengen $A$ und $D$ das gewünschte maximale Matching dar.

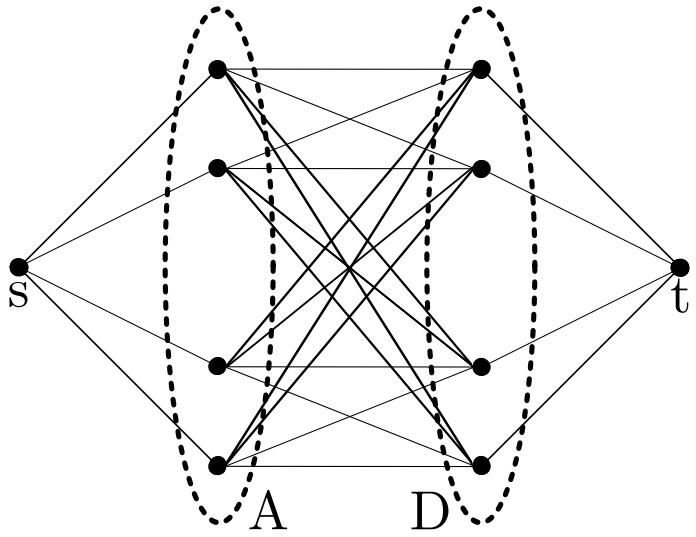



Abb. 3.2: Matchingproblem als Maximales Fluss-Problem

Unter Verwendung des vorgestellten b-Flusses 3.1.17 lässt sich die Problematik sogar ohne Hinzufügen neuer Knoten und Kanten als Fluss-Problem formulieren. Anstelle der einen Quelle $s$ werden nun einfach alle Knoten aus der Menge $A$ zu Quellen, indem man $b(v) = 1 \ \forall v \in A$ setzt, und anstelle von $t$ analog $b(v) = -1 \ \forall v \in D$. Diese Darstellung beschreibt das eigentliche Matchingproblem sogar präziser und umgeht die zusätzlichen Knoten und Kanten.

## 3.2 Algorithmen

### 3.2.1 Reiner Matching-Algorithmus

Als erste Kategorie von Lösungsmöglichkeiten des Matchingproblems soll ein Algorithmus vorgestellt werden, der auf den Matchingeigenschaften direkt aufbaut, ohne das Problem zuerst in ein anderes zu überführen. Die Ungarische Methode, auch als Kuhn-Munkres-Algorithmus bezeichnet, gehört zu den bekanntesten und besten kombinatorischen Algorithmen zum Finden eines gewichtsmaximalen perfekten Matchings.

Im Folgenden bezeichne stets $n$ die Mächtigkeit der Knotenmenge $V$ und $m$ die der Kantenmenge $E$ des Graphen $G$.

**Algorithmus 1: Ungarische Methode**

**Input:** *bipartiter Graph $G = (V, E)$, die Mengen $A$ und $D$ mit $A \dot\cup D = V$, nichtnegatives Gewicht $c = (m_{ij})$ mit Kosten der Kante $(i, j)$ mit $i \in A,\ j \in D$*

**Output:** *ein gewichtsmaximales perfektes Matching*

**Laufzeit:** $\mathcal{O}(n^3)$
**Pseudocode:**

1: Finde den minimalen Eintrag einer jeden Zeile und subtrahiere es von allen Elementen der entsprechenden Zeile.
2: Mache das gleiche für die Spalten.
3: Sei $k$ die minimale Anzahl der (horizontalen und vertikalen) Linien, die nötig ist, um alle Nulleinträge der Matrix abzudecken.
4: **while** $k < n$ **do**
5: Subtrahiere das Minimum der nicht durch Linien abgedeckten Einträge von jeder nicht abgedeckten Zahl und addiere es zu jedem Eintrag, bei dem sich zwei Linien kreuzen.
6: Bestimme $k$ neu.
7: **end while**
8: Finde in der neuen Matrix eine Kombination von Nulleinträgen, so dass es genau eine Null pro Zeile und eine pro Spalte gibt. Sie repräsentieren ein Matching.

### 3.2.2 Algorithmus für minimalen b-Fluss

Wie bereits im Kapitel 3.1.4 präsentiert, lassen sich Matchingprobleme in bipartiten Graphen auf minimale Fluss-Probleme zurückführen. Aus diesem Grund soll an dieser Stelle ein Algorithmus zur Berechnung eines minimalen b-Flusses vorgestellt werden, der im weiteren Verlauf auch das Zuordnungsproblem von Zügen und Abfahrtszeiten löst. Dabei handelt es sich um den Successive-Shortest-Path-Algorithm. Mehr Informationen zu diesem sowie anderen Algorithmen zur Lösung von Fluss-Problemen kann man im [KV06] nachlesen. Diese Literatur dient auch im Folgenden als Grundlage.

**Algorithmus 2: Successive-Shortest-Path-Algorithm**

**Input:** *Digraph $G = (V, E)$, Kapazität $u : E \to \mathbb{R}_+$, $b : V \to \mathbb{R}$ mit $\sum_{v \in V} b(v) = 0$ und konservative Kosten $c : E \to \mathbb{R}$*

**Output:** *minimaler b-Fluss $b$*

**Laufzeit:** $\mathcal{O}(nm + B(m + nlogn))$ *mit* $B = \frac{1}{2}\sum |b(v)|$
**Pseudocode:**

1: Setze $b' = b$ und $f(e) = 0\ \forall e \in E$.
2: **if** $b' = 0$ **then**
3: STOP
4: **else**
5: Wähle einen Knoten $s$ mit $b'(s) > 0$.
6: Wähle einen Knoten $t$ mit $b'(t) < 0$, so dass $t$ von $s$ aus in $G_f$ erreichbar ist.
7: **if** $\nexists$ solches $t$ **then**
8: STOP (Es gibt keinen b-Fluss.)
9: **end if**
10: **end if**
11: Finde einen ($s$-$t$)-Weg $P$ in $G_f$ mit minimalen Kosten (Moore-Bellman-Ford).
12: Bestimme $\gamma = \min\left\{\min_{e\in E} u_f(e),\ b'(s),\ -b'(t)\right\}$. Setze $b'(s) = b'(s) - \gamma$ und $b'(t) = b'(t) + \gamma$ und erhöhe den Fluss $f$ entlang $P$ um $\gamma$.

