Karlsruhe Institute of Technology

Visualisierung eines Abfahrtsgraphen mittels ILP

Bachelorarbeit
von

Anne Cathérine Jäger

An der Fakultät für Informatik
Institut für Theoretische Informatik

Erstgutachter:	Prof. Dr. D. Wagner
Zweitgutachter:	Prof. Dr. P. Sanders
Betreuende Mitarbeiter:	Dr. Martin Nöllenburg
	Dipl.-Inform. Ben Strasser

Bearbeitungszeit: 1. Juni 2014 – 30. September 2014

Erklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, 30.09.2014

.....

(**Anne Cathérine Jäger**)

Deutsche Zusammenfassung

Diese Arbeit ist im Bereich der Graphenvisualisierung angesiedelt. Anstatt einen Graphen alogorithmisch zu zeichnen, befassen wir uns mit dem Versuch, einen Abfahrtsgraphen als ganzzahliges lineares Programm (ILP) zu modellieren und die berechnete Lösung als Bild darzustellen. Angenommen jemand fährt mit dem Zug und verpasst an einer Umstiegsstation seinen Anschlusszug. Nun möchte der Bahnfahrer wissen, welchen Zug er am Besten wann und wohin nehmen sollte. Der Abfahrtsgraph stellt dies dar und zeigt dem Benutzer für jede weitere Zwischenstation mögliche Verbindungen an. Solch ein Abfahrtsgraph soll nun schön visualisiert werden, um dem Benutzer das Verständnis zu erleichtern. Zu diesem Zweck werden der Graph und sein Design als ILP-Modell formuliert. Dabei soll untersucht werden, ob geeignete Lösungen für dieses Modell schnell berechnet werden können und welche Laufzeitbeschleunigungen Modellveränderungen bewirken. Außerdem wollen wir abwägen, ob dieser Ansatz der klassischen algorithmischen Graphenvisualisierung für den Abfahrtsgraphen vorzuziehen ist.

Abstract

This theses is located in the field of graph visualisation. Instead of drawing a graph algorithmically, we try to model a decision graph as an integer linear program (ILP) and draw the calculated solution as graph picture. Say someone goes by train and misses the next train at an intermediate station. He then wants to know which train to take next to where and when. The decision graph shows possible trains to take for each intermediate station. This graph must now be drawn nicely, so that the user has a better understanding of the possible trains. For this purpose, the graph and its design are modeled as an integer linear program. We then want to examine, if good solutions for this model can be computed efficiently. Furthermore, variations of the model can bring possible speed ups for the calculation. At the end, we want to discuss if this idea is better for decision graph drawing then classical algorithmical graph drawing.

Inhaltsverzeichnis

1. Einleitung

Graphen gehören zu den meist genutzten Datenstrukturen im Bereich der Algorithmik. Ein Graph als Datenstruktur enthält jedoch meist nur abstrakte Informationen, er hat aber vorerst noch keine Gestalt. Um die Informationen für einen Benutzer sichtbar zu machen, ist es vonnöten, den Graphen zu visualisieren. Graphenvisualisierung gehört in den Bereich der Informationsvisualisierung. Sie beschäftigt sich mit der Darstellung von Graphstrukturen anhand von verschiedenen Kriterien. Dabei können diese Kriterien sehr unterschiedlich motiviert sein, beispielsweise kann es für verschiedene Anwendungen wichtiger sein, schnelle Algorithmen zu erhalten, andere wiederum legen mehr Wert auf die Leserlichkeit der Ausgabe. Die traditionelle Graphenvisualisierung beschäftigt sich also mit dem Erstellen von Algorithmen zur Darstellung von Graphen anhand von verschiedenen Kriterien.

Diese Arbeit ist im Bereich der Graphenvisualisierung angesiedelt. Der Eingabegraph, den wir darstellen wollen, ist der *Decision Graph*, oder *Abfahrtsgraph* von Dibbelt et al. [DSW14]. Jeder, der schon einmal mit der Bahn gefahren ist, kennt das folgende Problem: Welchen Zug soll ich als nächstes nehmen, wenn ich einen Anschlusszug verpasse? Um wie viel Uhr fährt dieser Zug und wohin? Muss ich dann am Ankunftsbahnhof nochmals umsteigen? Der Abfahrtsgraph soll darauf eine Antwort geben. Er stellt im Wesentlichen eine Reihenfolge von Zugabfahrten und Zugankünften dar. Ausgehend vom aktuellen Standort, dem Startbahnhof, zeigt er dem Benutzer mehrere Möglichkeiten für weitere Verbindungen an. An jedem Zwischenbahnhof werden wiederum neue Verbindungen für den Fall des Verpassens eines Zuges angezeigt. Verspätungen von Zügen werden bei der Berechnung des Abfahrtsgraphen mittels eines stochastischen Modells berechnet und die Ankunftszeit wird als zu minimierendes Optimierungsproblem formuliert.

Es gibt bereits Ausgabezeichnungen für diese Graphen. Je nach Größe des berechneten Graphen werden diese aber extrem unübersichtlich. Abbildung 1.1 zeigt ein Negativbeispiel. Wie man sehen kann, liegen hier viele Kanten übereinander und man kann nicht erkennen, welche Kanten in welchem Knoten starten und in welchem sie enden. Außerdem haben die Kanten keine vorgegebenen Richtungen. Dies führt dazu, dass die Kreuzungswinkel oft sehr klein sind. Ein weiteres Problem ist, dass Kanten oft sehr lang sind, obwohl es eigentlich einen "kürzeren Weg" durch das Bild gäbe. Ein Ziel dieser Arbeit ist es, eine bessere Darstellung zu finden um den Abfahrtsgraph zu visualisieren. Abbildung 1.2 zeigt eine manuell gezeichnete Beispiellösung für denselben Abfahrtsgraphen. Das ist das Ziel, das wir erreichen möchten.

Im Gegensatz zur direkten algorithmischen Graphenvisualisierung werden wir jedoch einen anderen Ansatz verfolgen. In dieser Arbeit soll kein Algorithmus für die gute Darstel-

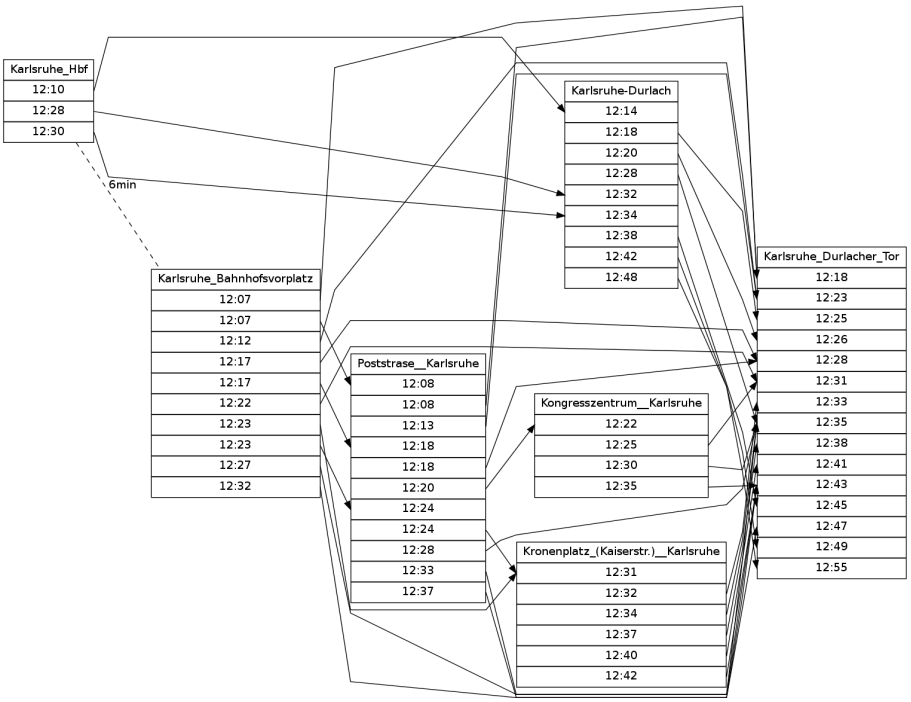


Abbildung 1.1.: Negativbeispiel für die Darstellung eines Abfahrtsgraphen

lung eines Abfahrtsgraphen gefunden werden. Stattdessen formulieren wir eine optimale Graphenzeichnung als Variablen, Nebenbedingungen (engl. *Constraints*) und Zielfunktion eines ganzzahligen linearen Programms. Diese stellen wir dann anhand von bestimmten Designentscheidungen in Bezug zu einander. Dadurch erhalten wir ein Modell für ein lineares Optimierungsproblem, dass von einem Optimierungslöser berechnet werden soll. Zu unterscheiden sind dabei verschiedene Begriffe. Ein lineares Programm (LP) hat nur reellwertige Variablen, ein ganzzahliges lineares Programm (ILP) hat nur ganzzahlige Variablen. Dementsprechend hat ein gemischt ganzzahliges lineares Programm (MLP) sowohl reellwertige als auch ganzzahlige Variablen. Reellwertige lineare Programme lassen sich in polynomialer Zeit berechnen. Ganzzahlig lineare Programme hingegen sind *NP*-schwer.

Die Optimierungstheorie ist ein Bereich der angewandten Mathematik, der sich damit beschäftigt, komplexe Probleme zu lösen. Ein großer Anwendungsbereich der Optimierungstheorie in der Wirtschaft ist Operations Research, jedoch kann sie in beliebigen Fachgebieten eingesetzt werden. Ein Instrument der Optimierungstheorie sind oben genannte Lineare Programme. Wir verknüpfen in dieser Arbeit also die informatische Graphenvisualisierung mit der mathematischen Optimierungstheorie. Um zu erklären, warum wir diesen Ansatz verfolgen, sehen wir uns zunächst die klassische Graphenvisualisierung an.

In der klassischen, algorithmischen Graphenvisualisierung gibt es verschiedene Ansätze, um Graphen darzustellen. Es gibt Kräfte-basierte Verfahren, wie beispielsweise in [Ead84], diese sind physikalisch motiviert. Des Weiteren gibt es Ansätze zur lokalen und globalen Optimierung. Ein Beispiel findet sich in [DH96].

Ein weiterer Ansatz, Graphen darzustellen, sind sogenannte Lagen-Layouts. Dabei werden gerichtete Graphen verwendet, die eine Hierarchie aufweisen müssen. Ein wichtiger Algorithmus für Lagenlayouts wurde von Edes und Sugiyama entwickelt [ES90], [STT81]. Die Idee ist, in einem gegebenen Graphen Kreise zu entfernen, und den Graph so azyklisch zu machen. Dies geschieht durch das Umdrehen von Kanten. Dann werden die Lagen zugeordnet und Kreuzungen werden reduziert. Das Problem bei diesen Lagenalgorithmen ist, dass sie ILP-Optimierungsprobleme enthalten, die *NP*-schwer sind, siehe dazu die Arbeit von Garey

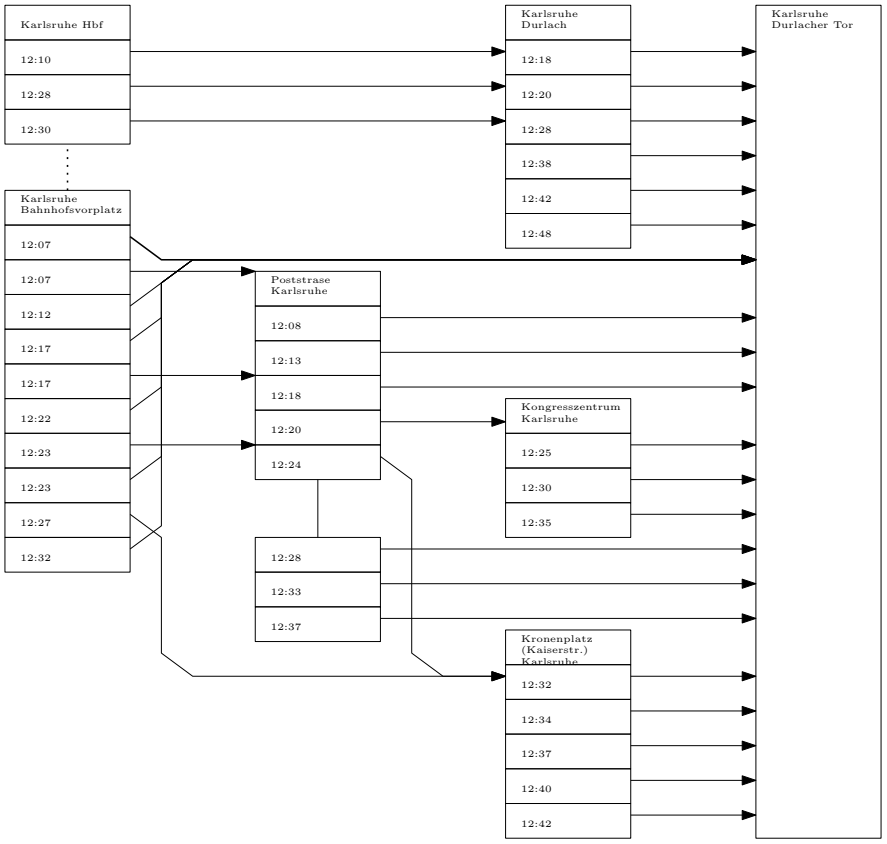


Abbildung 1.2.: manuell erstellte Beispielzeichnung für einen Abfahrtsgraphen

und Johnson [GJ79]. Außerdem sind Lagenlayout-Algorithmen starr in ihren einzelnen Teilschritten. Es ist außerdem nicht offensichtlich klar, wie man die einzelnen Teilschritte auf den Abfahrtsgraphen übertragen kann. Oktolinearität, d.h. vorgegebene Richtungen aus Horizontalen, Vertikalen und Diagonalen können nicht berücksichtigt werden. Außerdem ist eine Modellierung von beliebig vielen Kantenknicken nicht möglich.

Andere Algorithmen versuchen Graphen in einem orthogonalen System darzustellen. Beispiele dazu finden sich in dem Handbuch von Tamassia [Tam13]. Orthogonale Graphen werden in den Werken von Eiglsperger et al. [EFK01] und von Battista et al. [DBETT98] ausführlich besprochen. Orthogonale Darstellungen werden oft genutzt, um vor allem technische Zeichnungen übersichtlich darzustellen. Sie eigenen sich hervorragend als Schemata. In unserem Graph wollen wir uns allerdings nicht auf vier Richtungen für die Zugkanten beschränken. Wir möchten auch Diagonalen verwenden, da wir davon ausgehen, dass das Ausgabebild so übersichtlicher ist und weniger Platz einnimmt.

Graphenzeichnungen wurden bereits von M. Nöllenburg [Nöl05] mittels eines linearen Programms dargestellt. Es handelt sich dabei um die automatisierte Darstellung von Liniennetzen von Untergrundbahnen. Ein entscheidender Unterschied zu der dortigen Modellierung besteht in der Eingabe. Während wir mit Multiknoten arbeiten, sind dort maximal Knoten mit Grad acht erlaubt.

Ein großes Problem aller Algorithmen ist, dass sie in irgendeiner Form beschränkt sind. Manche Algorithmen brauchen planare oder azyklische Graphen als Eingabe oder beides. Dies sind die Abfahrtsgraphen aber selten, meist nur für kleine Graphen. Was viele Algorithmen nicht betrachten ist außerdem die Knotengröße. Viele Algorithmen fordern einen Maximalgrad für die Knoten des Eingabegraphen. Die Abfahrtsgraphen haben einen beliebig hohen Knotengrad, denn ihre Knoten bestehen sind Multiknoten. Das heißt, dass

jede Station viele Unterknoten haben kann, aus denen Kanten ein- oder ausgehen. Außerdem haben die Knoten im Abfahrtsgraph eine Fläche, d.h. eine zweidimensionale Ausdehnung. Dies kann von den klassischen Algorithmen nicht miteinberechnet werden. Was uns im Allgemeinen fehlt ist also Flexibilität.

Wir versuchen, diese für unseren Graphen nötige Flexibilität durch die Verwendung ganzzahliger linearer Programmierung zu bekommen. Anstatt den Graphen und seine Darstellung algorithmisch zu berechnen, stellen wir den Graphen als Variablen und das von uns gewünschte Design als Constraints dar. Ein Vorteil der (ganzzahligen) linearen Programmierung im Allgemeinen ist, dass sie beliebig komplex sein kann. Zwar enthält die Modellierung selbst nur lineare Gleichungen, man kann aber fast jeden Sachverhalt damit darstellen. Außerdem können Mengen von Constraints einfach hinzu gefügt oder entfernt werden. Dies erleichtert das An- und Ausschalten von Varianten im Design. Ein weiterer Vorteil ist, dass der Löser keine Anforderungen an die Eingabe stellt. Wie wir den Graphen als Variablen darstellen, bleibt also uns überlassen. Wir sind darin völlig frei.

Das finale Ziel dieser Arbeit ist es, heraus zu finden, ob dieser Ansatz sinnvoll ist. Welche Vor- und Nachteile hat dieser Ansatz? Lohnt es sich, mehr Flexibilität zu haben? Müssen wir dafür auf etwas anderes verzichten? Ist die Modellierung des Graphen als Lineares Programm effizient genug, um in realen Anwendungen anderen Algorithmen vorgezogen zu werden?