**Satz 3.2.1** *Der Successive-Shortest-Path-Algorithm arbeitet korrekt.*

Der in Schritt 11 des Successive-Shortest-Path-Algorithm zu findende kürzeste ($s$-$t$)-Weg soll mit dem nachfolgenden Moore-*Bellman-Ford-Algorithmus* bestimmt werden. Man beachte, dass dieser als Grundlage für seine Berechnungen nicht den Ausgangsgraphen $G$ des Successive-Shortest-Paths-Algorithms übergeben bekommt, sondern den Residualgraphen $G_f$.

### Algorithmus 3: Moore-Bellman-Ford-Algorithm

**Input:** *Digraph* $G = (V, E)$*, konservative Kosten* $c : E \to \mathbb{R}$ *und ein Knoten* $s \in V$

**Output:** *kürzeste Wege von* $s$ *zu allen Knoten* $v \in V$ *und ihre Längen*

**Laufzeit:** $\mathcal{O}(nm)$
**Pseudocode:**

1: Setze $l(s) = 0,\ l(v) = \infty\ \forall v \in V\backslash\{s\}$.
2: **for** $i = 1$ to $n - 1$ **do**
3: **for** each $(u, v) \in E$ **do**
4: **if** $l(v) > l(u) + c(u, v)$ **then**
5: $l(v) = l(u) + c(u, v)$
6: $p(v) = u$
7: **end if**
8: **end for**

9: **end for**

**Satz 3.2.2** *Der Moore-Bellman-Ford-Algorithm arbeitet korrekt.*

**Beweis:** *Seien* $R = \{v \in V \mid l(v) < \infty\}$ *und* $F = \{(u, v) \in E \mid u = p(v)\}$.

1. $l(v) \geq l(u) + c(u, v)\ \forall (u, v) \in F$
2. *Falls* $F$ *einen Kreis* $C$ *enthält, dann hat* $C$ *negative Gesamkosten.*
3. *Falls* $c$ *konzervativ ist, dann gilt für* $(R, F)$*:* $\delta^-(s) = \varnothing$

*zu 1.* *Sobald* $p(v) = u$ *gesetzt wird, gilt Gleichheit. Danach wird aber* $l(u)$ *nicht mehr erhöht, höchstens verringert, und bei einer Änderung von* $l(v)$ *bekommt* $v$ *einen neuen Vorgänger zugewiesen.*

*zu 2.* *Angenommen, es entsteht ein Kreis* $C$ *in* $F$ *beim Setzen von* $p(y) = x$*. Das heißt, dass davor galt*

$$l(y) > l(x) + c(x, y)$$

*und für* $(u, v) \in E(C) \backslash \{(x, y)\}$

$$l(v) \geq l(u) + c(u, v)$$

*Aufsummieren dieser Ungleichungen und Umformung ergibt*

$$0 > \sum_{e \in E(C)} c(e)$$

*zu 3.* *Da* $c$ *konservativ ist, folgt mit Punkt* 2*, dass* $F$ *keinen Kreis enthält. Außerdem gilt für* $v \in R \backslash \{s\}$*, dass auch* $p(v) \in R$*, wodurch alle Knoten in* $R$ *nach Konstruktion von* $s$ *aus erreichbar sind.*

$\Rightarrow$ $l(v)$ *ist die Länge des* $(s\text{-}v)$*-Weges in* $(R, F)\ \forall v \in R$ *zu jeder Zeit im Algorithmus.*

**Behauptung:** *Nach* $k$ *Iterationen entspricht* $l(v)$ *maximal der Länge eines kürzesten* $(s\text{-}v)$*-Weges mit höchstens* $k$ *Kanten.*

**Induktion:** *nach* $k$

*IA:* $k = 0$ *und somit* $l(s) = 0$

*IV: Die Behauptung gelte für* $k - 1$ *und sei* $P_{[s,u]}$ *ein kürzester* $(s\text{-}u)$*-Weg mit maximal* $k - 1$ *Kanten.*

*IS: Sei $P$ ein kürzester $(s\text{-}v)$-Weg mit $(u, v)$ als letzte Kante. Laut IV ist aber $(s, ..., u) = P_{[s,u]}$ ein kürzester $(s\text{-}u)$-Weg mit höchstens $k-1$ Kanten und $l(u) \leq c(P_{[s,u]})$ nach maximal $k-1$ Iterationen. Dann gilt nach der $k$-ten Iteration:*

$$l(v) \leq l(u) + c(u, v) \leq P_{[s,u]} + c(u, v) = c(P)$$

$\Rightarrow$ *Insgesamt durchläuft der Algorithmus $n-1$ Iteartionen, wonach alle gefundenen Wege eine Maximallänge von $n-1$ Kanten aufweisen und gleichzeitig aber auch alle von $s$ aus erreichbaren Knoten mit einem entsprechenden Weg versehen sind.*

□

Im Kapitel 4.3.1 wird der Successive-Shortest-Path-Algorithm auf die Ausgangsproblematik, dem Machten von Zügen und Abfahrszeiten an einem Bahnhof, angewendet und stellenweise etwas angepasst. Zuvor soll dieses Problem jedoch in ein äquivalentes b-Fluss-Problem überführt werden.

# 4 Umsetzung

## 4.1 Graphentheoretische Formulierung

Das Ausgangsproblem soll nun noch mathematisch formuliert werden. Hierfür dient [Sch09] als Grundlage für die verwendeten Bezeichnungen, das Delay-Management ohne Berücksichtigung der Fahrzeugumlaufplanung ($DM$) und für das Delay-Management mit fix integrierten Umläufen ($DMC$).

Im Folgenden sei $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ ein Ereignis-Aktivitäts-Netzwerk, wie in 2.1.1 definiert und mit den eingeführten Teilmengen in 2.1.2 versehen, mit unteren Schranken $L : \mathcal{A} \to \mathbb{N}$, einem zulässigen Fahrplan $\pi : \mathcal{E} \to \mathbb{N}$ und Zeiten $x_i$ des Ereignisses $i \in \mathcal{E}$ im Dispositionsfahrplan.

Weiterhin gelte für alle Umsteigekanten

$$z_a = \begin{cases} 0 & \text{falls Umsteigeaktivität } a \text{ aufrechterhalten} \\ 1 & \text{sonst} \end{cases}$$

und für alle Headway-Aktivitäten

$$g_{ij} = \begin{cases} 0 & \text{falls Ereignis } i \text{ vor Ereignis } j \text{ stattfindet} \\ 1 & \text{sonst} \end{cases}$$

Außerdem gelte für alle Umlaufaktivitäten folgende Zuordnung

$$v_{ij} = \begin{cases} 1 & \text{falls Aktivität } (i,j) \in \mathcal{A}_{circ} \text{ ausgewählt} \\ 0 & \text{sonst} \end{cases}$$

Letztlich werden noch Kosten bzw. Gewichte benötigt, um abhängig von den auftretenden Verspätungen einen für die Passagiere bestmöglichen Fahrplan zu zu gestalten. Dafür seien $w_i \in \mathbb{R}_{\geq 0}\ \forall i \in \mathcal{E}$ definiert als die Anzahl der Passagiere, die ihre Fahrt mit Ereignis $i$ beenden, sowie $w_a \in \mathbb{R}_{>0}\ \forall a \in \mathcal{A}_{change}$ als die Anzahl der Passagiere, die Aktivität $a$ nutzen wollen. Da im Gegensatz zum Verspätungsmanegement

mit fix integrierten Umläufen jetzt auch über diese minimiert werden soll, ist es notwendig, sie ebenfalls mit einer Bewertung zu versehen:

$$w_a = \begin{cases} \pi_j - \pi_i & \text{falls Ereignis } i \text{ vor Ereignis } j \text{ stattfindet, wobei } a = (i,j) \in \mathcal{A}_{circ} \\ 0 & \text{sonst} \end{cases}$$

Unter Verwendung der bereitgestellten Notationen lässt sich das Problem des Verspätungsmanegements mit Integration variabler Umlaufkanten wie folgt darstellen:

$$(\textbf{DMCvar}\ ) \min f(x,z,g) = \sum_{i \in \mathcal{E}} w_i(x_i - \pi_i) + \sum_{a \in \mathcal{A}_{change}} z_a w_a T + \sum_{a \in \mathcal{A}_{circ}} w_a,$$

so dass

$$\begin{aligned}
x_i &\geq \pi_i + d_i && \forall i \in \mathcal{E} && (4.1)\\
x_j - x_i &\geq L_a + d_a && \forall a = (i,j) \in \mathcal{A}_{train} && (4.2)\\
M z_a + x_j - x_i &\geq L_a && \forall a = (i,j) \in \mathcal{A}_{change} && (4.3)\\
M g_{ij} + x_j - x_i &\geq L_{ij} && \forall (i,j) \in \mathcal{A}_{head} && (4.4)\\
g_{ij} + g_{ji} &= 1 && \forall (i,j) \in \mathcal{A}_{head} && (4.5)\\
M(1 - v_{ij}) + x_j - x_i &\geq L_a && \forall a = (i,j) \in \mathcal{A}_{circ} && (4.6)\\
\sum_{i \in \mathcal{E}_{end} \cup \mathcal{E}_{init}} v_{ij} &= 1 && \forall j \in \mathcal{E}_{start} && (4.7)\\
\sum_{i \in \mathcal{E}_{end}} v_{ij} &= 1 && \forall j \in \mathcal{E}_{term} && (4.8)\\
\sum_{j \in (\mathcal{E}_{start} \cup \mathcal{E}_{term})} v_{ij} &= 1 && \forall i \in \mathcal{E}_{end} && (4.9)\\
\sum_{j \in \mathcal{E}_{start}} v_{ij} &= 1 && \forall i \in \mathcal{E}_{init} && (4.10)\\
x_i &\in \mathbb{N} && \forall i \in \mathcal{E} && (4.11)\\
z_a &\in \{0,1\} && \forall a \in \mathcal{A}_{change} && (4.12)\\
g_{ij} &\in \{0,1\} && \forall (i,j) \in \mathcal{A}_{head} && (4.13)\\
v_{ij} &\in \{0,1\} && \forall (i,j) \in \mathcal{A}_{circ} && (4.14)
\end{aligned}$$

**Satz 4.1.1** *Sei $f_{DM}^{opt}$ optimale Lösung von (DM) und $f_{DMC}^{opt}$ optimale Lösung von (DMC). Dann gilt*

$$f_{DM}^{opt} \leq f_{DMC}^{opt}$$

**Beweis:** *Die Behauptung ist klar, da (DMC) lediglich mehr Nebenbedingungen fasst als (DM) und somit mehr Optionen ausschließt als (DM) (siehe [Sch09]).*

□

**Satz 4.1.2** *Sei $f_{DM}^{opt}$ die Gesamtverspätung der optimalen Lösung von (DM), $f_{DMC}^{opt}$ die der optimalen Lösung von (DMC) und $f_{DMCvar}^{opt}$ die Gesamtverspätungen der optimalen Lösung von (DMCvar). Dann gilt*

$$f_{DM}^{opt} \leq f_{DMCvar}^{opt}$$

*Dagegen ist die Ungleichung*

$$f_{DMCvar}^{opt} \leq f_{DMC}^{opt}$$

*im Allgemeinen nicht unbedingt erfüllt.*

**Beweis:** *$f_{DM}^{opt} \leq f_{DMCvar}^{opt}$ gilt zum einen, weil (DMCvar) zusätzliche Nebenbedingungen (4.6-4.10, 4.14) im Vergleich zu (DM) aufweist, und zum anderen, weil nach Definition der Gewichte $w_a \geq 0$ folgt*

$$\begin{aligned} f_{DM}^{opt} &= \sum_{i \in \mathcal{E}} w_i(x_i - \pi_i) + \sum_{a \in \mathcal{A}_{change}} z_a w_a T \\ &\leq \sum_{i \in \mathcal{E}} w_i(x_i - \pi_i) + \sum_{a \in \mathcal{A}_{change}} z_a w_a T + \sum_{a \in \mathcal{A}_{circ}} w_a = f_{DMCvar}^{opt} \end{aligned}$$