Die Arbeit ist folgendermaßen gegliedert: In Kapitel 2 werden die Grundlagen für die Arbeit gelegt. Es enthält eine Einführung in die lineare Programmierung, sowie eine formalere Beschreibung des Abfahrtsgraphen. Kapitel 3 enthält das Grundmodell für den ILP-Löser, das den Graphen und seine Darstellung modelliert. In Kapitel 4 werden Variationen dieses Modells vorgestellt. Von diesen erhoffen wir uns die Beschleunigung der Berechnungszeit. Ob diese Variationen den erhofften Effekt haben, überprüfen wir in Kapitel 5. Das letzte Kapitel 6 enthält eine kurze Zusammenfassung über das bisher geleistete und gibt Antworten auf die Leitfrage.

2. Grundlagen

2.1. Einführung in die lineare Programmierung

Ein lineares Programm (LP) ist ein Optimierungsproblem, dessen Lösung eine Zielfunktion (engl. *objective function*) minimieren oder maximieren soll. Dabei wird eine endliche Menge an *Variablen* gegeben, deren Werte durch Bedingungen (engl. *Constraints*) beschränkt sind. Bei einem Linearen Programm nehmen alle Variablen reelle Werte an. Das Optimierungsproblem lässt sich mathematisch wie folgt formulieren:

Definition 2.1. *Sei $A \in \mathbb{R}^{m \times n}$ eine $m \times n$ Matrix, $b \in \mathbb{R}^m$ ein m-dimensionaler Vektor und $c \in \mathbb{R}^n$ ein n-dimensionaler Vektor. Sei $x \in \mathbb{R}^n$ ein Vektor aus Variablen. Ein lineares Programm wird formuliert als*

$$\begin{aligned} \min \quad & c^T x \\ \text{bedingt durch} \quad & Ax \leq b \end{aligned}$$

Jedes Minimierungsproblem kann als Maximierungsproblem formuliert werden. Die *Zielfunktion* $c^T x$ beschreibt dabei, welche Variablen aus dem Variablenvektor x mit welchen Koeffizienten aus c minimiert oder maximiert werden sollen. Ist der entprechende Eintrag im Koeffizientenvektor gleich Null, so geht die Variable nicht in die Optimierung mit ein. Die Matrix A stellt dabei die linearen Bedingungen dar, die durch das lineare Programm modelliert werden.

Eine Lösung des Optimierungsproblems heißt *zulässig* (engl. feasible), wenn sie alle linearen Bedingungen erfüllt. Gibt es keine zulässige Lösung für ein lineares Programm, so heißt das lineare Problem *unzulässig* (engl. infeasible), oder auch unlösbar. Eine Lösung des linearen Programms heißt *optimal*, wenn sie das Minimierungs- bzw. Maximierungsproblem erfüllt, d.h. wenn es die kleinst bzw. größt mögliche zulässige Lösung ist.

Ein *Integer Linear Program* (ILP) ist ein ganzzahlig lineares Programm, bei dem alle Variablen nur ganzzahlige Werte annehmen können. Dies ist manchmal nötig, um Dinge wie Stückzahlen, Produktmengen oder Ähnliches modellieren zu können. Ein *Mixed Integer Linear Program* (MIP) ist dementsprechend ein gemischt ganzzahliges lineares Programm, bei dem einige Variablen reelle Werte, andere wiederum nur ganzzahlige Werte annehmen können.

Es gibt eine Menge an Algorithmen, die zur Lösung von linearen Programmen genutzt werden. Diese sollen in dieser Arbeit jedoch nicht besprochen werden. Ein *ILP-Löser* ist ein

Programm, das ganzzahlige lineare Programme als Modelle entgegennehmen und zulässige Lösungen anhand dieser Algorithmen berechnen kann. Reellwertige lineare Programme sind in polynomialer Zeit lösbar. Ändert man dagegen den Wertebereich der Variablen auf ganze Zahlen, so erhält man ein ILP. Diese sind *NP*-schwer.

2.2. Der Abfahrtsgraph

Der Abfahrtsgraph oder auch "Entscheidungs-Graph" ist das Ergebnis einer Arbeit von Dibbelt et al. [DSW14]. Der Abfahrtsgraph soll einem Bahnfahrer anzeigen, welchen Zug er nehmen soll, wenn er einen Anschlusszug verpasst. Er besteht also aus einer Reihe von Zugkanten mit Abfahrtszeiten und Ankunftszeiten an verschiedenen Bahnhöfen. Dabei werden Verspätungen von Zügen mittels einem Wahrscheinlichkeitsmodell formuliert. Die zu erwartende Ankunftszeit wird als Minimierungsproblem aufgefasst.

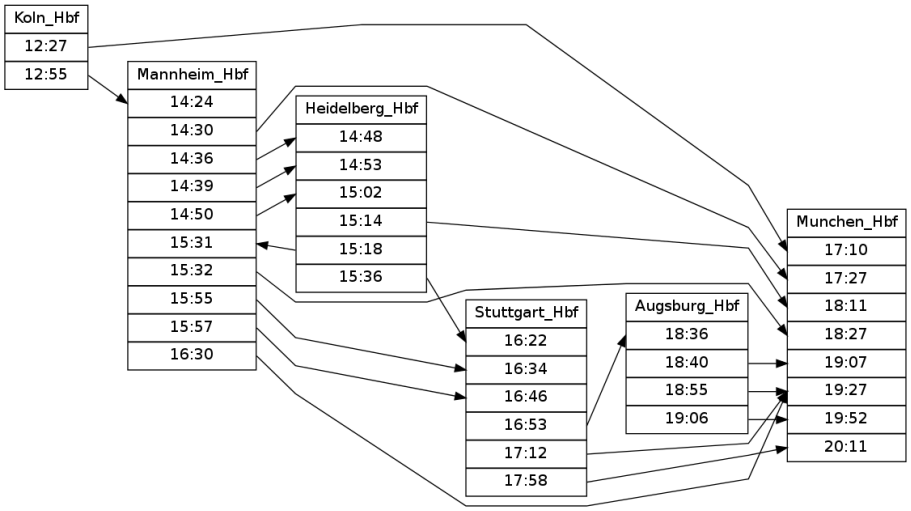


Abbildung 2.1.: Ein Beispiel eines Abfahrtsgraphen von Köln nach München

Der *Abfahrtsgraph* ist ein gerichteter Graph $G = (V, E)$. Er kann sowohl Kreise enthalten, als auch nicht planar sein. Er enthält Informationen über potentielle Ausweichzüge im falle von Verspätungen und soll dem Benutzer Auskunft darüber geben, welchen Zug er nehmen kann, wenn er den gewünschten Zug verpasst. Dabei werden Verspätungen mit einem stochastischen Modell berechnet und die erwartete Ankunftszeit als Minimierungsproblem formuliert.

Die Knotenmenge V besteht aus *Stationen* $v \in V$. Dabei ist die Station v_1 der *Startknoten*, d.h. der Startbahnhof. Der Endbahnhof, oder auch *Stoppknoten* ist die Station v_n, wobei $n = |V|$ gilt. Zwischen dem Start- und dem Stoppknoten liegen weitere Stationen, die mit gerichteten Kanten verbunden sind. Dabei sind Kreise zwischen Stationen möglich, d.h. der Abfahrtsgraph ist nicht notwendiger Weise azyklisch. Jede Station hat Unterknoten, die mit einer Uhrzeit beschriftet sind. Diese Unterknoten sind entweder Ankunftsknoten oder Abfahrtsknoten.

Die Kantenmenge E besteht aus *Zugkante* $e \in E$. Eine *Zugkante* ist eine Kante, die aus dem Abfahrtsknoten einer Station ausgeht und in den Ankunftsknoten einer anderen Station einläuft. Sie wird mit $e_{i_j,u}$ bezeichnet, wobei v_{i_j} ihr Abfahrtsknoten und v_u ihre Ankunftsstation ist. Aus jedem Abfahrtsknoten geht genau eine Zugkante aus, d.h. die Kantenmenge E ist also genauso groß wie die Menge an Abfahrtsknoten und wir nennen $|E| = m$. Jede Zugkante $e_{i_j,u}$ hat eine Abfahrtszeit, ein sog. *Label*, das in ihrem zugehörigen Abfahrtsknoten notiert ist. Sie hat außerdem ein Kantengewicht $g(e) \in [0, 1]$ und einen

Zugtyp. Einige Beispiele für mögliche Zugtypen sind ICE, EC und RB. Auch S-Bahnen und Fußwege sind möglich. Außerdem hat jede Kante ebenfalls eine Ankunftszeit, die im Modell aber nicht berücksichtigt wird.

Ein *Abfahrtsknoten* ist ein Knoten, der mit der Abfahrtszeit eines Zuges beschriftet ist und aus dem genau eine Zugkante ausgeht. Jede Station v_i hat genau max_i Abfahrtsknoten, wobei max_i für jede Station unterschiedlich sein kann. Abfahrtsknoten werden mit v_{i_j} bezeichnet, wobei $1 \leq j \leq max_i$ gilt. Der unterste, d.h. der letzte, Abfahrtsknoten einer Station wird folglich mit $v_{i_{max_i}}$ bezeichnet.

Ein *Ankunftsknoten* ist ein Unterknoten einer Station, in den eine oder mehrere Kanten eingehen. In ihm ist die Ankunftszeit einer Zugkante notiert. In der von Dibbelt et al. vorgestellten Originalausgabe werden alle Abfahrtsknoten eingezeichnet. In dem von uns erstellten ILP-Modell werden Ankunftsknoten jedoch nicht berücksichtigt. Der Stoppknoten hat lediglich Ankunftsknoten, aber keine Abfahrtsknoten.

Der Abfahrtsgraph enthält weitere Informationen wie die Verspätung von Zügen. Diese ist zur reinen Darstellung des Graphen als Bild jedoch nicht nötig und wird daher vernachlässigt. Abbildung 2.2 zeigt eine Beispielzeichnung für einen solchen Abfahrtsgraphen.

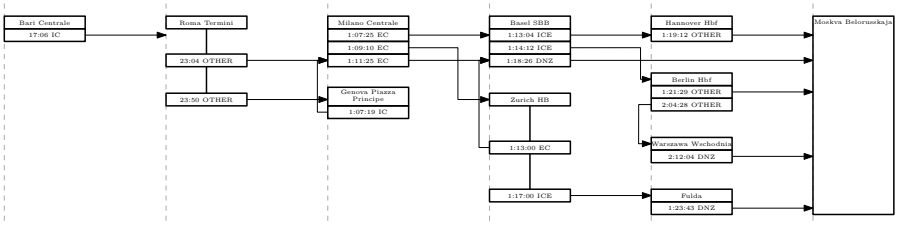


Abbildung 2.2.: Beispielzeichnung für einen Abfahrtsgraphen

2.3. Programm

Das Ergebnis dieser Arbeit ist ein Programm, das einen Abfahrtsgraphen als Textdatei einlesen und daraus ein ILP-Modell erstellen kann. Das Modell wird dann an einen ILP-Löser gereicht, der zulässige Lösungen berechnet. Der von uns verwendete Löser ist Gurobi[1]. Gibt der Löser eine zulässige Lösung für das ILP-Modell aus, so schreibt das Programm die Lösung in eine XML-Datei. Diese kann vom mathematischen Zeicheneditor IPE [2] eingelesen und daraus das gewünschte Bild als Pdf-Datei generiert werden. Abbildung 2.3 zeigt den schematischen Aufbau.

[1] *Gurobi Optimizer*, Version 5.6.;
Offizielle Website: http://www.gurobi.com/

[2] The Ipe extensible drawing editor, Version 7.1.5.;
Offizielle Website: http://ipe7.sourceforge.net/index.html
Download: http://sourceforge.net/projects/ipe7/

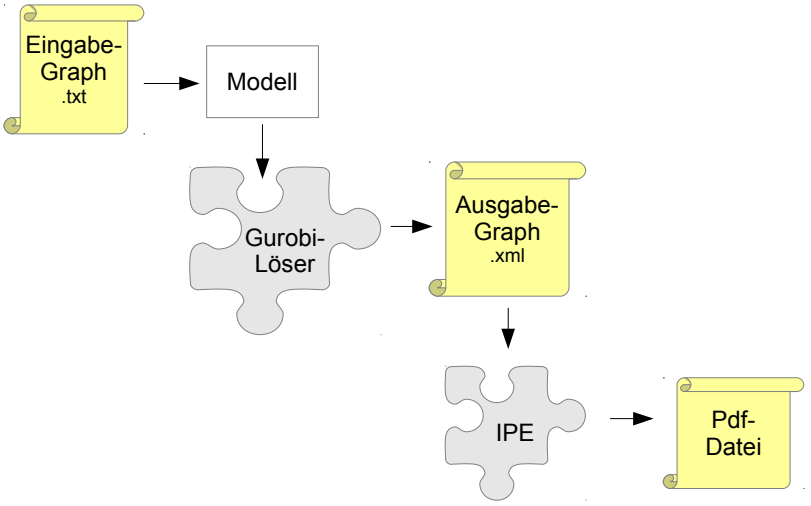


Abbildung 2.3.: Aufbau des Programms

3. Das ILP-Modell

In diesem Kapitel wird das ILP-Grundmodell für das Zeichnen des Abfahrtsgraphen vorgestellt. Die Modellierung bezieht sich dabei auf das Erstellen von Variablen und Constraints als ganzzahliges lineares Programm. Wir formulieren unser Modell als Minimierungsproblem. In Abschnitt 3.1 wird die Zeichenfläche beschrieben. Danach werden in Abschnitt 3.2 Stationen mit Stationsknoten und Abfahrtsknoten modelliert. Anschließend führen wir in Abschnitt 3.3.1 ein neues Koordinatensystem ein, mit dem Diagonalen beschrieben werden können. Danach modellieren wir in den Abschnitten 3.3.2 bis 3.3.4 die Zugkanten mit Dummyknoten, Segmenten, Knicken und Richtungen. Abschnitt 3.4 enthält eine Beschreibung für Sperrflächen, um Kreuzungen von Kanten mit Stationen zu verhindern. Schlussendlich wird in Abschnitt 3.5 die Zielfunktion für die Minimierung beschrieben.

Bevor das Modell implementiert werden kann, muss ein Design gewählt werden. Das Modell hält sich in großen Teilen an die Originaldarstellung des Abfahrtsgraphen aus der Arbeit von Dibbelt et al [DSW14], d.h. Stationen werden vertikal dargestellt. Das bedeutet, dass sich Abfahrtsknoten einer Station vertikal unter dem Stationsknoten befinden. Im Unterschied zur Originaldarstellung gibt es in dem Modell aber keine Ankunftsknoten. Stattdessen dürfen Kanten an der gesamten Station ankommen.

Wir wählen dabei eine Darstellung der Knoten als rechteckige Boxen und der Kanten als Polylinien. Jede Polylinie hat einen Startpunkt $(x_1|y_1)$, einen Endpunkt $(x_2|y_2)$ und dazwischen beliebig viele Dummyknoten. Diese sind auf der Kante allerdings nur als Knick sichtbar. Die Teillstrecken zwischen den einzelnen Dummyknoten sind gerade Strecken ohne Kurven. Alle Punkte im Bild werden mit kartesischen Koordinaten dargestellt und in Points (pts) angegeben. Dies erleichtert uns das Konvertieren der Koordinaten des Modells in Koordinaten für das Zeichenprogramm IPE (siehe Abschnitt 2.3). Alle Koordinaten sind ganzzahlig, außer die Koordinaten der Dummyknoten. Diese sind reellwertig, darauf wird jedoch später noch genauer eingegangen. Alle Konstanten sind ebenfalls ganzzahlig.

Außerdem müssen wir Kriterien festlegen, was eine "gute" Lösung ausmachen soll. Diese sollen dann anhand der Zielfunktion umgesetzt werden. An der Originaldarstellung wurde bemängelt, dass Kanten oft zu lange Wege durch das Bild beschreiten. Wir möchten also, dass Kanten möglichst kurz sind. Außerdem wollen wir unnötig viele Kreuzungen vermeiden, um mehr Ruhe in das Kantenbild zu bekommen. Des Weiteren soll die Anzahl aller Kantenknicke im Bild minimiert werden. Das Ausgabebild sollte möglichst kompakt sein. Stationen sollen nicht weit verteilt sein und man sollte erkennen können, welche Abfahrtsknoten innerhalb einer Station zusammengehören. Im Folgenden werden wir diese Ziele angehen.

3.1. Die Zeichenfläche

Um einen Abfahrtsgraphen als Bild darstellen und diesen später auch tatsächlich zeichnen zu können, definieren wir als erstes eine Zeichenfläche, das sogenannte *Canvas*. Darin sollen alle Stationen und Kanten platziert werden. Der Ursprung befindet sich links unten. Damit wird gefordert, dass alle Koordinaten größer oder gleich Null sein müssen. Die Größe des Canvas beschränkt die möglichen Koordinaten nach oben hin in x- und y-Richtung. Die Konstante Ch steht für die Höhe des Canvas (Canvas Height), während die Konstante Cw für die Weite des Canvas (Canvas Width) steht. Es muss also für alle Koordinaten gelten:

$$\begin{aligned} Ch \geq 0 \quad &\wedge \quad Cw \geq 0 \\ 0 \leq y \leq Ch \quad &\wedge \quad 0 \leq x \leq Cw \end{aligned}$$

Dabei sind alle Koordinaten außer den Koordinaten der Dummyknoten ganzzahlig und positiv.

3.2. Stationen

Eine Station eines Abfahrtsknotens besteht aus genau einem Stationsknoten und beliebig vielen darunter liegenden Abfahrtsknoten. Die Anzahl aller Stationen im Eingabegraph ist $|V| = n$.

3.2.1. Stationsknoten

Stationsknoten werden mit v_i bezeichnet, wobei v_1 der Startknoten ist und v_n der Stoppknoten. Jeder Stationsknoten enthält ein sogenanntes *Label* mit dem Namen der Station. Um eine Station als Box zeichnen zu können, muss sie einen Ankerpunkt, eine Höhe und eine Breite haben. Den Ankerpunkt definieren wir als linkes unteres Eck der Box. Er wird in dieser Arbeit mit einem kleinen rechteckigen Kästchen dargestellt. Dieses ist in den fertigen Zeichnungen jedoch nicht zu sehen. Er hat die Koordinaten

$$(x(v_i)|y(v_i))$$

Die Höhe H und die Breite B von Stationsknoten werden als Konstanten definiert. Beide sind positiv. Für alle Stationsknoten ist die Höhe und die Breite gleich. Nur der Stoppknoten bildet eine Ausnahme: Er soll sich über die gesamte Canvashöhe ziehen, da aus ihm keine Kanten ausgehen und so Platz für die Ankunft aller Kanten geschaffen wird. Seine Höhe wird mit $H(v_n)$ bezeichnet.

Das Ziel ist es, ein möglichst übersichtliches Bild berechnen zu lassen. Deshalb sollen die Stationsknoten auf einem vertikalen Gitter liegen. Wir definieren also eine Gitterweite Gw, die nur bestimmte x-Werte für Stationen zulässt. Damit Stationen nicht ineinander ragen, sollte die Stationsbreite kleiner als die Gitterweite gewählt sein. Außerdem wird eine Variable ϕ_i eingeführt, die anzeigt, auf welchem Gitterstab eine Station liegt. Um einen Abstand des Gitters vom Rand zu bekommen, wird eine Konstante T (Translation) hinzu addiert. Für alle Stationen gilt also:

$$x(v_i) = \phi_i \cdot Gw + T$$

Außerdem soll der Abfahrtsgraph unserer natürlichen (europäischen) Leserichtung entsprechen. Der Startknoten soll deshalb ganz links im Bild, der Stoppknoten ganz rechts im Bild liegen. Alle anderen Stationen sollen dazwischen liegen. Dies beschränkt die x-Koordinaten wie folgt:

$$\begin{aligned} & x(v_1) = T \\ \forall v_i \neq v_n : \quad & x(v_i) \leq x(v_n) \end{aligned}$$

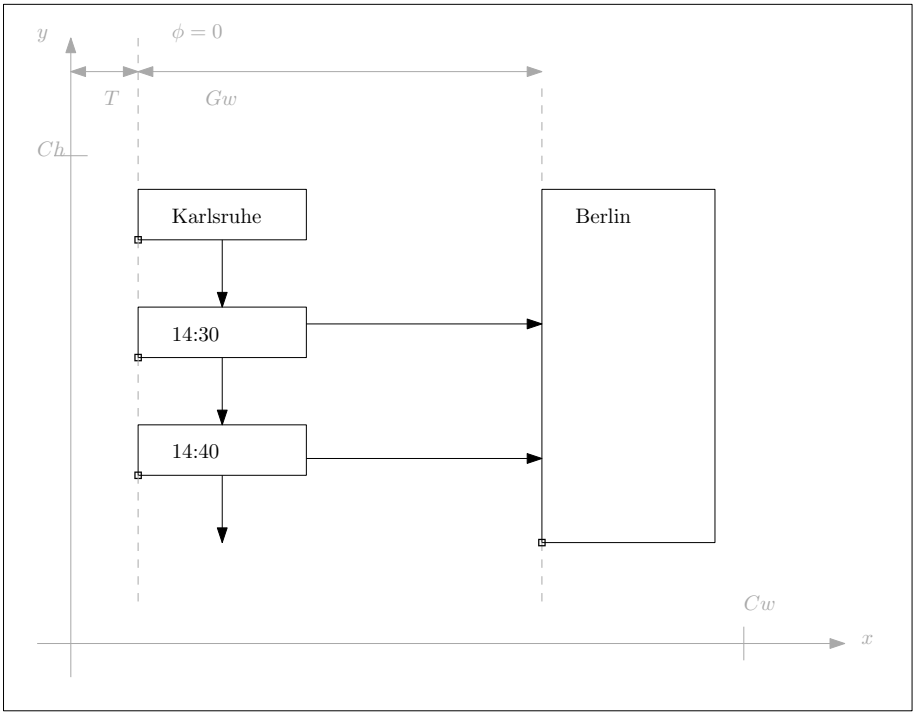


Abbildung 3.1.: Übersicht über die Zeichenfläche mit eingezeichneten Gitterlinien

3.2.2. Abfahrtsknoten

Die Abfahrtsknoten einer Station v_i werden mit v_{i_j} bezeichnet, wobei $1 \leq j \leq max_i$ gilt. Der unterste, d.h. der letzte, Abfahrtsknoten einer Station wird folglich mit $v_{i_{max}}$ bezeichnet. Abfahrtsknoten sollen, genau wie Stationsknoten, als rechteckige Boxen dargestellt werden. Dazu brauchen sie ebenfalls einen Ankerpunkt, eine Höhe und eine Breite. Die Höhe und die Breite sind dieselbe wie die der Stationen. So entsteht ein gleichmäßiges Bild. Der Ankerpunkt eines Abfahrtsknotens v_{i_j} liegt ebenfalls im linken unteren Eck und wird durch seine kartesischen Koordinaten dargestellt.

Um eine Station als Ganzes übersichtlich darzustellen, sollen alle Abfahrtsknoten einer Station vertikal unter dem Stationsknoten liegen. Die x-Werte von Abfahrtsknoten sind deshalb gleich der x-Werte ihrer Stationsknoten. Es gilt also für alle Abfahrtsknoten v_{i_j} einer Station v_i:

$$x(v_{i_j}) = x(v_i)$$

Im Folgenden wird deshalb auch immer der x-Wert des Stationsknotens verwendet. Aus demselben Grund werden im ILP keine Variablen für die x-Werte von Abfahrtsknoten angelegt. Abbildung 3.1 zeigt einen Überblick über den Canvas mit einem Start- und einem Stoppknoten, so wie vertikalen Abfahrtsknoten.

3.2.2.1. Homogener Abstand von Abfahrtsknoten

Um Überschneidungen von Abfahrtsknoten innerhalb einer Station zu vermeiden, müssen ihre Ankerpunkte einen Mindestabstand der Höhe H haben. Sind sie weiter auseinander, so soll die gesamte Stationshöhe möglichst klein, der Abstand zwischen den einzelnen Knoten jedoch möglichst homogen sein. Für jede Station v_i wird eine, innerhalb der Station feste, variable Höhe $hvar_i$ eingeführt. Diese wird dann auf die y-Koordinaten von Abfahrtsknoten addiert. Um tiefer liegende Abfahrtsknoten nach oben bzw. unten verschieben zu können, wird für jeden Knoten v_{i_j} eine Verschiebung nach oben a_{i_j} abgezogen und eine Verschiebung

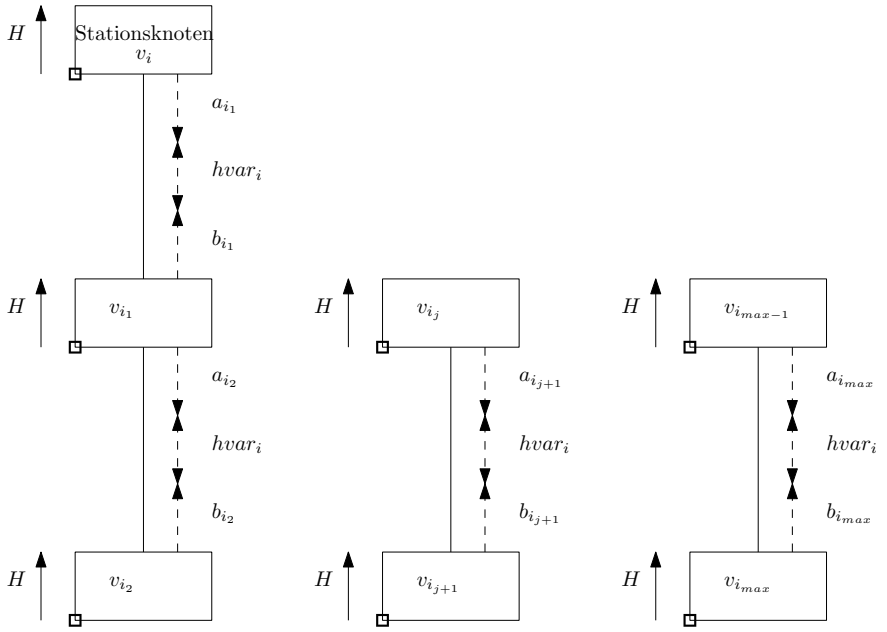


Abbildung 3.2.: Verschiebung der Abfahrstknoten innerhalb einer Station

nach unten b_{i_j} hinzu addiert. Diese sind alle positiv. Außerdem darf die Verschiebung nach oben nicht dazu führen, dass sich Boxen überlappen, sie muss also kleiner als die Zwischenhöhe sein. Abbildung 3.2 illustriert die Modellierung.

Die y-Werte der Stationsknoten berechnen sich dann wie folgt:

$$\begin{aligned}
y(v_i) &= y(v_{i_1}) + H + hvar_i - a_{i_1} + b_{i_1} \\
&\vdots \\
y(v_{i_j}) &= y(v_{i_{j+1}}) + H + hvar_i - a_{i_{j+1}} + b_{i_{j+1}} \\
&\vdots \\
y(v_{i_{max_i-1}}) &= y(v_{i_{max_i}}) + H + hvar_i - a_{i_{max_i}} + b_{i_{max_i}}
\end{aligned}$$

3.2.3. Überschneidungsfreiheit von Stationen

Nun sind die Stationen mit ihren Stationsknoten und Abfahrtsknoten modelliert. Trotzdem ist es möglich, dass zwei Stationen auf dem selben Gitterstab ineinander hinein ragen. Dies möchten wir natürlich verhindern. Es gibt nur vier relevante Fälle, wie zwei Stationen v_i und v_u zueinander auf dem Canvas liegen können, wenn sie sich nicht überschneiden (Abb. 3.3). Für diese vier Fälle fügen wir pro Station-Station-Paar (v_i, v_u) vier binäre Variablen $\psi(i,u)_0, \ldots, \psi(i,u)_3 \in \{0,1\}$ ein. Da nur einer dieser Fälle auftreten kann, modellieren wir die Variablen wie folgt:

$$\psi(i,u)_0 + \psi(i,u)_1 + \psi(i,u)_2 + \psi(i,u)_3 = 1$$

Für den Fall dass beide Stationen auf dem selben Gitterstab liegen, sind ihre x-Werte genau gleich. Die y-Koordinaten müssen jedoch so gewählt werden, dass sich die Stationen nicht überschneiden. Dazu muss der letzte Abfahrtsknoten der oben liegenden Station höher sein, als der Stationsknoten der unteren Station. Außerdem sollen beide Stationen in diesem Fall

einen vertikalen Mindestabstand *Gap* haben. Dieser kann beliebig groß gewählt werden, eine gute Wahl ist beispielweise die doppelte Höhe eines Stationsknotens.

$$\psi(i,u)_0 = 1 \;\Rightarrow\; (y(v_u) + H + Gap < y(v_{i_{max}}) \;\wedge\; x(v_u) = x(v_i))$$

Nehmen wir an, die Stationen liegen auf unterschiedlichen Gitterstäben. Dann unterscheiden sich ihre x-Koordinaten. Über ihre y-Koordinaten muss keine Aussage getroffen werden. Liegt v_i links von v_u, dann ist ihr x-Wert kleiner. Es gilt also:

$$\psi(i,u)_2 = 1 \;\Rightarrow\; (x(v_i) < x(v_u) \;\Leftrightarrow\; x(v_i) + 1 \leq x(v_u))$$

Die beiden anderen Fälle werden analog modelliert. Nun müssen die Gleichungen noch mit ihren Binärvariablen verknüpft werden. Dazu wählen wir eine sehr große Konstante M (eine mögliche Wahl ist $M = Ch + Cw$). Für den ersten Fall wird die Gleichung folgendermaßen verknüpft:

$$\begin{aligned} y(v_u) + H + Gap + 1 - y(v_{i_{max}}) &\leq M \cdot (1 - \psi(i,u)_0) \\ x(v_u) - x(v_i) &\leq M \cdot (1 - \psi(i,u)_0) \\ x(v_i) - x(v_u) &\leq M \cdot (1 - \psi(i,u)_0) \end{aligned}$$

Für die anderen Fälle erfolgt die Verknüpfung analog. Eine Auflistung findet sich in Anhang A.1.

Zu guter Letzt muss dafür gesorgt werden, dass wenn Station v_i links von Station v_u liegt, Station v_u rechts von Station v_i liegt. Für den vertikalen Fall gilt dasselbe. Es muss also gelten:

$$\begin{aligned} \forall u, i : u \neq i : &\quad \psi(u,i)_0 = \psi(i,u)_1 \\ \forall u, i : u \neq i : &\quad \psi(u,i)_2 = \psi(i,u)_3 \end{aligned}$$

3.2.4. Stationskanten

Einzelne Abfahrtsknoten können sehr weit auseinander liegen. Um diese trotzdem zu verbinden, soll zwischen zwei Abfahrtsknoten jeweils eine sogenannte *Stationskante* eingefügt werden. Diese ist nicht Teil des Originalgraphen, sondern eine reine Zeichenmaßnahme. Dem entsprechen sind die folgenden Gleichungen nicht Teil des ILPs.

Eine Stationskante e_{i_j} verläuft senkrecht und genau mittig zwischen den Abfahrtsknoten $v_{i_{j-1}}$ und v_{i_j}. Ihre Koordinaten werden aus den Koordinaten der beiden Abfahrtsknoten berechnet. Diese liegen erst nach der Berechnung durch den ILP-Solver vor. Da die Kante senkrecht verläuft, sind die x-Koordinaten aller Stationskanten innerhalb einer Station gleich. Die Kante soll sich unten und oben an die Boxen der Abfahrtsknoten anschließen. Es gilt:

$$\begin{aligned} y_1(e_{i_j}) &= y(v_{i_{j-1}}) \\ y_2(e_{i_j}) &= y(v_{i_j}) + H \\ x(e_{i_j}) &= x(v_i) + 1/2 * B \end{aligned}$$

3.3. Zugkanten

In diesem Abschnitt wird die Modellierung der Zugkanten beschrieben. Wir führen ein neues Koordinatensystem mit acht Richtungen ein, um dann die Kanten anhand dieser Richtungen zu modellieren. Weiterhin werden Aussagen über das Austreten und Ankommen von Kanten in Boxen getroffen. Die Zugkanten sollen jedoch nicht nur aus einer geraden Linie bestehen. Es soll auch ermöglicht werden, im 45-Grad-Winkel nach oben oder unten abknicken zu können. Dies wird anhand von sogenannten *Dummyknoten* auf der Kante modelliert. Abschließend stellen wir Konzepte für die Längenmodellierung vor.

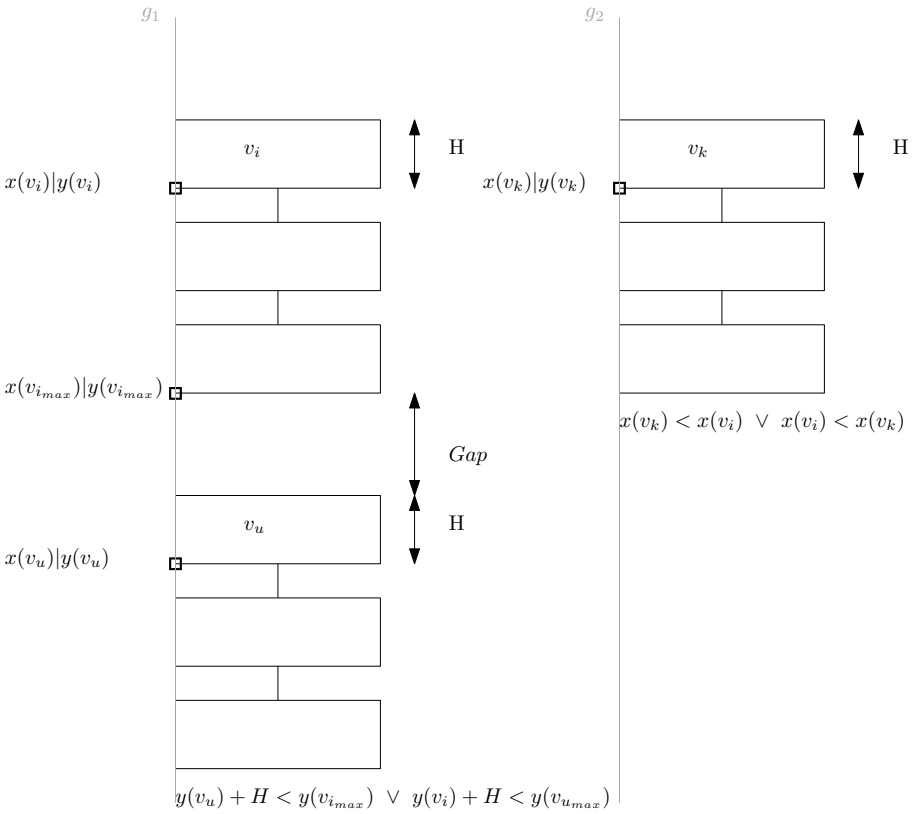


Abbildung 3.3.: Stationen, die sich nicht überschneiden

3.3.1. Oktolinearität

Die Zugkanten in der Ausgabezeichnung sollen sowohl horizontal, als auch vertikal und diagonal im 45-Grad-Winkel verlaufen. Dies ergibt insgesamt acht mögliche Richtungen. Diese Richtungen sollen nun in das ILP eingefügt werden. Ein Beispiel für die Modellierung dieser sogenannten *Oktolinearität* findet sich schon in der Arbeit von Nöllenburg[Nöl05].