*$f_{DMCvar}^{opt} \leq f_{DMC}^{opt}$ gilt nicht ohne weitere Annahmen, da die Ungleichung*

$$\begin{aligned} f_{DMvar}^{opt} &= \sum_{i \in \mathcal{E}} w_i(x_i - \pi_i) + \sum_{a \in \mathcal{A}_{change}} z_a w_a T + \sum_{a \in \mathcal{A}_{circ}} w_a^{min} \\ &\leq \sum_{i \in \mathcal{E}} w_i(x_i - \pi_i) + \sum_{a \in \mathcal{A}_{change}} z_a w_a T + \sum_{a \in \mathcal{A}_{circ}} w_a = f_{DMCvar}^{opt} \end{aligned}$$

*nur dann direkt erfüllt ist, wenn der Zielfunktionswert des $(DMC)$ mit den identischen Termen des $(DMCvar)$ übereinstimmt und der dritte Term $\sum_{a \in \mathcal{A}_{circ}} w_a$ einen Zuwachs bedeutet.*

□

## 4.2 Der heuristische Ansatz

Eine Möglichkeit, das im vorigen Kapitel definierte Problem $(DMCvar)$ 4.1 anzugehen, stellt ein heuristischer Ansatz dar unter Verwendung des bereits in LinTim

implementierten und integrierten Delay-Management-Verfahrens. Dieses weist nach [Sch09] sowohl im Fall vernachlässigter Umläufe als auch im Fall berücksichtigter fixer Umlaufaktivitäten folgende Zielfunktion auf

$$\min f(x,z,g) = \sum_{i \in \mathcal{E}} w_i(x_i - \pi_i) + \sum_{a \in \mathcal{A}_{change}} z_a w_a T$$

Mit der Ausführung des $(DMC)$ können alle Ankunfts- sowie Abfahrtszeiten im Fahrplan abhängig von den auftretenden Verspätungen bestimmt werden. Auf dieser Grundlage lassen sich dann die Umläufe an den einzelnen Bahnhöfen bewerten. Hier wird dann lokal mit Hilfe der nun zur Verfügung stehenden Informationen und aufbauend auf dem Minimale-Kosten-Fluss-Poblem (3.1.19) separiert optimiert:

$$\sum_{a \in \mathcal{A}_{circ}} w_a$$

Das heißt, dass an jedem Bahnhof ein gewichtsminimales Matching zu finden ist, wofür der vorgestellte Successive-Shortest-Path-Algorithm 3.2.2 zum Einsatz kommt.

Problemtaisch bei diesem heuristischen Ansatz ist allerdings, dass jeder Bahnhof, der auf Grund des gelösten Matchingproblems neu angeordnete Umläufe zugewiesen bekommt, die nachfolgenden Abläufe im Fahrplan beeinflusst. Im besten Fall bedeutet dies, dass eine dem $(DMC)$ nach verspätet beendete Tour früher abgeschlossen werden kann. Für das Matchingproblem und die Bewertungen der Umläufe an nachfolgenden Bahnhöfen hätte dies somit eine Veränderung eben jener Bewertungen zur Folge, die nach dem $(DMC)$ jedoch noch nicht bekannt waren. Entsprechend wird bei Ausführung der Matchings ohne Beachtung der Folgen anderer Matchings nicht mehr gewährleistet, dass an jedem Bahnhof im Hinblick auf den Gesamtablauf des Fahrplans die optimale Umlaufplanung erhält.

## 4.3 Der implementierte Algorithmus

### 4.3.1 Beschreibung

Im Folgenden werden die Ideen der Implementation des Successive-Shortest-Path-Algorithms und die Umsetzung der einzelnen Schritte kurz erläutert.

Grundlage aller Berechnungen stellt der in Kapitel 4.1 konstruierte Graph $G = (V, E)$ mit $n = |V|,\ m = |E|$ dar. Dieser wird im Programm als $n \times n$-Matrix gespeichert. Die Zeilen repräsentieren die ankommenden, zu matchenden Züge und die

Spalten die geplanten Abfahrtszeiten neuer Touren. Dabei geben zwei gleichgroße Matrizen einmal über die Kosten und einmal über die Kapazitäten der Kanten des Residualgraphen Auskunft. Dies wird dadurch realisiert, dass in der einen Matrix die Kosten der Vorwärtskanten festgehalten werden. Die Kosten der Rückwärtskanten entsprechen dann einfach den negativen Einträgen. Die zweite Matrix spiegelt die Kapazitäten einer jeden Kante in $G_f$ wider, indem der Eintrag 1 für eine freie Einheit auf der Vorwärtskante steht und der Eintrag $-1$ wiederum für eine freie Einheit in Richtung der Rückwärtskante. Die jeweils dazugehörige entgegengesetzte Kante ist auf Grund der Wahl der Kapazität von 1 einer jeden Kante von $G$ im Residualgraphen $G_f$ nicht enthalten.

Die weiterhin benötigten Werte von b-Fluss, Level sowie Vorgängerknoten werden als Arrays umgesetzt, wobei die ersten $n$ Einträge die Quellen und die restlichen Einträge die Senken repräsentieren.

Nach der Initialisierung kommt nun der eigentliche Algorithmus. Dieser beginnt damit, dass die erste Quelle genommen und ein Weg von ihr zu einer Senke gesucht wird. An dieser Stelle kommt der Moore-Bellman-Ford-Algorithm zum Einsatz.

Um die kürzesten Wege von der gewählten Quelle zu den Senken zu bestimmen, nutzt der Programmcode die Kapazitätenmatrix aus. Hierbei muss stets zwischen Vorwärtskanten und Rückwärtskanten unterschieden werden, indem man bei den Rechnungen die richtigen Vorzeichen der obigen Definition entsprechend verwendet.