Dort wird das bekannte kartesische Koordinatensystem durch zwei weitere Achsen erweitert, die die Diagonalen bilden. Für einen Knoten v mit x- und y-Koordinaten werden diese wie folgt definiert:

$$z_1(v) = x(v) + y(v)$$
$$z_2(v) = x(v) - y(v)$$

Dabei ist zu beachten, dass die Diagonalachsen nur eine für den Leser verständlichere und intuitive Schreibweise für x- und y-Werte darstellen. Gleichungen für das ILP werden weiterhin mit x- und y-Werten berechnet.

Nun können die acht Richtungen definiert werden. Dazu legen wir für die verschiedenen Richtungen eine Nummerierung fest. Diese beginnt bei "rechts", d.h. in positive x-Richtung, gleich Null. Im Uhrzeigersinn drehend wird die Nummerierung ins Negative fortgeführt, gegen den Uhrzeigersinn ins Positive. Eine Illustration befindet sich in Abb. 3.5. Da wir später mit fortlaufenden Nummern rechnen wollen, wird für größere Zahlen modulo gerechnet. Der Modulo-Sprung befindet sich in diesem Beispiel zwischen der Fünf und der Minus-Zwei, dies ist jedoch willkürlich und dient lediglich der Veranschaulichung.

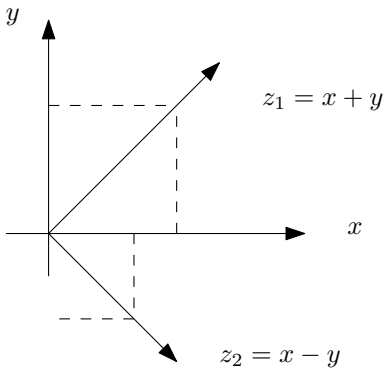


Abbildung 3.4.: Die Diagonalachsen z_1 und z_2

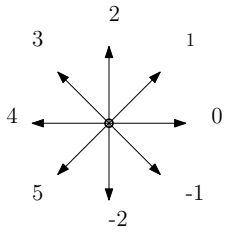


Abbildung 3.5.: Acht mögliche Kantenrichtungen mit dem Modulo-Sprung zwischen fünf und minus-zwei

3.3.2. Start- und Endpunkte von Zugkanten

Zugkanten sollen als Polylinien gezeichnet werden. Eine *Polylinie* besteht aus einem Startpunkt $(x_1|y_1)$, einem Endpunkt $(x_2|y_2)$ und einer festgelegten Anzahl von Punkten dazwischen. Diese Punkte werden durch so genannte *Dummyknoten* modelliert. Diese werden ausführlicher in Abschnitt 3.3.3 diskutiert. Zwischen den einzelnen Punkten verläuft die Polylinie als gerade Linie, d.h. sie hat keine Kurven. Jeden dieser geraden Linienabschnitte bezeichnen wir als *(Kanten-)Segment*.

3.3.2.1. Zugabfahrt

Wir legen fest, dass jede Kante $e_{i_j,u}$ entweder am rechten oder am linken Rand ihres Abfahrtsknotens v_{i_j} austreten kann. Um die Entscheidung über die Ausgangsrichtung zu modellieren, wird für jede Zugkante eine binäre Variable $\alpha(e_{i_j,u}) \in \{0, 1\}$ eingeführt. Im Folgenden verwenden wir die Bezeichnung wegen der Leserlichkeit ohne Kantenzusatz. Wir definieren, dass für $\alpha = 1$ die Kante links austreten soll und für $\alpha = 0$ rechts. Es gilt für ihren Startpunkt:

$$x_1(e_{i_j,u}) = x(v_i) \ \vee \ x_1(e_{i_j,u}) = x(v_i) + B$$

Wir fügen deshalb folgende Gleichungen in das ILP ein:

$$\begin{aligned}
x_1(e_{i_j,u}) \quad & -x(v_i) \leq M \cdot (1 - \alpha) \\
x(v_i) \quad & -x_1(e_{i_j,u}) \leq M \cdot (1 - \alpha) \\
x_1(e_{i_j,u}) \quad & -(x(v_i) + B) \leq M \cdot \alpha \\
x(v_i) + B \quad & -x_1(e_{i_j,u}) \leq M \cdot \alpha
\end{aligned}$$

Als Beschränkungskonstante M ist hier $M = B$ eine gute Wahl.

Außerdem soll die Kante genau mittig aus dem Abfahrtsknoten austreten. Deshalb gilt für ihre y-Koordinate:

$$y_1(e_{i_j,u}) = y(v_{i_j}) + 1/2 \cdot H$$

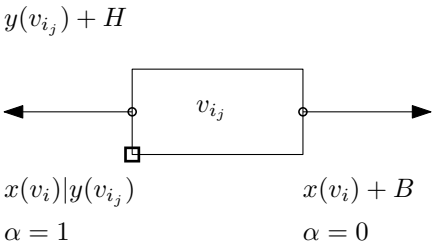


Abbildung 3.6.: Startpunkt einer Zugkante

3.3.2.2. Zugankunft

Für den Endpunkt $(x_2|y_2)$ einer Kante $e_{i_j,u}$ gilt Ähnliches wie für ihren Startpunkt. Bezeichne v_u die Zielstation der Kante. Dann soll die Kante entweder rechts oder links in v_u ankommen können. Dabei soll es egal sein, an welchem Ankunftsknoten sie im Originalgraph angekommen wäre. Wir legen fest, dass die Kante auf der ganzen Höhe der Station, d.h. vom Stationsknoten v_u bis zum letzten Abfahrtsknoten $v_{u_{max}}$ ankommen kann. Für den Endpunkt gilt daher:

$$\begin{aligned} &x_2(e_{i_j,u}) = x(v_u) \ \vee \ x_2(e) = x(v_u) + B \\ &y_2(e_{i_j,u}) \geq y(v_{u_{max}}) \\ &y_2(e_{i_j,u}) \leq y(v_u) + H \end{aligned}$$

Für die Entscheidung über die Ankunftsrichtung wird analog zum Startknoten für jede Kante eine weitere Binärvariable $\beta(e_{i_j,u}) \in \{0, 1\}$ angelegt. Auch hier lassen wir aus Gründen der Lesbarkeit die Kantenbezeichnung ab sofort weg. Es soll gelten, dass die Kante für $\beta = 0$ rechts ankommen soll. Es werden, unter Berücksichtigung der Beschränkungskonstante M, folgende Gleichungen in das ILP eingefügt:

$$\begin{aligned} x_2(e_{i_j,u}) \quad & -x(v_u) \leq M \cdot (1 - \beta) \\ x(v_u) \quad & -x_2(e_{i_j,u}) \leq M \cdot (1 - \beta) \\ x_2(e_{i_j,u}) \quad & -(x(v_u) + B) \leq M \cdot \beta \\ x(v_u) + B \quad & -x_2(e_{i_j,u}) \leq M \cdot \beta \end{aligned}$$

3.3.3. Dummyknoten und Kantensegmente

Im folgenden beziehen sich alle Variablen auf ihre jeweilige Kante $e_{i_j,u}$. Für die Lesbarkeit wird manchmal die abgekürzte Bezeichnung verwendet.

Wir haben nun ein Modell für den Start- und den Endpunkt jeder Zugkante modelliert. Unsere Kanten sollen aber nicht nur aus einer geraden Linie bestehen, sondern auch im 45-Grad-Winkel nach oben oder unten abknicken können. Dazu legen wir eine feste Anzahl an sogenannten *Dummyknoten* $d(e_{i_j,u})$ fest. Die Abschnitte zwischen diesen Dummyknoten heißen *Segmente*. Ein Dummyknoten wird lediglich als Punkt mit $x-, y-, z_1-$ und

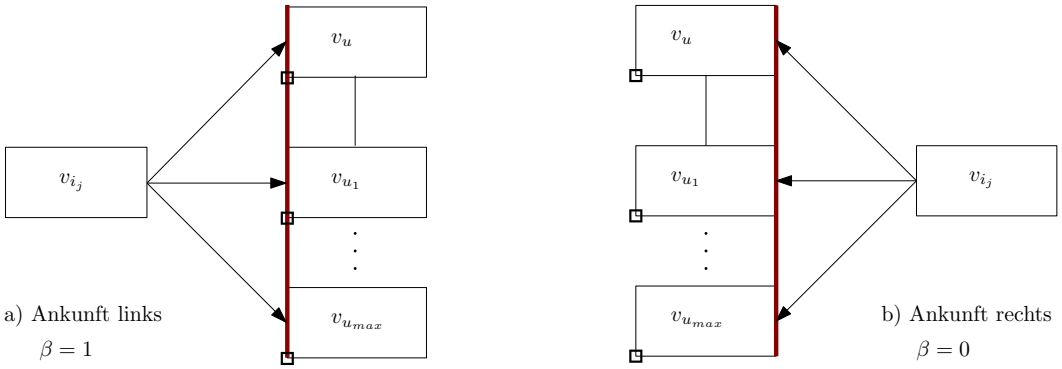


Abbildung 3.7.: Ankunft der Zugkante an der Zielstation

z_2-Koordinaten dargestellt. Wir beginnen beispielsweise mit 4 Dummyknoten pro Kante: Eine Kante $e_{i_j,u} = (v_{i_j}, v_u)$ besteht also aus dem Pfad

$$d_1(e_{i_j,u}), d_2(e_{i_j,u}), d_3(e_{i_j,u}), d_4(e_{i_j,u}), d_5(e_{i_j,u}), d_6(e_{i_j,u})$$

Dabei sind $d_1 = (x_1|y_1)$ und $d_6 = (x_2|y_2)$ der Start- und Endpunkt einer Kante, so wie oben Abschnitt beschrieben. Ein Beispiel mit vier Dummyknoten findet sich in Abb. 3.11.

Damit Dummyknoten möglichst einfach auf dem Canvas verteilt werden können, werden sie als reelle Zahlen implementiert. Alle anderen Koordinaten dagegen sind ganzzahlig. Dies erleichtert auch das platzieren der Kanten genau in der Mitte der Höhe eines Abfahrtsknoten.

Wir halten die Anzahl an Dummyknoten innerhalb einer Probleminstanz variabel und bezeichnen diese Anzahl mit der Konstante Q. Für Q Dummyknoten iterieren wir also über $Q+1$ Segmente und, wenn man den Start- und Endknoten einer Kante mitzählt, über $Q+2$ Knoten. Dabei sind null Dummyknoten theoretisch möglich, dies wird das Modell aber für die meisten Eingaben unlösbar machen.

3.3.3.1. Kantensegmente

Wenden wir uns nun den Segmenten zwischen den Dummyknoten zu. Wir wissen, dass jedes Segment eine von acht möglichen Richtungen haben soll. Dies soll nun modelliert werden.

Jedes Kantensegment $q \in \{1, ..., Q+1\}$ auf der Kante $e_{i_j,u}$ bekommt ein Integer-Offset $r_q(e_{i_j,u}) \in \{-1, 0, 1\}$ für seine *relative Richtung*, abhängig von der Vorgängerrichtung. Dabei ist $r_q = 1$ ein Linksknick, $r_q = 0$ gerade und $r_q = -1$ ein Rechtsknick. Summiert man über alle relativen Richtungen einer Kante, so bekommt man ihre relative Gesamtrichtung (kann auch negativ sein!):

$$r_{i_j,u} = \sum_q r_q(e_{i_j,u})$$

Ein Ziel dieser Arbeit ist es, die Kanten möglichst knickfrei zu zeichnen. Da die Variablen r_q ein Indikator für Kantenknicke sind, wollen wir deren Gesamtzahl also minimieren. Für jedes $r_q(e_{i_j,u})$ wird eine Binärvariable $z_q(e_{i_j,u}) \in \{0, 1\}$ hinzugefügt, für die gilt:

$$-z_q(e_{i_j,u}) \leq r_q(e_{i_j,u})$$
$$r_q(e_{i_j,u}) \leq z_q(e_{i_j,u})$$

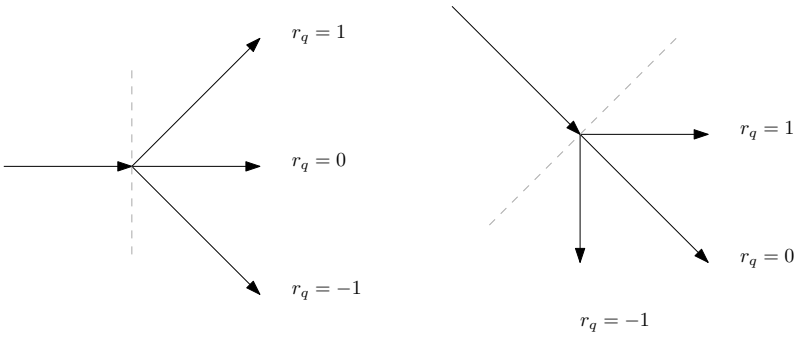


Abbildung 3.8.: relativer Richtungswechsel

Die Minimierung von $z_q(e_{i_j,u})$ berechnet $z_q(e_{i_j,u}) = |r_q(e_{i_j,u})|$, es zeigt also an, ob die Kante an diesem Segmnet knickt, egal ob nach oben oder unten. In der Zielfunktion wird dann die Summe aller Knickvariablen minimiert, siehe Abschnitt 3.5.

Nun werden den Segmenten die absoluten, durchnummerierten Richtungen zugewiesen. Dazu bekommt jedes Kantensegment q auf der Kante $e_{i_j,u}$ eine Variable für seine *absolute Richtung* $s_q(e_{i_j,u}) \in \mathbb{Z}$. Theoretisch ist ein beliebig großes s_q möglich, der Zahlenbereich wird jedoch beschränkt, um eine schnellere Lösung zu bekommen.

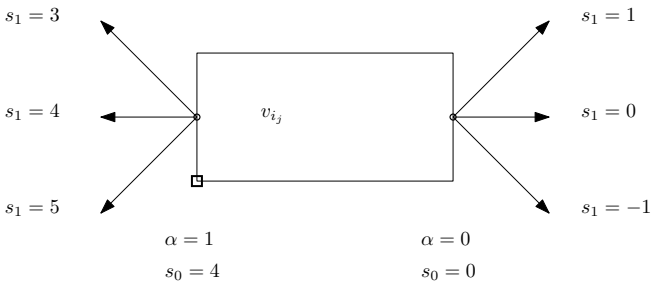


Abbildung 3.9.: Mögliche Richtungen für das erste Segment einer Zugkante

Die Frage stellt sich nun, welches Segment welche Richtung bekommt. Beginnen wir wie im obigen Abschnitt mit den einfachen Fällen. Das Kantensegment s_1 ist das erste Segment auf der Kante, d.h. das Segment zwischen dem Startknoten $(x_1|y_1)$ und dem ersten Dummyknoten. Das Segment s_1 kann abhängig davon, ob die Kante rechts oder links aus dem Knoten austritt, verschiedene Werte annehmen. Wir benutzen dabei die binäre Variable $\alpha(e_{i_j,u})$ von oben (Abschnitt 3.3.2.1). Für die zwei möglichen Fälle gilt:

$$\begin{aligned} &\text{Kante tritt links aus } (\alpha = 1): && 3 \leq s_1 \ \wedge \ s_1 \leq 5 \\ &\text{Kante tritt rechts aus } (\alpha = 0): && -1 \leq s_1 \ \wedge \ s_1 \leq 1 \end{aligned}$$

Wir fügen also folgende Ungleichungen in das ILP mit einer Beschränkungskonstante M ein:

$$\begin{aligned} 3 - s_1 &\leq M \cdot (1 - \alpha) \\ s_1 - 5 &\leq M \cdot (1 - \alpha) \\ s_1 - 1 &\leq M \cdot \alpha \\ -1 - s_1 &\leq M \cdot \alpha \end{aligned}$$

Der zweite einfache Fall ist das letzte Kantensegment s_l, wobei $l = Q + 1$ gilt. Es wird vom letzten Dummyknoten und dem Endpunkt $(x_2|y_2)$ der Kante begrenzt. Das letzte Kantensegment kann ebenfalls rechts oder links an seine Ankunftsstation anschließen. Auch hierfür gibt es schon obige Binärvariablen $\beta(e_{i_j,u})$ (Abschnitt 3.3.2.2). Wir benutzen diese analog zu s_1 um die Fälle zu unterscheiden:

$$\begin{aligned} &\text{Kante tritt links ein } (\beta = 1): && s_l \geq -1 \ \wedge \ s_l \leq 1 \\ &\text{Kante tritt rechts ein } (\beta = 0): && s_l \geq 3 \ \wedge \ s_l \leq 5 \end{aligned}$$

Wir fügen also folgende Ungleichungen in das ILP ein:

$$\begin{aligned} s_l - 1 &\leq M \cdot (1 - \beta) \\ -1 - s_l &\leq M \cdot (1 - \beta) \\ 3 - s_l &\leq M \cdot \beta \\ s_l - 5 &\leq M \cdot \beta \end{aligned}$$

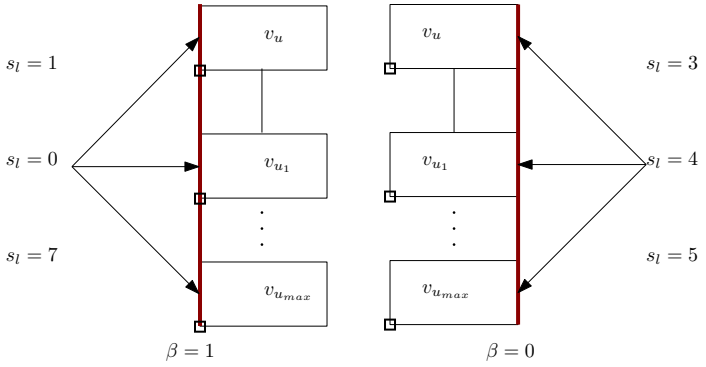


Abbildung 3.10.: Mögliche Richtungen für das letzte Kantensegment s_l

Nun müssen wir noch festlegen, welche Richtungen die Zwischensegmente abhängig von ihren Vorgängersegmenten haben können. Wir möchten erreichen, dass die Kanten nur im 45-Grad-Winkel knicken können. Dazu haben wir bereits Variablen für die relative Richtung angelegt. Nun müssen wir diese relativen Richtungen mit den absoluten Segmentrichtungen verknüpfen. Eine Knick um 45° nach links erzeugt eine Richtungsänderung um plus eins, dementsprechend ist auch die relative Richtungsvariable gewählt. Für die anderen Knickmöglichkeiten gilt dies analog. Die absolute Richtung eines Segments ergibt sich also durch Addition der Vorgängerrichtung mit seiner eigenen relativen Richtungsänderung:

$$s_{q+1} = s_q + r_{q+1}$$

Ein Beispiel mit eingezeichneten Richtungen findet sich in Abb. 3.11.

Wie wir bereits festgelegt haben, kann das Startsegment verschiedene Richtungen annehmen. Doch was ist seine relative Richtung? Um die relative Richtung des Startsegments nicht auch festlegen zu müssen, fügen wir für jede Kante ein imaginäres nulltes Segment ein, das sich vor dem Startsegment befindet und nennen es s_0. Dieses soll per Definition immer horizontal sein. Verlässt die Kante ihren Abfahrtsknoten links, so muss s_0 gezwungen den Wert vier annehmen. Verlässt die Kante den Abfahrtsknoten rechts, so muss s_0 gleich null sein. Da wir für die Ausgangsrichtung bereits eine Variable α angelegt haben, verknüpfen wir diese wie folgt mit dem imaginären ersten Segment:

$$s_0 = \alpha \cdot 4$$

Des Weiteren muss das letzte Segment s_l natürlich ebenfalls aus seinen Vorgängerrichtungen und deren relativen Richtungswechseln hervorgehen. Setzt man dies in obige Gleichungen ein und iteriert bis zum letzten Segment, so kann man dies darstellen als

$$s_l(e_{i_j,u}) = \sum_q r_q(e_{i_j,u}) + s_0(e_{i_j,u})$$

Im Übrigen ist die Modellierung asymmetrisch, da wir keine Gleichung der Art

$$s_{l+1} = \beta \cdot 4$$

in das ILP einfügen. Dies brauchen wir jedoch gar nicht, da sich der gewünschte Effekt aus der bisherigen Modellierung ergibt.

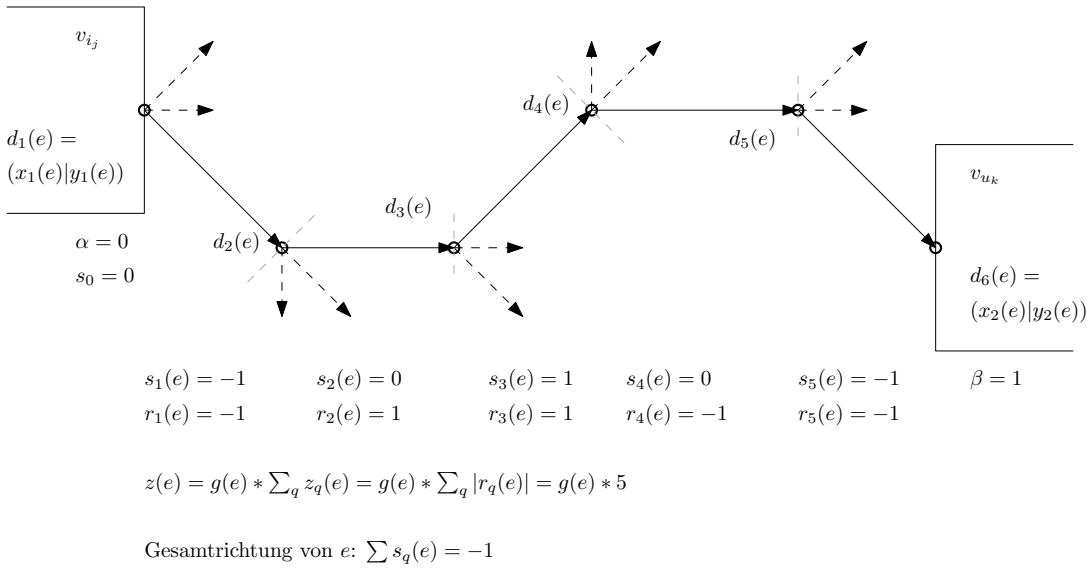


Abbildung 3.11.: Ein Beispiel für den Kantenverlauf einer Zugkante zwischen zwei Abfahrtsknoten.

3.3.3.2. Koordinatenbestimmung anhand von Richtungen

Nun hat jedes Kantensegment seine festgelegte Richtung. Wir wissen, in welche Richtung eine Kante knickt, wieviele Knicke eine Kante insgesamt hat und zu an welcher Seite sie einen Knoten verlässt bzw. ankommt.

Schlussendlich fehlt jedoch der essentielle Kern unseres Zeichenproblems: Welche Koordinaten haben die Dummyknoten? Schließlich soll der Graph am Ende als Boxen und Linien mit Koordinaten gezeichnet werden. Und wie hängen die Dummyknoten von den absoluten Richtungen ihrer Segmente ab?

Wir machen uns zu Nutze, dass wir acht genau festgelegte Richtungen haben: Liegt ein Kantensegment horizontal so müssen die y-Koordinaten seiner begrenzenden Dummyknoten offensichtlich gleich sein. Für ein vertikales Kantensegment gilt dasselbe in Bezug auf die x-Koordinaten und für die Diagonalen analog. Listet man alle Gleichungen auf und nummeriert die Richtungen mit Null bis Sieben durch, so bekommt man die Übersicht in Abb. 3.12.

Nun muss diese Koordinatenverschiebung noch mit den absoluten Richtungsvariablen $s_q(e_{i_j,u})$ verknüpft und ins ILP eingefügt werden. Um für ein Kantensegment zu entscheiden, welche der acht möglichen Richtungen es hat und welche Koordinatenveränderung

0	$y(d_q) = y(d_{q+1})$	$x(d_q) < x(d_{q+1})$
1	$z_1(d_q) < z_1(d_{q+1})$ $x(d_q) + y(d_q) < x(d_{q+1}) + y(d_{q+1})$	$z_2(d_q) = z_2(d_{q+1})$ $x(d_q) - y(d_q) = x(d_{q+1}) - y(d_{q+1})$
2	$x(d_q) = x(d_{q+1})$	$y(d_q) < y(d_{q+1})$
3	$z_2(d_q) > z_2(d_{q+1})$ $x(d_q) - y(d_q) > x(d_{q+1}) - y(d_{q+1})$	$z_1(d_q) = z_1(d_{q+1})$ $x(d_q) + y(d_q) = x(d_{q+1}) + y(d_{q+1})$
4	$y(d_q) = y(d_{q+1})$	$x(d_q) > x(d_{q+1})$
5	$z_1(d_q) > z_1(d_{q+1})$ $x(d_q) + y(d_q) > x(d_{q+1}) + y(d_{q+1})$	$z_2(d_q) = z_2(d_{q+1})$ $x(d_q) - y(d_q) = x(d_{q+1}) - y(d_{q+1})$
-2	$x(d_q) = x(d_{q+1})$	$y(d_q) > y(d_{q+1})$
-1	$z_2(d_q) < z_2(d_{q+1})$ $x(d_q) - y(d_q) < x(d_{q+1}) - y(d_{q+1})$	$z_1(d_q) = z_1(d_{q+1})$ $x(d_q) + y(d_q) = x(d_{q+1}) + y(d_{q+1})$

Abbildung 3.12.: Veränderung der Koordinaten in Abhängigkeit der absoluten Richtung für ein Kantensegment (d_q, d_{q+1})

dementsprechend vorgenommen werden muss, werden für jedes Kantensegment q der Kante $e_{i_j,u}$ acht Binärvariablen erstellt. Diese heißen $\gamma_{q,p} \in \{0, 1\}$, wobei p ein Zähler für die acht Richtungen Null bis Sieben ist. Da immer nur eine absolute Richtung belegt sein kann, darf auch immer nur eine dieser Binärvariablen als wahr belegt sein. Es gilt:

$$\sum_{p \in \{0..7\}} \gamma_{q,p} = 1$$

Die absolute Richtungssvariable könnte jedoch einen Wert größer als acht annehmen. Deshalb muss eine Modulo-Rechnung durchgeführt werden. Diese wird mittels einer Variablen $\delta \in \{-1, 0, 1\}$ durchgeführt. Der Zahlenbereich wird durch die enge Wahl von δ ebenfalls beschränkt, was die Berechnung beschleunigen sollte. Insgesamt gilt nun:

$$s_q = \sum_{p \in \{0..7\}} \gamma_{q,p} \cdot p + \delta_q \cdot 8$$

Nun kann für jedes Kantensegment entschieden werden, welche der Koordinatenveränderungen vorgenommen werden muss. Die Koordinatenveränderung zwischen den einzelnen Dummyknoten ist aber noch nicht modelliert. Anhand der acht Binärvariabeln kann die Koordinatenveränderung nun für jeweils zwei Dummyknoten d_q, d_{q+1} eines Kantensegments $q(e_{i_j,u})$ vorgenommen werden. Dabei ist zu beachten, dass der Start- und der Endknoten einer Kante ebenfalls ein Segment begrenzen und daher auch mit einbezogen werden müssen. Dazu werden die Binärvariablen mit den entsprechenden Gleichungen aus der Liste 3.12

verknüpft. Dies erfolgt beispielhaft an Richtung Null mittels einer großen Konstante M. Da im ILP-Solver nur "kleiner gleich" statt "streng kleiner" benutzt wird, fügen wir bei jedem "streng kleiner" ein +1 ein:

$$
\begin{aligned}
&x(d_1) = x_1(e)\\
&y(d_1) = y_1(e)\\
&x(d_{Q+2}) = x_2(e)\\
&y(d_{Q+2}) = y_2(e)
\end{aligned}
$$

$$
\begin{aligned}
x(d_q) + 1 - x(d_{q+1}) &\leq M(1 - \gamma_{q,0}) \quad &//x(d_q) < x(d_{q+1})\\
y(d_q) - y(d_{q+1}) &\leq M(1 - \gamma_{q,0}) \quad &//y(d_q) = y(d_{q+1})\\
y(d_{q+1}) - y(d_q) &\leq M(1 - \gamma_{q,0})
\end{aligned}
$$

Eine komplette Auflistung aller Verknüpfungen findet sich in Anhang A.3.

3.3.4. Kantenlängen

Nun ist das Modell so weit, dass Stationen und Kanten gezeichnet werden können. Jedoch gibt es außer der Anzahl an Dummyknoten keine Beschränkung, wie die Kanten im Canvas liegen können. Fälle, in denen Kanten über die gesamte Canvasbreite und wieder zurück laufen, sollen vermieden werden. Dazu soll die Kantenlänge minimiert werden. Wir fügen also für jede Kante eine reelle, positive Längenvariable $l(e_{i_j,u})$ ein.

3.3.4.1. Unendlich-Norm

Nun stellt sich die Frage, mit welcher Formel die Kantenlänge berechnet werden soll. In [Nöl05] wurde als Längenmaß die Unendlich-Norm L^∞ verwendet. Daran ein Beispiel nehmend, modellieren wir die Kantenlänge in der Unendlich-Norm. Sie ist für zwei Punkte u, v definiert als:

$$L^\infty(u, v) = \max(|x(u) - x(v)|, |y(u) - y(v)|)$$

Eine Zugkante besteht als Polylinie aus vielen geraden Teilsegmenten. Deshalb berechnen wir die Gesamtkantenlänge l einer Zugkante aus ihren Teillängen, d.h. den Längen ihrer Segmente. Es muss also über alle Segmente q, begrenzt von den Dummyknoten d_q und d_{q+1}, summiert werden:

$$l(e) = \sum_q \max\left(|x(d_q) - x(d_{q+1})|, |y(d_q) - y(d_{q+1})|\right)$$

Beträge wurden bereits in einem vorherigen Abschnitt (siehe 3.3.3.1) modelliert. Dazu muss für jeden Betrag eine neue Variable eingeführt werden. Da es für jedes Segment sowohl einen x-Betrag als auch einen y-Betrag gibt, legen wir pro Segment zwei neue, reellwertig positive Variablen $l_{x,q}, l_{y,q}$ an und modellieren die Beträge folgendermaßen:

$$
\begin{aligned}
-l_{x,q}(e_{i_j,u}) &\leq x(d_q) - x(d_{q+1})\\
x(d_q) - x(d_{q+1}) &\leq l_{x,q}(e_{i_j,u})\\
-l_{y,q}(e_{i_j,u}) &\leq y(d_q) - y(d_{q+1})\\
y(d_q) - y(d_{q+1}) &\leq l_{y,q}(e_{i_j,u})
\end{aligned}
$$

Nun muss noch das Maximum gebildet werden, das dann der Teillänge entspricht. Für jedes Segment wird eine weitere positive Variable m_q angelegt. Diese soll größer als die Beträge sein. Wir binden sie später in die Minimierung ein und bilden dadurch das Maximum der Beträge:

$$l_{y,q}(e_{i_j,u}) \leq m_q(e_{i_j,u})$$
$$l_{x,q}(e_{i_j,u}) \leq m_q(e_{i_j,u})$$

Nun muss noch eine Gleichung eingefügt werden, die die Gesamtkantenlänge als Summe ihrer Segmentlängen darstellt:

$$l(e_{i_j,u}) = \sum_q m_q(e_{i_j,u})$$

Um die Kantenlängen nun so kurz wie möglich zu machen, muss die Gesamtkantenlänge ebenfalls in die Minimierung mit einbezogen werden. Weiteres dazu findet sich in Abschnitt 3.5.

3.3.4.2. Euklidische Norm

Ein von Anfang an gehegter Verdacht, die Berechnung der Kantenlängen über die Unendlich-Norm bevorzuge das Zeichnen von Diagonalen, bestätigte sich im Laufe der Arbeit. Ein zweiter Ansatz ist deshalb, die Kantenlänge als geometrischen Abstand mittels der Euklidischen Norm zu berechnen. Für zwei Punkte u, v ist die Euklidische Norm definiert als:

$$L^2(u,v) = \sqrt{|x(u) - x(v)|^2 + |y(u) - y(v)|^2}$$

Ein Problem der linearen Programmierung ist, dass mathematische Funktionen wie die Wurzelfunktion nicht trivial implementiert werden können. Für zwei beliebige Punkte im Raum ist es kaum möglich, den Abstand mit der Euklidischen Norm als lineares Programm zu berechnen. Unser Modell bringt den Vorteil, dass Punkte auf einer Kante nicht beliebig sind. Da wir die Kante als Polylinie darstellen, sind zwei Dummyknoten auf der Kante durch eine gerade Linie verbunden. Diese gerade Linie hat die Eigenschaft genau horizontal, vertikal oder diagonal zu verlaufen. Dies bringt den Vorteil, dass ein Teil des Wurzelausdrucks immer wegfällt und der Restterm sich durch eine lineare Gleichung darstellen lässt.

Für horizontale Segmente fällt der y-Term weg, für vertikale Segmente der x-Term. Für diagonale Segmente sind die Beträge gleich groß und die Wurzel löst sich auf. Die Konstante Wurzel von zwei kann als Variablenkoeffizient in die Gleichung eingetragen werden. Nun muss nur darauf geachtet werden, dass die Differenzen positiv sind. Schlüsselt man die einzelnen Fälle für die acht möglichen Richtungen auf, so bekommt man die Liste in Abbildung 3.13.

Für die acht möglichen Segmentrichtungen gibt es für jedes Kantensegment die aus Abschnitt 3.3.3.2 bekannten Binärvariablen $\gamma_{q,p}$. Diese müssen nun mit der entsprechenden Berechnungsvorschrift verknüpft werden, um für jedes Segment seine Segmentlänge m_q zu berechnen. Für den Fall der Richtung Null geht dies wie folgt:

$$\begin{aligned} x(d_{q+1}) - x(d_q) - m_q &\leq M \cdot (1 - \gamma_{q,0}) \\ m_q - (x(d_{q+1}) - x(d_q)) &\leq M \cdot (1 - \gamma_{q,0}) \end{aligned}$$

Alle anderen Fälle werden analog modelliert. Eine genaue Auflistung findet sich in Anhang A.4. Wie bei der Unendlich-Norm werden auch hier die Segmentlängen aufsummiert und die Längenvariable l zur Minimierung hinzugefügt. Ein Hinzufügen der Teilstrecke m_q ist jedoch nicht nötig.