Als Zielknoten für die weiteren Berechnungen wählt das Programm die Senke mit der kürzesten Distanz zur gewählten Quelle. Es könnte aber auch jeder andere Knoten mit einem b-Flusswert von $-1$ genutzt werden.

Letztlich nimmt das Programm die nötigen Änderungen vor, um die Auswirkungen des nun zusätzlich transportierten Flusses einzufügen. Dafür werden die Vorzeichen der Einträge dem verwendeten Weg entsprechend in der Kapazitätenmatrix gewechelt. Außerdem werden Quelle sowie Senke im Array des b-Flusses mit 0 markiert.

Die $-1$-Einträge in der Kapazitätenmatrix symbolisieren das gesuchte Matching von Zügen und Abfahrtszeiten.

### 4.3.2 Ein Beispiel

Die Funktionsweise des Programms soll nun noch an Hand des Beispiels 1.1 demonstriert werden. Die Zeiten in den Kostenmatrizen sind hier der Übersichtlichkeit halber in Minuten gewählt, wobei stets eine Wartezeit zwischen Ankunft und Abfahrt von 5 Minuten einzuhalten ist.

An Bahnhof I sieht die Kostenmatrix wie folgt aus

$$CostMatrix = \begin{pmatrix} 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 36 & 23 & 7 & 0 \\ 17 & 6 & 0 & 0 \end{pmatrix}$$

Nach Durchlaufen des Programms ergibt sich letztlich eine Zuordnung der Umläufe nach

$$CapMatrix = \begin{pmatrix} 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \end{pmatrix}$$

Unabhängig von den bereits neu gemachtchten Umlaufkanten an Bahnhof I wird als nächstes noch die Optimierung an Bahnhof II vorgenommen. Diese liefert

$$CostMatrix = \begin{pmatrix} 28 & 12 \\ 0 & 0 \end{pmatrix}$$

$$CapMatrix = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

Die Optimalität des bereits vorgeschlagenen, an die Verspätungen angepassten Fahrplans (1.1) ist damit verifiziert.

## 4.4 Die Einbindung in LinTim

Auf Grundlage der bereits in der Vorarbeit [Nah11] verwendeten Linienpläne für die Hochgeschwindigkeits-Intercity-Schienennetze Bahn-gross sowie Bahn-klein sind auf einem aperiodischen Plan, der noch durch Ausrollen erzeugt werden muss, in einem begrenzten Zeitfenster Verspätungen zu erzeugen und auszuwerten. Jeder vorhandene Datensatz wird hierzu mit mehreren Verspätungsszenarien belegt und auf drei verschiedene Arten evaluiert.

Das einzelne Verspätungsszenario kommt zuächst ohne Integration der Umlaufkanten auf dem zugrundeliegenden Fahrplan zum Einsatz. Danach findet es beim gleichen Betriebsablauf mit eingebundenen fixen Umläufen Anwendung. Im Anschluss an diesen Durchlauf ist mit Hilfe des Delay-Managements die Datenbasis für das Matchingproblem an den einzelnen Bahnhöfen gelegt. Es schließt sich also

die dritte Evaluation des Fahrplans mit den zuvor erzeugten Verspätungen unter Anpassung der vorhandenen Umläufe an eben diese neuen Bedingungen im Betriebsablauf an.

Mit Hilfe des Successive-Shortest-Path-Algorithms (3.2.2) wird ein kostenminimales Matching an jedem Bahnhof gesucht. Dies repräsentiert eine Zuordnung von Zügen und Abfahrszeiten, die möglichst wenig Verspätungen erlaubt. Das neue Matching ist folglich besser oder gleich dem im Falle der fixen Umläufe, die graphentheoretisch ein beliebiges Matching darstellen.

Da die Ausführung des Delay-Managements mit Abstand die meiste Laufzeit in Anspruch nimmt, sollte ihre Häufogkeit so gering wie möglich gehalten werden. Daher werden vor dem Aufruf des Delay-Managements zunächst alle Bahnhöfe einzeln neu gematcht. Das dieses Vorgehen zu einer zumindest nicht schlechteren Lösung führt als die der fixen Umlaufkanten ist leicht einzusehen.

Hierfür betrachte man das Ergebnis eines einzelnen Matchings an einem beliebigen Bahnhof. Man stellt fest, dass sich die tatsächlichen Abfahrtszeiten entweder verbessern oder gleich gebleiben. Das bedeutet aber für die am Bahnhof startenden neuen Touren, dass kein Zug später als im Fall fixer Umlaufkanten in den folgenden Bahnhöfen einschließlich der Endstation eintrifft. Dies gilt ohne Weiteres ausschließlich unter der Prämisse, dass beim Delay-Management keine veränderten Entscheidungen getroffen werden.

Falls also nun ein Zug schlechter bewertet wird, als er nach einem im Fahrplan zeitlich zuvor stattfindenden Matching tatsächlich verdient, fällt das Matching am neuen Bahnhof möglicherweise nicht so gut aus, wie es hätte sein können unter Berücksichtigung der früheren Ankunftszeit. Dennoch wird nachwievor ausschließlich zum Wohle des Fahrplans, zur Verringerung der Fortpflanzung von Verspätungen gematcht.

# 5 Auswertung der Ergebnisse

## 5.1 Die Ausgangsbedingungen

Um fundierte Aussagen über die Effektivität der Integration von Umlaufenkanten und Verspätungsmanagement treffen zu können, stellen insgesamt 165 Fahrpläne, auf denen wiederum jeweils fünf Verspätungsszenarien erzeugt wurden, die Datenbasis der Rechnungen dar. Ausgangspunkt für diese Fahrpläne wiederum sind die Netzwerke Bahn-gross sowie Bahn-klein, deren Größen man der Tabelle 5.1 entnehmen kann.

| Netzwerk | Anzahl Bahnhöfe | Anzahl Verbindungen | Anzahl Instanzen |
|---|---:|---:|---:|
| Bahn-gross | 319 | 452 | 72 |
| Bahn-klein | 250 | 326 | 93 |

Tab. 5.1: Netzwerke

Der Betrachtungszeitraum beschränkt sich auf acht Stunden, von 8 bis 16 Uhr. Somit werden die Auswirkungen von fixen Umlaufkanten bereits bei überschaubarer Gesamtlaufzeit der Rechnungen hinreichend sichtbar.