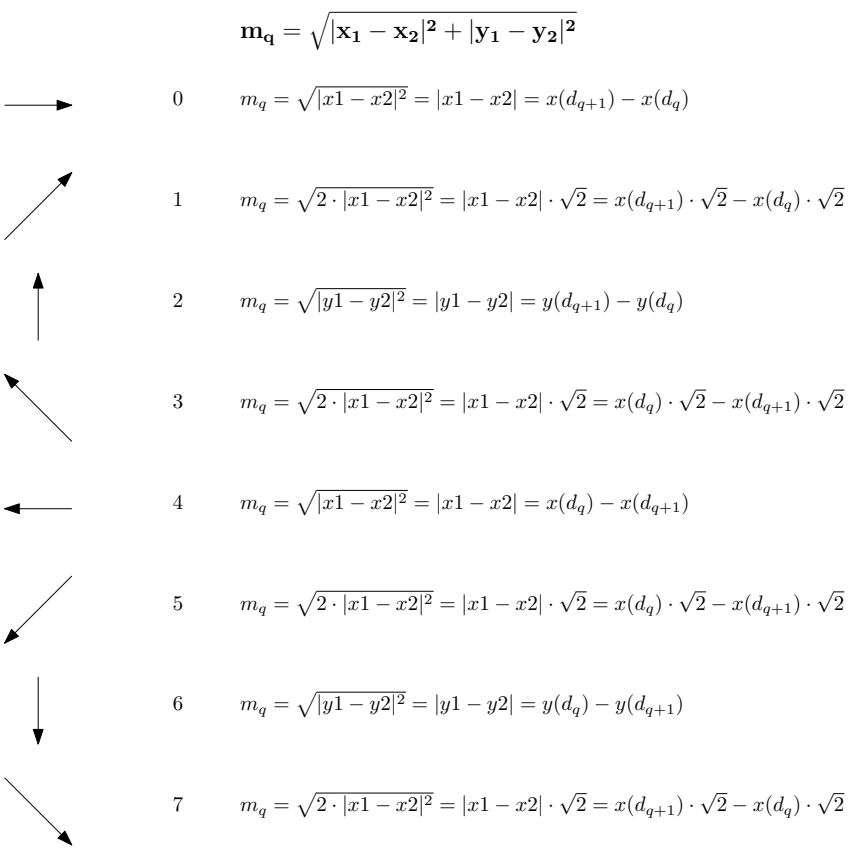


Abbildung 3.13.: Aufschlüsselung der einzelnen Längenberechnungen in Abhängigkeit der Segmentrichtung

3.4. Kreuzungsfreiheit

Nun ist das Modell zum größten Teil fertig. Es enthält Boxen für Stations- und Abfahrtsknoten und die Kanten sind ausmodelliert mit relativen Richtungen, absoluten Richtungen und Kantenknicken. Kanten sind auch möglichst kurz, sie können aber über Boxen oder Schriftzügen liegen. Es muss also noch eine Methode gefunden werden, um bestimmte Flächen, wie z.B. das Innere von Stationen unpassierbar für Kanten zu machen. Im Übrigen wird Kanten-Kanten-Kreuzungsfreiheit nicht modelliert. Da die Lösungsberechnung durch das Hinzufügen von Stations-Kanten-Kreuzungsfreiheit bereits erheblich verlangsamt wurde, gehen wir davon aus, dass die Kreuzungsfreiheit von Kanten mit Kanten zu einer enormen Verschlechterung der Laufzeit führen würde. Deshalb lässt das Modell diese Variante außen vor.

Kanten sollen eine Station als Ganzes nicht passieren dürfen. Dazu stellen wir uns vor, eine Station sei eine rechteckige Box, die die Stationsknoten und alle darunter liegenden Abfahrtsknoten umschließt. Um den Raum zu bestimmen, den diese Stationsbox einnimt, werden um die Station acht Halbebenen $N, NO, O, SO, S, SW, W, NW$ definiert. Ihre Benennung entspricht den Himmelsrichtungen. Die Halbebenen liegen dabei an der Station an. Abbildung 3.14 zeigt die Halbebenen als gestrichelte Linien um eine Station k. Diese beinhaltet den Stationsknoten v_k und die darunter liegenden Abfahrtsknoten $v_{k_1} \dots v_{k_{max}}$.

Nun muss für jede Kante $e_{i_j,u}$ gestestet werden, ob sie die Station k schneidet. Schneidet ein Segment $q = (d_q, d_{q+1})$ der Kante die Box nicht, so müssen seine beiden Punkte in einer oder mehrerer der Halbebenen um die Box herum liegen. Die Richtung des Segments

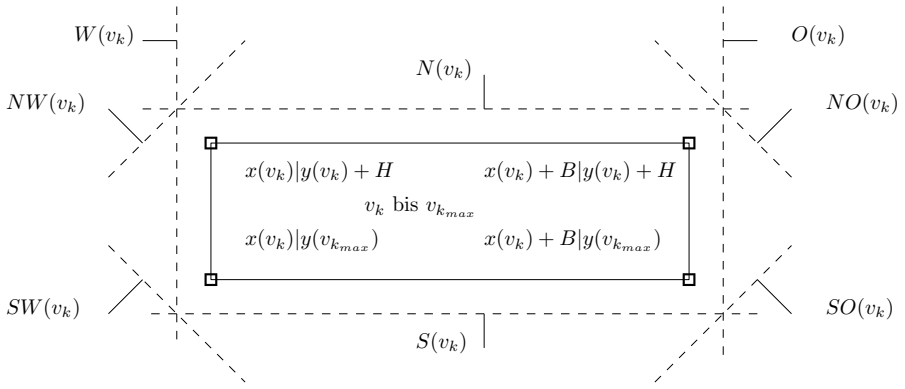


Abbildung 3.14.: Halbebenen um eine Station

ist dabei egal. Ein Dummyknoten $d_q(e_{i_j,u})$ muss die in Liste 3.15 notierten Anforderungen erfüllen, um in den Halbebenen von v_k zu liegen.

Jetzt müssen diese Anforderungen in das ILP übertragen werden. Dafür müssen pro Stations-Kantensegmentpaar v_k und $q(e_{i_j,u}) = (d_q, d_{q+1})$ acht binäre Variablen eingeführt werden. Diese stehen für die acht Halbebenen einer Station und heißen dem entsprechend $N_{i_{j_q},k}, NO_{i_{j_q},k}, O_{i_{j_q},k}, SO_{i_{j_q},k}, S_{i_{j_q},k}, SW_{i_{j_q},k}, W_{i_{j_q},k}, NW_{i_{j_q},k} \in \{0,1\}$.

Dabei müssen einige Sonderfälle beachtet werden. Prinzipiell kann k gleich i sein, denn auch stationseigene Kanten können die Station kreuzen und müssen somit getestet werden. Das erste Segment einer Kante hat aber eine so festgelegte Richtung, dass ein Durchkreuzen der Ausgangsstation unmöglich ist, siehe dazu Abbildung 3.9. Für diesen Fall werden also keine Constraints aufgestellt. Dasselbe gilt für das letzte Segment einer Kante, wie man in Abbildung 3.10 sehen kann. Ist deshalb die zu prüfende Station k gleich der Zielstation u einer Kante, so müssen auch hier für das letzte Segment keine Constraints eingefügt werden.

Außerdem sollen Kanten nicht direkt an einer Station anliegen, sondern einen kleinen Mindestabstand von den Boxen haben. Dazu führen wir einen kleinen Abstand ϵ ein, den wir zu den Gleichungen hinzu rechnen. Nun können die Gleichungen in das ILP eingefügt werden. Für den Fall von $N_{i_{j_q},k}$ geschieht dies folgender Maßen. Die Indizes werden zwecks der Übersichtlichkeit weg gelassen.

$$\begin{aligned} N + NO + O + SO + S + SW + W + NW &\geq 1 \\ y(v_k) + H + \epsilon - y(d_q) &\leq M \cdot (1 - N) \\ y(v_k) + H + \epsilon - y(d_{q+1}) &\leq M \cdot (1 - N) \end{aligned}$$

Eine vollständige Auflistung aller Halbeben-Gleichungen findet sich in Anhang A.5.

3.5. Zielfunktionen und Minimierung

Nachdem alle Variablen angelegt und alle Constraints in das Modell eingefügt wurden, ist es Zeit dem ILP-Löser ein Ziel zu setzen. Das Ausgabebild soll dabei einigen Kriterien entsprechen, die nicht allein durch das Einfügen von Constraints gelöst werden können.

Wie bereits anfangs erwähnt, soll das Ausgabebild möglichst kurze Kanten, möglichst wenige Knicke und möglichst niedrige Stationen haben. Die Entscheidung fällt daher nicht schwer, aus dem gegebenen Problem ein Minimierungsproblem zu machen. Die zu

Abbildung 3.15.: Anforderungen an die Koordinaten von Dummyknoten, wenn sie in den Halbebenen liegen

Halbebene	Anforderungen fuer $d_q(e_{i_j,u})$
$N(v_k)$	$y(d_q) > y(v_k) + H$
	$y(d_{q+1}) > y(v_k) + H$
$NO(v_k)$	$z_1(d_q) > z_1(v_k + H + B) = (x(v_k) + B) + (y(v_k) + H)$
	$z_1(d_{q+1}) > z_1(v_k + H + B) = (x(v_k) + B) + (y(v_k) + H)$
$O(v_k)$	$x(d_q) > x(v_k) + B$
	$x(d_{q+1}) > x(v_k) + B$
$SO(v_k)$	$z_2(d_q) > z_2(v_k + B) = (x(v_k) + B) - y(v_{k_{max}})$
	$z_2(d_{q+1}) > z_2(v_k + B) = (x(v_k) + B) - y(v_{k_{max}})$
$S(v_k)$	$y(d_q) < y(v_{k_{max}})$
	$y(d_{q+1}) < y(v_{k_{max}})$
$SW(v_k)$	$z_1(d_q) < z_1(v_{k_{max}}) = x(v_k) + y(v_{k_{max}})$
	$z_1(d_{q+1}) < z_1(v_{k_{max}}) = x(v_k) + y(v_{k_{max}})$
$W(v_k)$	$x(d_q) < x(v_k)$
	$x(d_{q+1}) < x(v_k)$
$NW(v_k)$	$z_2(d_q) < z_2(v_k + H) = x(v_k) - (y(v_k) + H)$
	$z_2(d_{q+1}) < z_2(v_k + H) = x(v_k) - (y(v_k) + H)$

minimierende Funktion muss dabei aus einem linearen Term bestehen. Dazu werden die einzelnen Teilprobleme linear formuliert und anschließend addiert.

$$\min \quad C_{Stationen} + C_{Knicke} + C_{Kanten}$$

Optional können die einzelnen Teilsummen gewichtet werden. Dies tun wir jedoch nicht, da wir uns auf das Basis-Modell konzentrieren. Die richtige Gewichtung der Minimierungsterme ist ein experimentelles Thema und sprengt den Rahmen dieser Arbeit.

Im Folgenden werden die Teilprobleme als lineare Funkionen unter Verwendung der bisher aufgestellten Variablen modelliert.

3.5.1. Stationshöhe

Stationen sollen überall auf dem Canvas liegen dürfen. Eine Ausnahme bilden der Start- und der Stoppknoten, deren Position auf den linkesten bzw. rechtesten Gitterstab festgelegt ist. Die horizontale Position der Zwischenstationen ist durch das Gitter teilweise vorgegeben, ihre vertikale Position jedoch nicht. Da Zugkanten nicht zwischen einzelnen Abfahrtsknoten hindurch laufen sollen, sondern um Stationen als Ganzes verlaufen, sollen die Stationen möglichst niedrig sein. Das heißt, dass der Abstand zwischen dem Stationsknoten v_i und dem untersten Abfahrtsknoten $v_{i_{max}}$ möglichst gering sein soll. Auch hier bilden der Start- und der Stoppknoten eine Ausnahme. Beide dürfen über die gesamte Canvashöhe verlaufen. Da der Stoppknoten außerdem keine Abfahrtsknoten hat, entfällt er in diesem Teilterm.

In Abschnitt 3.2.2 wurden Variablen eingeführt, um die Höhe zwischen Abfahrtsknoten zu modellieren. Zur Erinnerung: die Höhe zwischen einzelnen Abfahrtsknoten beträgt

$$hvar_i - a_{i_{j+1}} + b_{i_{j+1}}$$

Dabei beträgt $hvar_i$ die innerhalb der Station i gleiche Höhe zwischen zwei Knoten. Die Variablen a_{i_j}, b_{i_j} bewirken eine Verschiebung des nächsten Knotens nach oben oder nach unten. Um den homogenen Abstand zwischen Abfahrtsknoten und die Verschiebungen zu minimieren, fügen wir die entsprechenden Variablen zur Minimierung hinzu. Der Startknoten wird weggelassen. Da der Stoppknoten keine Abfahrtsknoten hat, fällt er automatisch weg. Dazu fügen wir in die Zielfunktion ein:

$$\min\ C_{Stationen} = \left(\sum_{i=2}^{n-1} hvar_i + \sum_{i=2;j=1}^{i=n-1;j=max_i} a_{i_j} + \sum_{i=2;j=1}^{i=n-1;j=max_i} b_{i_j} \right)$$

Wie man sieht, bildet dieses Teilproblem aus seiner Definition heraus eine lineare Funktion aus Variablen.

3.5.2. Kantenknicke

Ein weiteres Teilproblem besteht darin, Kanten möglichst gerade verlaufen zu lassen. Dazu soll die Anzahl der Knicke in der Kante minimiert werden. Außerdem sollen wichtige Kanten, das heißt Kanten mit hohem Gewicht, möglichst wenige Knicke haben. Unwichtigere Kanten mit niedrigem Kantengewicht dürfen mehr Knicke haben, diese sollen aber trotzdem so gering wie möglich sein. Das Gewicht einer Kante wird mit $g(e_{i_j,u})$ bezeichnet.

In Abschnitt 3.3.3.1 wurden Variablen eingeführt, die einen Knick anzeigen. Die Variable $z_q(e_{i_j,u})$ gibt an, ob die Zugkante $e_{i_j,u}$ nach dem Segment q einen Knick hat oder nicht. Um hochgewichtige Kanten gerader zu machen, reicht es aus die Anzahl ihrer Knicke mit ihrem Kantengewicht zu multiplizieren. Wir legen also den Wert einer Kante im ILP durch eine neue Variable fest.

$$z(e_{i_j,u}) = g(e_{i_j,u}) \cdot \sum_q z_q(e_{i_j,u}) = \sum_q (g(e_{i_j,u}) \cdot z_q(e_{i_j,u}))$$

Wie man sieht, ergibt sich auch hier eine einfache lineare Funktion. Die Kantengewichte sind Konstanten im Sinne des ILP und können daher als Variablenkoeffizienten miteinberechnet werden. Für den Fall, dass der Eingabegraph keine Kantengewichte hat, sind alle Kanten gleich wichtig. Um die Knickminimierung trotzdem durchführen zu können, werden die Kantengewichte deshalb mit eins initialisiert. Nun müssen die Gesamtknicke nur noch zur Zielfunktion hinzugefügt werden:

$$\min\ C_{Knicke} = \sum_{e_{i_j,u}} z(e_{i_j,u})$$

3.5.3. Kantenlänge

Das letzte Teilproblem besteht darin, die Kanten möglichst kurz zu machen. In Abschnitt 3.3.4 wurden Variablen $l(e_{i_j,u})$ für die Gesamtlänge einer Kante eingeführt. Bei Kantenlängen soll keine Priorisierung nach dem Kantengewicht vorgenommen werden. Alle Kanten sollen möglichst kurz sein. Dabei ist es egal, wie die Kantenlänge im ILP berechnet wird, das heißt ob per Unendlich-Norm oder per Euklidischer Norm. Die Variablen für die Kantenlängen werden also zur Zielfunktion hinzugefügt.

$$\min\ C_{Kanten} = \sum_{e_{i_j,u}} l(e_{i_j,u})$$

Nach dem Setzten der Zielfunktion ist die Modellierung des Grundmodells abgeschlossen. Die Berechnung der Ausgaben dauert allerdings sehr lange, wie man in Kapitel 5 sehen wird. Wir versuchen daher im nächsten Kapitel, das Grundmodell so zu variieren, dass die Berechnung beschleunigt wird.

4. Variationen des Grundmodells zur beschleunigten Berechnung

Implementieren wir das Grundmodell so wie wir es bisher beschrieben haben, kann der ILP-Löser nur dann eine Lösung ausgeben, wenn er die Berechnung vollkommen abschließt und eine beweisbar optimale, d.h. minimale, Lösung bezüglich des Grundmodells findet. Dies dauert für die meisten Eingaben jedoch zu lange, oder es kann gar keine optimale Lösung bestimmt werden. Im Folgenden werden Ansätze beschrieben, die die Laufzeit der Berechnung beschleunigen sollen. Ein erstes Ziel ist es, mögliche Lösungen zu finden, die alle Constraints erfüllen, aber nicht zwingend minimal sind. Diese Lösungen nennen wir *zulässige Zwischenlösungen*. Weitere Ansätze bestehen darin, die Anzahl der Variablen und Constraints auf eine bestimmte Weise zu verringern und so die Laufzeit zu verkürzen. Wir schlagen mehrere Variationen des Grundmodells vor, die dies erreichen könnten. Im darauf folgenden Kapitel 5 werden diese möglichen Variationen auf ihre Effizienz geprüft.

4.1. Callbacks und Lazy-Constraints

Bisher gibt der ILP-Löser dem aufrufenden Programm nur beweisbar optimale, d.h. minimale Lösungen bezüglich des Eingabemodells aus. Da die Aufgabe des Lösers ist, optimale Lösungen zu finden, werden diese optimalen Lösungen offensichtlich nur am Ende der Berechnung ausgegeben. Über weitere mögliche zulässige Lösungen gibt er jedoch keine Auskunft. Da optimale Lösungen oft nur sehr langsam gefunden werden können, soll der ILP-Löser auch zulässige Zwischenlösungen ausgeben. Dafür gibt es sogenannte *Callbacks* in unterschiedlichen Varianten. Der *Gurobi-Mipsol-Callback* wird vom Gurobi-ILP-Löser immer dann aufgerufen, wenn er eine zulässige Zwischenlösung findet. Dies nutzen wir aus, um überhaupt Ausgaben für die Eingabedatei zu bekommen. Es liegt dann im Ermessen des Benutzers, ob er diese zulässigen, aber nicht optimalen Lösungen als gut erachtet. Meist reicht es aus, wenn das Bild der besten Lösung nahe kommt, ohne mathematisch beweisbar optimal zu sein.