Außerdem wird an die Touren die Bedingung gestellt, dass sowohl ihr Start-, sowie ihr Endzeitpunkt in diesem Zeitfenster liegen müssen, um im Ablauf berücksichtigt zu werden. Dies sorgt nicht nur für kleinere Instanzen, sondern auch für gerigere Laufzeiten, als wenn alle Fahrten einbezogen werden würden. Gleichzeitig verringert dies jedoch die Anzahl der Umläufe. Wie viele Umlaufkanten und wiederum auch Aktivitäten es insgesamt gibt, ist Tabelle 5.3 zu entnehmen.

In den jeweiligen Zeitabschnitten werden 25 Ereignisse beliebig mit Verspätungen zwischen 3 und 15 Minuten belegt, deren Auswirkungen mit dem Delay-Management einerseits bestimmt und andererseits bestmöglich minimiert werden. Dabei kommt ein Satz Verspätungen auf allen drei Szenarien (ohne, mit fixen und mit variablen Umläufen) zum Einsatz. Ein direkter Vergleich der Ergebnisse liefert somit erste Aussagen über die Auswirkungen von Umlaufkanten bei der Fortpflanzung von Verspätungen.

| Parameter der GlobalConfig | Wert |
|---|---|
| rollout_whole_trips | true |
| delays_count | 25 |
| delays_min_delay | 300 |
| delays_max_delay | 1200 |
| delays_min_time | 28800 |
| delays_max_time | 57600 |
| DM_earliest_time | 28800 |
| DM_latest_time | 57600 |
| DM_method | “best-of-all” |

Tab. 5.2: Parametereinstellungen in der Global-Config.cnf

| Instanz | Gesamtanzahl Aktivitäten | Umlaufkanten |
|---|---|---|
| Bahn-gross game | 42 250,08 | 20,6 |
| Bahn-gross H7 | 92 717,28 | 19,04 |
| Bahn-gross xpress | 46 794,45 | 21,19 |
| Bahn-klein direct | 94 531,5 | 13,2 |
| Bahn-klein game | 32 549,08 | 8,44 |
| Bahn-klein H7 | 74 901,31 | 13,45 |
| Bahn-klein xpress | 40 080,47 | 7,48 |

Tab. 5.3: Gesamtanzahl der Aktivitäten und durchschnittliche Anzahl der Umlaufkanten pro Instanz

## 5.2 Analyse der Verspätungen

Das Ziel dieser Arbeit lag darin, mit Hilfe einer Anpassung der Fahrzeugumlaufplanung an die den Ablauf störenden Verspätungen die Unannehmlichkeiten für die Passagiere best möglich zu minimieren. Dabei war im Vorlauf untersucht worden ([Nah11]), ob die Diskrepanz zwischen den bereits implementierten Optimierungsproblemen ($DM$) und ($DMC$) hinreichend groß ist, so dass begründet anzunehmen war, durch die variablen Umläufe merkliche Verbesserungen des Fahrplans im Vergleich aum Ablauf nach Ausführung des Verspätungsmanagements mit fix integrierten Umlaufkanten ($DMC$) erzielen zu können.

Das Resultat der vorherigen Arbeit [Nah11] bestätigend zeigte sich auch bei den neuen Testdurchläufen, dass die Integration (fixer) Umläufe höhere Verspätungen verursacht im Vergleich zu den Werten bei Vernachlässigung dieser (linker Plot 5.1). Allerdings lässt sich eine noch größere Differenz hinsichtlich der Anzahl verspäteter Trips zwischen dem Fahrplan ohne Umlaufkanten und dem mit variabel eingebundenen feststellen (rechter Plot 5.1). Somit bringt die Anpassung der Umläufe in diesem Fall keine direkte Verbesserung des Fahrplans mit sich.

Diese Entwicklung ist ebenso bei der durchschnittlichen Verspätung umgelegt auf alle Ereignisse zu erkennen (5.4). Hier wird sogar noch deutlicher, dass durch den gewählten heuristischen Ansatz mit Neuordnung der Umläufe an allen Bahnhöfen gleichzeitig nicht automatisch Verspätungen im gesamten Fahrplan entgegengewirkt wird, sondern dass diese im Gegenteil zunehmen können. Bei einigen Instanzen (Bahn-gross H7 und xpress, Bahn-klein direct und xpress) zeigt sich mehr als nur eine Verdoppelung der Verspätungen vom Fahrplan mit fix integrierten Umläufen im Vergleich zum Fahrplan mit variabel eingebundenen Umlaufkanten.

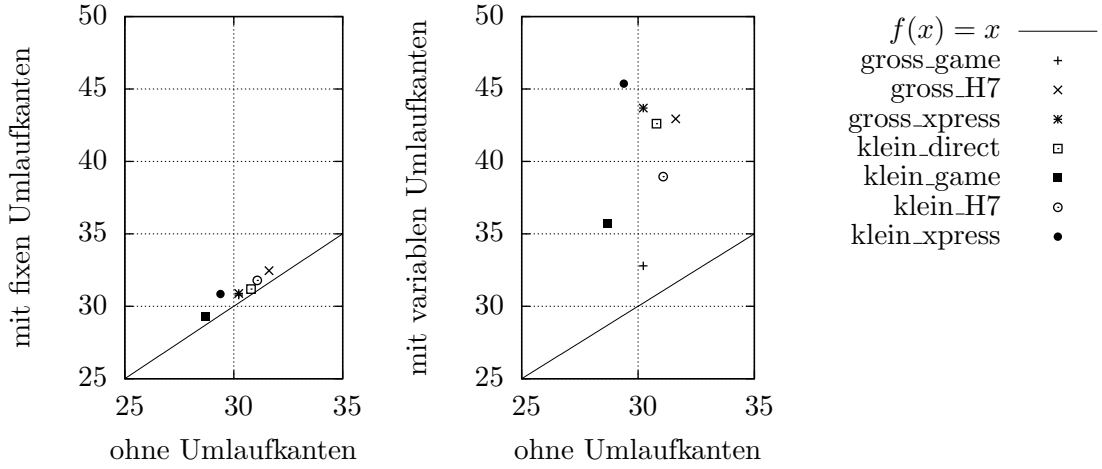



Abb. 5.1: Anzahl verspäteter Trips im Ausrollzeitraum

Insgesamt ist es somit nicht gelungen, auf dem gewählten heuristischen Wege die Verspätungen im Fahrplan zu reduzieren. Diese haben sich im Gegenteil sogar noch deutlich erhöht.