Eine weitere Möglichkeit, den ILP-Löser zu nutzen, besteht in der Implementierung von sogenannten *Lazy-Constraints*. Diese sind nicht im Original-Modell vorhanden, sondern werden erst während der Berechnung eingefügt. Die Idee dahinter ist, dass nicht immer alle Constraints von Nöten sind, um eine zulässige Lösung zu finden. Der ILP-Löser muss diese überschüssigen Constraints aber trotzdem mitberücksichtigen. Dadurch wird die Laufzeit unnötig verlängert.

Im bisherigen ILP-Modell wurden in Abschnitt 3.4 Constraints eingefügt, um Kanten-Stations-Schnitte zu verhindern. Dabei wird bei jeder Zwischenlösung stets jede Kante gegen jede Station vom aufrufenden Programm auf einen Schnitt geprüft. Dies ist jedoch nicht nötig, da nicht jede Kante jede Station kreuzt. Um den Löser zu entlasten, werden die Constraints für die Kreuzungsfreiheit wieder aus dem Modell entfernt. Der Löser wird aufgerufen und versucht nun, eine zulässige Lösung zu finden. Findet er eine solche zulässige Lösung, so wird der Callback aufgerufen und das aufrufende Programm testet die Zwischenlösung auf eventuelle Kanten-Stations-Schnitte. Schneidet eine Kante eine Station, so werden die Constraints für die Kreuzungsfreiheit für genau dieses Station-Kanten-Paar als Lazy-Constraints vom Callback eingefügt und der ILP-Löser setzt die Berechnung mit den neuen Constraints fort. Auf diese Weise werden nur so viele Constraints wie nötig in das Modell eingefügt.

4.2. Hauptpfad

Im folgenden werden Ansätze vorgestellt, die Berechnung durch das Festlegen bestimmter Werte oder Variablen zu beschleunigen.

Der Hauptpfad ist ein Pfad im Graph. Er startet beim Startknoten v_1 und endet im Stoppknoten v_n. Er beschreibt die schnellste Verbindung im Abfahrtsgraph. Er verfolgt stets die erste Kante, die aus einer Station ausgeht, vom Start- bis zum Stoppknoten. Es liegen also nicht immer alle Stationen auf dem Hauptpfad. Abbildung 4.1 zeigt eine Originalausgabe aus der Arbeit von Dibbelt et al. [DSW14], in dem der Hauptpfad fett eingezeichnet ist. Man beachte, dass in der Originalausgabe Ankunftsknoten vorhanden sind, die in der Ausgabezeichnung des ILP-Modells nicht vorhanden sind.

Wir benennen den Hauptpfad durch seine Knoten mit

$$h = v_{1_1}, \ldots, v_{i_1}, \ldots, v_n$$

Da die Kanten auf dem Hauptpfad immer in dem obersten Abfahrtsknoten einer Station starten, kann man den Pfad auch anhand seiner Kanten darstellen als

$$h = e_{1_1,k}, \ldots, e_{i_1,u}, \ldots, e_{j_1,n}$$

Dabei sind k, i, u und j beliebige Stationen auf dem Pfad.

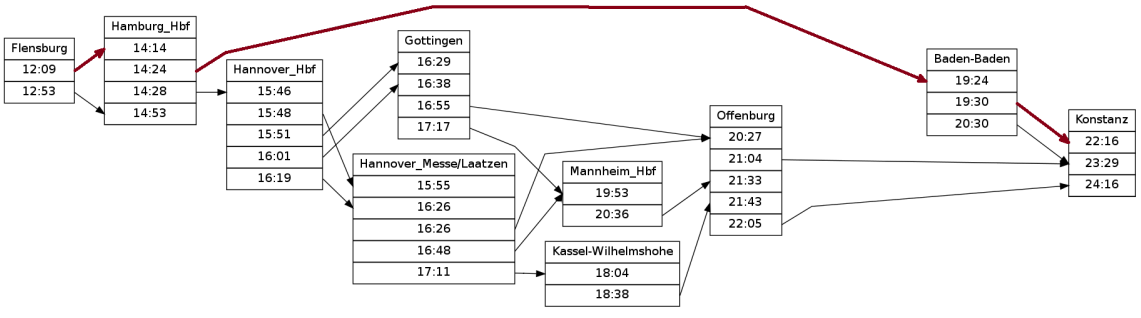


Abbildung 4.1.: Beispiel für den Hauptpfad in der Verbindung Flensburg-Konstanz

Die höchstgewichtigsten Kanten im Eingabegraph sind die Kanten auf dem Hauptpfad, da diese die schnellste Verbindung darstellen. Ein Ziel der Modellierung ist, hochgewichtige Kanten möglichst gerade, d.h. knickfrei zu zeichnen. Da bekannt ist, dass der Hauptpfad also möglichst gerade sein soll, kann diese Tatsache das Lösen des ILPs eventuell einfacher machen. Dazu werden die Werte bestimmter Variablen festgelegt.

Wir legen fest, dass alle Stationen auf dem Hauptpfad in derselben Höhe starten sollen und die Pfadkanten zwischen diesen Stationen knickfrei und stets "in Leserichtung", d.h. von links nach rechts, verlaufen sollen. Dies erzwingt, dass die einzelnen Stationen nebeneinander auf verschiedenen Gitterstäben liegen. Dazu werden die in Abschnitt 3.2.1 vorgestellten Gittervariablen ϕ_i festgelegt. Der Startknoten liegt schon stets auf dem linkesten Gitterstab, deshalb werden alle weiteren Stationen nun in der Reihenfolge auf die Gitterstäbe verteilt, in der sie auf dem Hauptpfad vorkommen. Besteht der Hauptpfad aus nur einer Kante, kann dies dazu führen, dass auf dem Canvas kein Platz mehr für andere Stationen ist. Der Stoppknoten soll stets rechts am Bildrand sein. Läge er auf dem zweiten Gitterstab und der Graph hätte viele andere Stationen, so könnten diese nicht verteilt werden und das Modell würde unlösbar. Deshalb geben wir vor, dass der Stoppknoten für kleinere Eingaben mindestens auf dem dritten Gitterstab liegen soll. Hat der Hauptpfad mehr als drei, beispielsweise $p+1$ Stationen, so soll der Stoppknoten auf dem p-ten Gitterstab liegen. Wir können also die Gittervariablen ϕ_i festsetzen. Durch die feste Gitterweite können die x-Koordinaten der entsprechenden Stationen außerdem ebenfalls berechnet und festgelegt werden. Im Übrigen muss die Zeichenfläche noch entsprechend angepasst werden. Dies geschieht durch die entsprechende Festlegung der Canvasweite Cw.

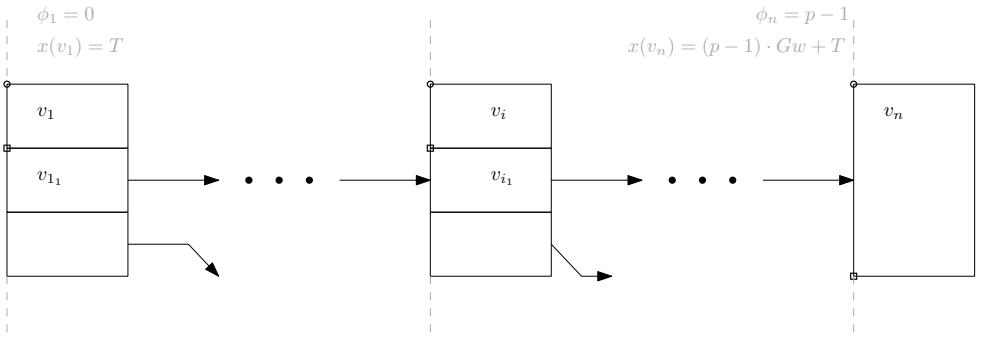


Abbildung 4.2.: Beschränkung der Koordinaten auf dem Hauptpfad

Außerdem sollen alle Stationen auf dem Hauptpfad auf der selben Höhe beginnen, d.h. der linke obere Eckpunkt der Stationsknoten soll stets dieselbe y-Koordinate haben. Dieser berechnet sich aus dem Ankerpunkt und der Höhe des Stationsknotens. Für alle Stationsknoten v_i auf dem Hauptpfad muss also gelten:

$$y(v_1) + H = y(v_i) + H = y(v_n) + H(v_n)$$

Nun kann die Berechnung noch beschleunigt werden, indem Werte für die Kanten $e_{i_j,u}$ auf dem Hauptpfad festgesetzt werden. Um die Leserichtung zu erhalten, soll jede Kante von links nach rechts verlaufen, d.h. sie soll ihren Abfahrtsknoten rechts verlassen und links in ihre Zielstation einlaufen. In Abschnitt 3.3.2 wurden Variablen für die Richtungsentscheidung eingeführt. Diese werden dementsprechend festgelegt auf

$$\alpha(e_{i_j,u}) = 0$$
$$\beta(e_{i_j,u}) = 1$$

Des Weiteren sollen die Kanten auf dem Hauptpfad gerade, das heißt knickfrei, gezeichnet werden. Dazu werden für alle Segmente $q(e_{i_j,u})$ auf den Hauptpfadkanten die in Abschnitt 3.3.3.1 vorgestellten absoluten Richtungsvariablen auf die Richtung "rechts" festgelegt:

$$s_0(e_{i_j,u}) = 0$$
$$s_l(e_{i_j,u}) = 0$$
$$s_q(e_{i_j,u}) = 0$$

Die relativen Richtungsvariablen, die ein Knickindikator sind, werden ebenso wie die Knickvariablen selbst auf "knickfrei" festgelegt:

$$\begin{aligned} r_q(e_{i_j,u}) &= 0 \\ z_q(e_{i_j,u}) &= 0 \\ r(e_{i_j,u}) &= 0 \\ z(e_{i_j,u}) &= 0 \end{aligned}$$

Zuletzt werden die Variablen, die für die Verknüpfung der Segmentrichtung mit den Koordinaten der Dummyknoten verantwortlich sind, auf die entsprechende Richtung festgelegt. Die Variablen, deren Richtung nicht gewählt ist, werden genullt:

$$\begin{aligned} \gamma_{q,0} &= 1 \\ \gamma_{q,p\neq 0} &= 0 \end{aligned}$$

Wieviele Constraints durch diese Festlegung tatsächlich wegfallen, wird in den Experimenten in Kapitel 5 evaluiert.

4.3. Diagonalen entfernen

Ein weiterer Ansatz besteht darin, die möglichen Richtungen zu beschränken, die Kanten nehmen können. Ein Beispiel ist, aus dem octolinearen System ein orthogonales System zu machen. Das heißt, die Diagonalen werden verboten und die Kantensegmente dürfen nur horizontal und vertikal verlaufen. Ausgehend vom Grundmodell gibt es dabei aber folgendes Problem: in Abschnitt 3.3.3.1 wurden relative Richtungsvariablen $r_q(e_{i_j,u})$ eingeführt. Diese können je nach Kickrichtung die Werte $\{-1, 0, 1\}$ annehmen. Dies ist aber durch das restliche Modell auf einen Knick von 45° festgelegt. Werden den Kanten nur horizontale und vertikale Richtungen erlaubt, so müssen aber Knicke von 90° möglich sein.

Um dieses Problem zu beheben, wird der Zahlenbereich der relativen Richtungen aller Kantensegmente erweitert:

$$\begin{aligned} r_q(e_{i_j,u}) &\in \{-2, -1, 0, 1, 2\} \\ z_q(e_{i_j,u}) &\in \{0, 1, 2\} \end{aligned}$$

Hier bedeutet nun $r_q = 2$ einen Linksknick um 90°, $r_q = -2$ einen Rechtsknick um 90° und $r_q = 0$ geradeaus, wie gehabt. Knicke um 45° sind dabei theoretisch noch erlaubt.

Nun müssen die Diagonalen verboten werden. Dazu werden wir für alle Kantensegmente im Graph die entsprechenden Entscheidungsvariablen auf Null gesetzt. Variablen für die horizontalen und vertikalen Richtungen bleiben weiterhin frei und müssen vom ILP-Löser bestimmt werden:

$$\gamma_{q,1}(e_{i_j,u}) = \gamma_{q,3}(e_{i_j,u}) = \gamma_{q,5}(e_{i_j,u}) = \gamma_{q,7}(e_{i_j,u}) = 0$$

Da der ILP-Löser nun keine Richtungen mehr mit Diagonalen besetzten kann, werden 45°-Knicke automatisch ausgeschlossen. Dies ist ein intuitiver Ansatz, der kaum Änderungen am Grundmodell vornimmt. Andere Modellierungen könnten effizienter sein, jedoch begnügen wir uns an dieser Stelle damit, dass unsere Modellierung die gewünschte Funktionalität erfüllt.

4.4. Blöcke

Der letzte Ansatz, die Berechnung zu verkürzen, besteht darin die Eingabe selbst und das dadurch erzeugte Modell zu verkleinern. In manchen Graphen gibt es Stationen, deren Kanten stets zur selben Zielstation verlaufen und denselben Zugtyp haben. Häufig ist dies der Fall im Nahverkehr, beispielsweise bei regelmäßig abfahrenden S-Bahnen. Diese Abfahrtsknoten können zusammengefasst werden. Ihr Zeitlabel entspricht dann einem Zeitintervall. Auch die ausgehenden Kanten können zu einer einzelnen Zugkante zusammengefasst werden. Die zusammengefassten Abfahrtsknoten werden dann zu einem neuen Abfahrtsknoten, einem sogenannten *Block*.

Die Berechnung von Blöcken geschieht algorithmisch, bevor das ILP-Modell konstruiert wird. Dabei wird innerhalb jeder Station der Blockalgorithmus aufgerufen. Der Algorithmus iteriert über alle Abfahrtsknoten einer Station und vergleicht stets zwei aufeinander folgende Knoten. Haben die ausgehenden Kanten den gleichen Zugtyp und dasselbe Ziel, werden sie zu einer Kante verschmolzen und die beiden Abfahrtsknoten werden zu einem Knoten mit Intervall-Label. Dabei muss darauf geachtet werden, die Indices der nachfolgenden Abfahrtsknoten neu zu nummerieren. Wenn das Modell dann erstellt wird, wird der Block vom ILP-Löser wie ein normaler Abfahrtsknoten behandelt. Abbildung 4.3 veranschaulicht die Blockbildung.

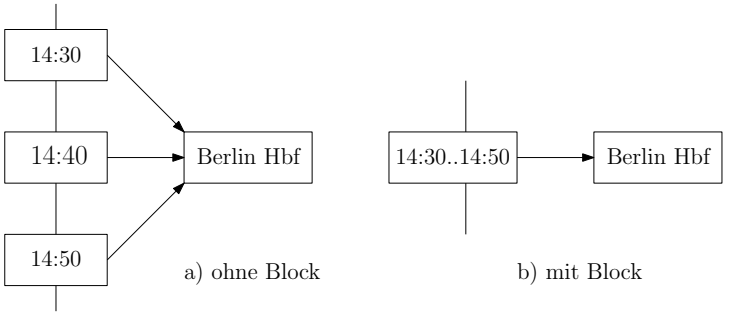


Abbildung 4.3.: Blockbildung

5. Experimente

Dieses Kapitel widmet sich der Analyse des Modells in Bezug auf verschiedene Eingaben. Außerdem soll geprüft werden, in wie weit die im vorherigen Kapitel vorgeschlagenen Varianten die Laufzeit von Lösungsberechnungen tatsächlich verbessern. Zum Schluss werden Folgerungen aus den Testergebnissen gezogen.

5.1. Rechner

Die Experimente wurden unter Microsoft Windows 7 Professional 64-Bit durchgeführt. Das zugrunde liegende System ist ein 64-Bit Intel i5-2500K Prozessor mit 4 Kernen, 3.3GHz Takt und 16GB Arbeitsspeicher. Das Programm wurde mit Eclipse Standard SDK Kepler entwickelt und getestet. Die verwendete Java-Version ist Version 1.7.0 Update 67, 64-Bit.

5.2. Instanzen

Dieser Abschnitt beschreibt die einzelnen Eingabegraphen, auf denen die verschiedenen Modell-Variationen getestet wurden. Tabelle 5.1 gibt eine Übersicht über die einzelnen Eingabegraphen, ihrer Anzahl an Stationen n und ihrer Anzahl an Kanten m.

Tabelle 5.1.: Übersicht über die Eingaben und ihre Größen

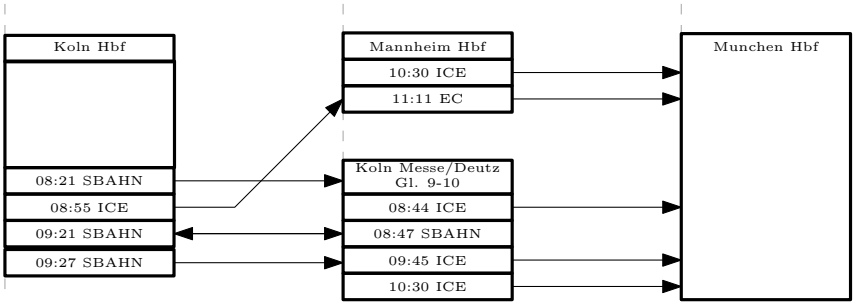
Eingabe	Stationen n	Kanten m
Trivialinstanz	2	1
Köln-München	4	10
Basel-Prenzlau	5	15
Flensburg-Konstanz	8	13
Loppenhausen-Paracin	9	13
Bari-Moskau	10	17
Karlsruhe-Trier	6	30

Die Trivialinstanz besteht dabei aus einem Ein-Kanten-Beispiel. Die Eingabe hat also genau einen Start- und Endknoten und dazwischen genau eine Zugkante. Abbildung 5.1 zeigt die minimale Lösung für diese Instanz bezüglich des Grundmodells.