## 5.3 Analyse der Rechenzeit

Zur Auswertung der benötigten Laufzeit für die ausgeführten Rechnungen dienen im Hauptskript "main.sh" entsprechend platzierte Variablen. Mit deren Hilfe ist es nun möglich, die angefallenen Zeiten der verschiedenen Abschnitte miteinander zu vergleichen.

Dabei ist es besonders interessant, den Aufwand für das Delay-Management in den drei unterschiedlichen Fällen (keine Berücksichtigung von Umläufen, Integration fixer Umläufe und Einbindung variabler Umläufe) gegenüberzustellen. Die jeweiligen Zeitspannen umfassen neben der Auswertung des Fahrplans, der Ausführung des Delay-Managements und dessen Auswertung auch die Integration der Umlaufkanten

(a) Durchschnittliche Verspätung auf allen Ereignissen

| Instanz | ohne Umläufe | mit fixen Umläufen | mit variablen Umläufen |
|---|---|---|---|
| Bahn-gross_game | 31.49 | 32.08 | 43.31 |
| Bahn-gross_H7 | 22.08 | 22.56 | 50.76 |
| Bahn-gross_xpress | 35.63 | 36.35 | 93.11 |
| Bahn-klein_direct | 26.67 | 27.20 | 67.08 |
| Bahn-klein_game | 42.61 | 43.42 | 60.91 |
| Bahn-klein_H7 | 29.38 | 29.93 | 51.60 |
| Bahn-klein_xpress | 39.69 | 41.56 | 101.81 |

(b) Differenz zu Verspätungen unter Integration variabler Umläufe

| Instanz | ohne Umläufe | mit fixen Umläufen |
|---|---|---|
| Bahn-gross_game | 11.82 | 11.23 |
| Bahn-gross_H7 | 28.68 | 28.20 |
| Bahn-gross_xpress | 57.48 | 56.76 |
| Bahn-klein_direct | 40.41 | 39.88 |
| Bahn-klein_game | 18.30 | 17.49 |
| Bahn-klein_H7 | 22.22 | 21.67 |
| Bahn-klein_xpress | 62.12 | 60.25 |

Tab. 5.4: Durchschnittliche Verspätung in Sekunden auf allen Ereignissen im Vergleich

sowie die zugehörige Auswertung, sofern erforderlich.

Betrachtet man nun die Resultate der Laufzeitmessung, fällt bei der grafischen Darstellung 5.2 direkt auf, dass der Rechenaufwand für den Abschnitt ohne Integration von Umlaufkanten in den Fahrplan bei allen Instanzen sichtlich geringer ist als in den beiden anderen Fällen. Ein Blick auf die Zahlen 5.5 zeigt des Weiteren, dass diese Rechnung lediglich Dreiviertel der Zeit beansprucht, die bei Integration fixer Umläufe anfällt. Dies verdeutlicht die zusätzlich aufzubringende Rechenleistung, um die Auswirkungen von Fahrzeugumläufen bei Auftreten von Verspätungen im Fahrplan einzubinden.

Weniger deutlich dagegen ist der Unterschied zwischen den Zeiten des Abschnitts mit Einbindung fixer Umläufe und dem mit Integration variabler Kanten, wie man dem Plot 5.2 ansieht. Die Differenz liegt in der Regel unter einer Sekunde (5.5), was zeigt, dass die Anpassung der Umlaufkanten an die Verspätungen keinen nennenswerten Mehraufwand verursacht.

| Instanz | ohne Umläufe | mit fixen Umläufen | mit variablen Umläufen |
|---|---|---|---|
| Bahn-gross game | 54,14 | 73,12 | 73,79 |
| Bahn-gross H7 | 93,20 | 118,00 | 133,00 |
| Bahn-gross xpress | 54,02 | 77,36 | 77,52 |
| Bahn-klein direct | 55,27 | 64,41 | 64,06 |
| Bahn-klein game | 18,96 | 26,30 | 24,74 |
| Bahn-klein H7 | 32,02 | 40,68 | 39,71 |
| Bahn-klein xpress | 19,00 | 27,00 | 26,39 |

Tab. 5.5: Durchschnittlich gemessene Laufzeiten in Sekunden

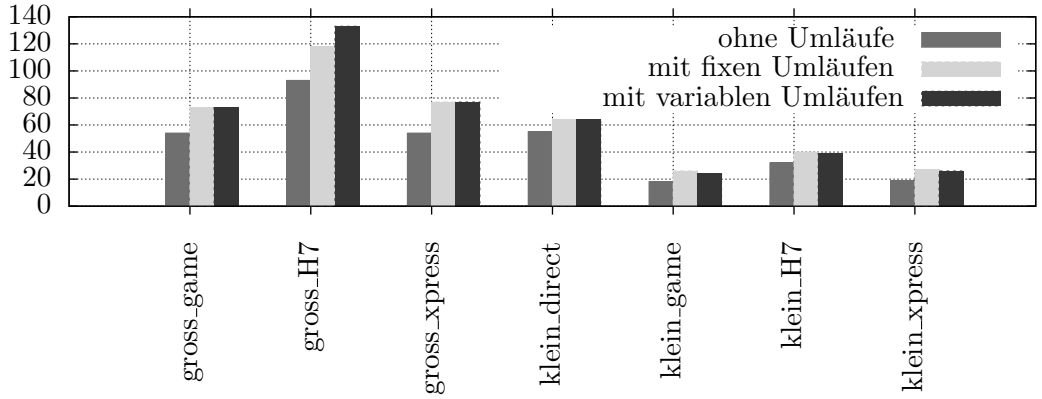



Abb. 5.2: Visualisierte Laufzeiten in Sekunden

# 6 Ausblick

Wie sich gezeigt hat, könnte sich eine Anpassung der Fahrzeugumläufe an die auftretenden Verspätungen lohnenswert gestalten. Allerdings hat der gewählte heuristische Ansatz nicht die erhofften Ergebnisse geliefert. Die Verspätungen haben sich sogar noch erhöht, wie im Kapitel 5.2 bei der Auswertung der Verspätungen zu sehen war. Die Idee, das Verspätungsmanagement und die Matchings an den einzelnen Bahnhöfen iterativ, mehrfach hintereinander auszuführen, stellt sich auf Grund der gemessenen Laufzeiten als recht aufwendig dar, weshalb von einer weiteren Untersuchung abgesehen wurde. Jede Iteration würde einen Zuwachs der aufzuwendenden Rechenleistung um die Zeit bedeuten, die aktuell für ein Delay-Management mit fix integrierten Umläufen benötigt wird (vgl. Kapitel 5.3).

Daher sollte die Lösung des Problems "Integration von Fahrzeugumlaufplanung und Verspätungsmanagement" 4.1 besser direkt implementiert werden, so dass an jedem Bahnhof abhängig von den zuvor stattfindenden Abläufen eine Optimierung hinsichtlich der Fahrzeugumläufe stattfindet.

# Anhang

**copynoncircs.cpp** kopiert die Angaben zu allen Aktivitäten mit Ausnahme der Umläufe aus der Datei "Activities-expanded.giv", die nach der Ausführung des Delay-Managements mit enthaltenen fixen Umläufen entstand.

**main.sh** verwaltet den Ablauf, das Ausführen der Programme und nötigen Dateien. Kopiert die Evaluation ßtatistic.sta"jeder einzelnen Berechnung und speichert diese unter individuellem Namen.

**matrix.h/matrix.cpp** implementiert eine Klasse zum Erstellen und Löschen einer Matrix. Weiterhin können einzelne Einträge beschrieben oder auch direkt abgerufen werden.

**provideinput.pl** extrahiert start- sowie endID, um einen Großteil der für den Matchingalgorithmus relevanten Daten wiederum der "Vehicle_Schedules.giv"zu entnehmen. Diese die aktuellen Umlaufkanten betreffenden Daten werden für jeden Bahnhof separiert in einer eigenen txt-Datei gespeichert. Zusätzlich zu den ursprünglich geplanten Start- und Endzeiten listet das Programm die verspäteten Ankunftszeiten der Züge auf, die in der Datei "Timetable-disposition.tim"zu finden sind.
"provideinput" stellt pro Umlauf folgende Informationen bereit:

*startID; starttime; endID; endtime; delayedstarttime*

**ssp.cpp** müssen beim Aufruf direkt zwei Parameter übergeben werden:

*./ssp dim numb*

Der erste enthält die Anzahl der Umlaufkanten am betrachteten Bahnhof und der andere die Anzahl der Aktivitäten in "Activities-expanded.giv"abzüglich der Umlaufkanten. Im Verlauf des Programm ßsp"werden die benötigten Daten der dem BAhnhof entsprechenden txt-Datei entnommen und alle Instanzen für den Matchingalgorithmus bereitgestellt. Mit Hilfe vom Successive-Shortest-Path-Algorithm und Moore-Bellman-Ford entsteht ein neues Matching. Letztlich realisiert ßsp"die Zusammenstellung dieser neuen Daten die Umläufe betreffend und übergibt sie der Datei "Activities-expanded.giv"in der Form

*actID; -1; "fixed-circulation"; startID; endID; 0; 0*

# Abbildungsverzeichnis



# Tabellenverzeichnis

# Literaturverzeichnis


[Bü10] Christina Büsing. *Graphen- und Netzwerkoptimierung*. Spektrum Akademischer Verlag, 2010.

[Die06] Reinhard Diestel. *Graphentheorie*. Springer-Verlag, 2006.

[Jun08] Dieter Jungnickel. *Graphs, Networks and Algorithms*. Springer-Verlag, 2008.

[KN09] Sven Oliver Krumke and Hartmut Noltemeier. *Grphentheoretische Konzepte und Algorithmen*. Vieweg+Teubner, Wiesbaden, 2009.

[KV06] Bernhard Korte and Jens Vygen. *Combinatorial Optimization, Theory and Algorithms*. Springer-Verlag, 2006.

[Nah11] Rebecca Nahme. *Integration der Fahrzeugumlaufplanung in das Delay-Management*, 2011.

[Sch04] Anita Schöbel. *Optimization Models in Public Transportation*, 2004.

[Sch09] Michael Schachtebeck. *Delay Management in Public Transportation: Capacities, Robustness, and Integration*. PhD thesis, Georg-August-Universität Göttingen, 2009.

[Uff10] Anke Uffmann. *Das Kanalmodell zur Effizienzsteigerung in der Fahrzeugumlaufplanung*. Master's thesis, Georg-August-Universität Göttingen, 2010.

[Wes11] Stephan Westphal. *Netzwerkflüsse*. Vorlesung an der Georg-August-Universität Göttingen, Wintersemester 2010/2011.

# Erklärung

Ich versichere, dass ich die vorliegende schriftliche Hausarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, wurden in jedem Fall unter Angabe der Quellen (einschließlich des World Wide Web und anderer elektronischer Text- und Datensammlungen) und nach den üblichen Regeln wissenschaftlichen Zitierens kenntlich gemacht. Dies gilt auch für beigegebene Zeichnungen, bildliche Darstellungen, Skizzen und dergleichen. Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschungsversuch behandelt werden.

Göttingen, 01.08.2012

Rebecca Nahme