Die nächst größere Eingabeinstanz ist der Graph für die Verbindung *Köln-München*. Es ist eine kleine Instanz und die einzige Eingabe außer der Trivialinstanz, für die der ILP-Löser

Abbildung 5.1.: Trivialinstanz

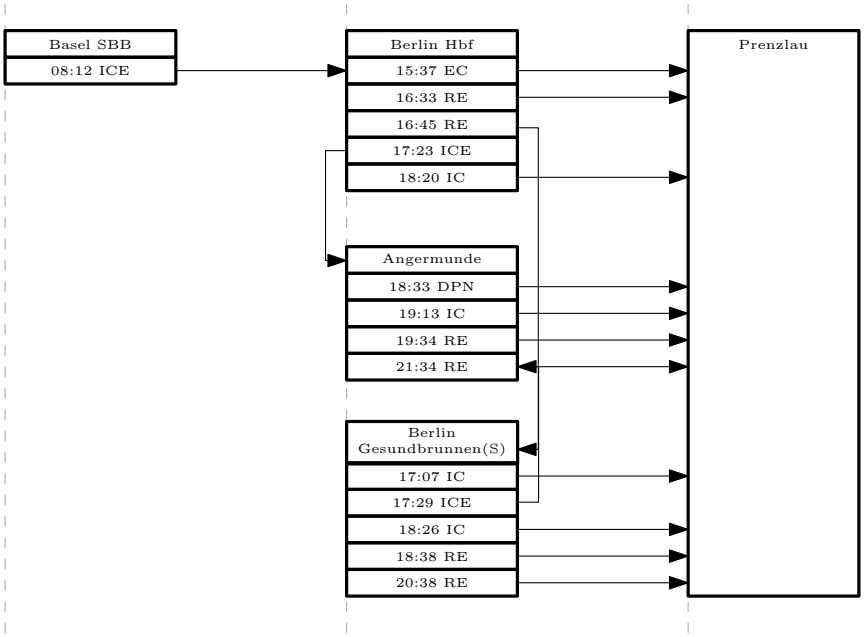
Abbildung 5.2.: Beispiel: letzte Lösung für das Grundmodell



unabhängig von der Modellvariante eine Lösung finden konnte. Deshalb wird diese Instanz im Folgenden als Erläuterungsbeispiel verwendet. Ein Beispiel findet sich in Abbildung 5.2.

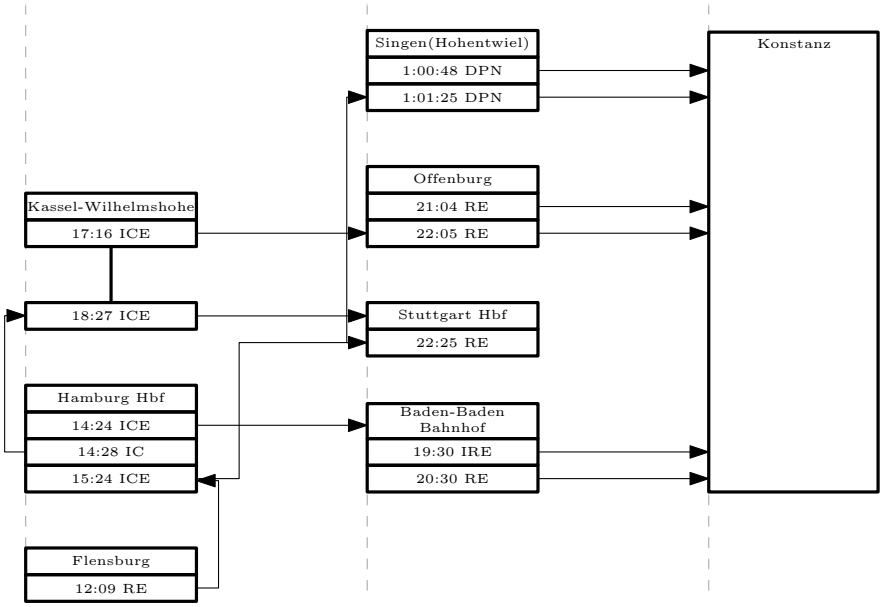
Die Eingabe Basel-Prenzlau hat fünf Stationen und fünfzehn Kanten. Sie gehört zu den mittleren Eingaben. Abbildung 5.3 zeigt ein Beispiel. Das Bild zeigt die vom Löser zuletzt gefundene Lösung im Rahmen der Testzeit. Es ist zu sehen, dass Informationen verloren gehen, da Kanten-Kanten-Kreuzungen nicht von unserem Modell und seinen Varianten abgedeckt werden. Trotzdem ist dies die beste Lösung, die im Rahmen der Testzeit für diese Eingabeinstanz gefunden werden konnte.

Abbildung 5.3.: Basel-Prenzlau, mit Hauptpfad, ohne Diagonalen



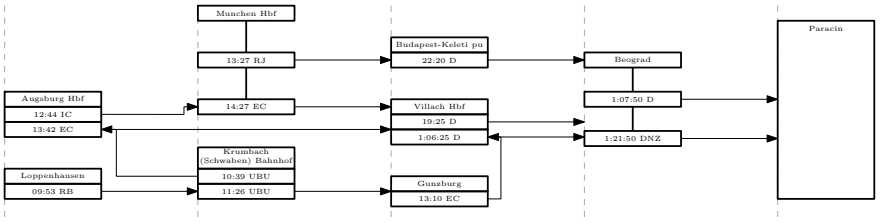
Der Eingabegraph für die Verbindung Flensburg-Konstanz hat acht Stationen und dreizehn Kanten. Er gehört zu den mittleren Eingabeinstanzen. Abbildung 5.4 zeigt eine Beispiellösung.

Abbildung 5.4.: Flensburg-Konstanz, ohne Hauptpfad, ohne Diagonalen



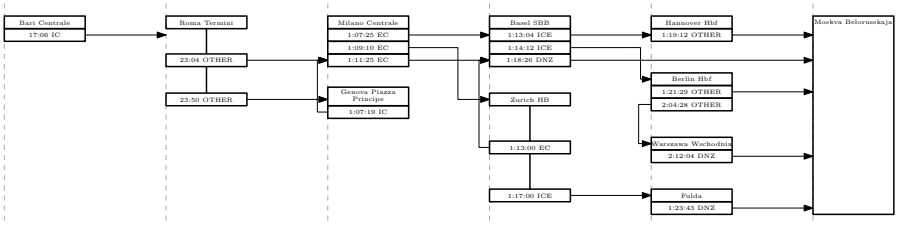
Die Eingabegraph für die Verbindung Loppenhausen-Paracin zählt zu den mittleren Eingabeinstanzen. Er hat viele Stationen, die untereinander stark vernetzt sind. Damit soll getestet werden, ob stark vernetzte Graphen gut kreuzungsfrei gezeichnet werden können, oder ob dies mehr Zeit in Anspruch nimmt. Abbildung 5.5 zeigt eine der wenigen Lösungen, die für diese Instanz generiert werden konnten. Man sieht, dass eine Kante von *Villach* nach *Beograd* nicht direkt an eine Box andockt. Dies ist kein Fehler, da Kanten auf der gesamten Höhe ihrer Zielstationen ankommen dürfen.

Abbildung 5.5.: Loppenhausen-Paracin, ohne Hauptpfad, ohne Diagonalen



Die zweitgrößte Eingabeinstanz, Bari-Moskau, hat die meisten Stationen. Damit sollen die Platzierungsmöglichkeiten für Stationen getestet werden. Abbildung 5.6 zeigt eine Beispiellösung.

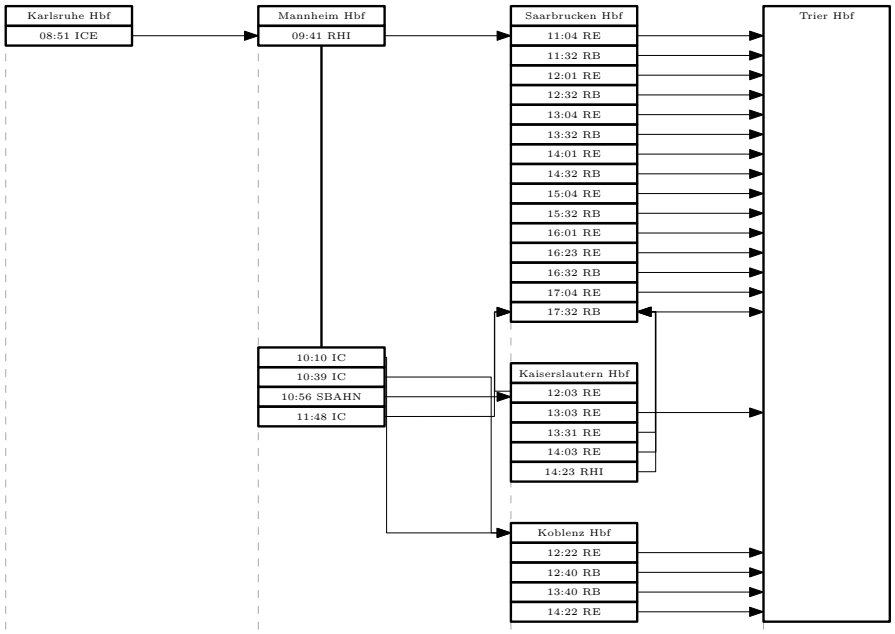
Abbildung 5.6.: Bari-Moskau, mit Hauptpfad, ohne Diagonalen



Für die größte Eingabeinstanz, die Verbindung Karlsruhe-Trier, werden im Grundmodell ohne Varianten über Zehntausend Variablen erzeugt. Die Eingabe zeichnet sich außerdem

dadurch aus, dass viele Kanten zu Blöcken zusammengefügt werden können. Abbildung 5.7 zeigt ein Beispiel.

Abbildung 5.7.: Karlsruhe-Trier, mit Hauptpfad, ohne Diagonalen



5.3. Laufzeitmessungen

Im Folgenden werden Laufzeiten für verschiedene Variationen des Modells und Kombinationen daraus vorgestellt. Die verfügbare Experimentalzeit war begrenzt. Bei den ersten unbefristeten Experimenten mit dem Grundmodell wurde vom ILP-Löser entweder sehr schnell oder gar keine Lösung gefunden. Wir möchten die Testzeit jedoch beschränken und gehen davon aus, dass der ILP-Löser für Modellvarianten ebenfalls entweder sehr schnell eine Lösung findet oder aber gar keine Lösung in einer von uns festgelegten Zeit finden kann. Deshalb wurden die Experimente auf eine Stunde pro Durchlauf beschränkt.

Was wir bereits wissen ist, dass sich orthogonale Layouts mittels Flussalgorithmen zeichnen lassen. Sie sind also leichter lösbar als Layouts mit Diagonalen. Dies werden wir auch bezüglich unseres Modell feststellen.

Die einzelnen Teilterme der Zielfunktion werden in unseren Experimenten stets gleich gewichtet. Um einen Effekt von unterschiedlicher Gewichtung auf die Laufzeiten des Modells festzustellen, müssten weitere Tests durchgeführt werden. Dazu war jedoch keine Zeit. Die Kantengewichte der Zugkanten in den Eingabegraphen betragen alle eins. Dies bedeutet, dass der Hauptpfad eigentlich nicht zwingend der wichtigste Pfad im Eingabegraph ist. Wir testen die von uns festgelegte Variation trotzdem auf ihre Effizienz.

In den folgenden tabellarischen Übersichten 5.2 bis 5.8 finden sich die Eingabeinstanzen mit den für die entsprechende Modellvariation initialisierten Constraints und Variablen aufgelistet. Dabei ist zu beachten, dass der ILP-Löser vor der eigentlichen Berechnung der Lösungen eine Vorberechnung (engl. *Presolve*) durchführt. Diese eliminiert überflüssige Constraints und Variablen. Da in den von uns vorgeschlagenen Varianten nicht weniger Variablen angelegt werden, ist es interessant zu sehen, wie viele Constraints und Variablen von dem Presolve-Algorithmus entfernt werden. C_{init} bezeichnet die Anzahl der angelegten Constraints, C_{pre} die Anzahl der Constraints für die entprechende Variation nach dem

Presolve-Algorithmus. V_{init} bezeichnet die Anzahl der angelegten Variablen für eine Eingabeinstanz. Diese verändert sich nur durch die Blockbildung. V_{pre} bezeichnet die Anzahl der übrigen Variablen nach dem Presolve-Algorithmus.

In der letzten Spalte finden sich die Laufzeiten der ersten gefundenen Lösung und der, innerhalb des Testrahmens gefundenen, letzten Lösung in Sekunden. Wichtig ist zu wissen, dass der Löser nur dann eine neue Lösung ausgibt, wenn diese bezüglich der Zielfunktion besser ist als die vorherige. Dies bedeutet also, dass viele Lösungen für eine Eingabe gefunden werden können, diese jedoch nicht unbedingt ausgegeben werden. Dabei ist zu beachten, dass der ILP-Löser für alle Eingaben außer der Trivialeingabe niemals die Optimalität bezüglich einer Modell-Variante berechnen konnte. Wir geben deshalb noch den Abstand (engl. *gap*) der letzten gefundenen Lösung, zur vom ILP-Löser geratenen heuristischen Lösung an. Die heuristische Lösung ist der vom ILP-Löser geratene Minimalwert, d.h. eine untere Schranke, bezüglich einer Eingabeinstanz und einer Modellvariation. Eine bereits gefundene Lösung bildet eine obere Schranke. Das heißt, dass eine Lösung optimal (hier minimal) ist, wenn $gap = 0\%$ ist. Ist dagegen $gap = 100\%$, so ist die gefundene Lösung noch weit von der vom ILP-Löser geratenen Minimallösung, der heuristischen Lösung, entfernt. Ob dieser Wert sinnvoll ist, um Vergleiche zu ziehen, werden wir später diskutieren.

5.3.1. Lazy-Constraints und das Grundmodell

Während der Implementierung des Grundmodells wurde die Kreuzungsfreiheit von Kanten und Stationen als normale Constraints in das ILP eingefügt. Dadurch verschlechterte sich die Laufzeit für Testeingaben erheblich. Danach wurden die Kreuzungsfreiheit mittels Callbacks als Lazy-Constraints zum Grundmodell hinzugefügt. Lazy-Constraints werden während der Berechnung eingefügt, wenn Bedarf besteht. Im diesem Fall wurde immer dann ein Lazy-Constraint eingefügt wenn in einer bis dahin zulässigen Zwischenlösung ein Station-Kanten-Schnitt gefunden wurde. Tabelle 5.2 zeigt die Anzahl der Constraints und Variablen für verschiedene Eingabeinstanzen bezüglich des Grundmodells ohne Lazy-Constraints im Vergleich zum Grundmodell mit Lazy-Constraints.

Tabelle 5.2.: Übersicht über die Variablen und Constraints des Grundmodells mit und ohne Lazy-Constraints

Eingabe	Grundmodell mit Lazy-Constraints				Grundmodell ohne Lazy-Constraints			
	C_{init}	C_{pre}	V_{init}	V_{pre}	C_{init}	C_{pre}	V_{init}	V_{pre}
Trivial	284	229	194	163	420	354	194	156
Ko-Mu	2688	2336	2580	2342	5748	5232	2580	2237
Ba-Pr	4050	3517	4470	4119	9915	9167	4470	3992
Fl-Ko	3950	3181	5618	5215	12348	11299	5618	5121
Lo-Pa	4130	3231	6210	5562	13633	12480	6210	5680
Ba-Mo	5572	4277	9538	8933	20889	19313	9538	8828
Ka-Tr	7974	6866	10.068	9406	22254	20732	10068	9232

Bei der Implementierung der Kreuzungsfreiheit als normale Constraints verdoppelt oder verdreifacht sich die Zahl der gesamten Constraints in etwa. Die Anzahl der Variablen hingegen bleibt ähnlich. Die Lösungsberechnung allein für die kleinste der Eingaben, die Verbindung *Köln-München* reizte die vorgegebene Maximalzeit schon vollkommen aus. Nach über 22 Stunden konnte keine Lösung gefunden werden. Die Tests für die größeren Eingabeinstanzen ohne Lazy-Constraints wurden daher verworfen. Alle weiteren Tests enthalten die Kreuzungsfreiheit somit als Lazy-Constraints.

5.3.2. Das Grundmodell: ohne Hauptpfad, mit Diagonalen

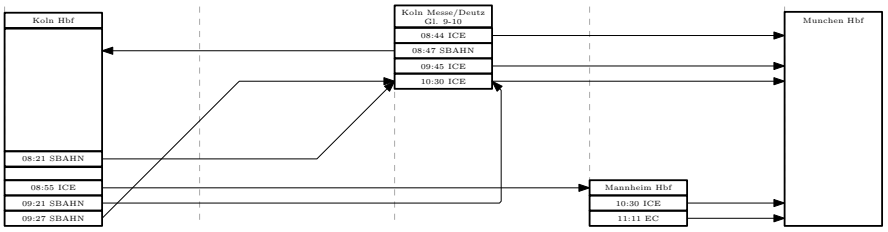
Zuerst wurden Eingaben mit dem Grundmodell ohne Varianten getestet. Das heißt, dass die Hauptpfad-Variante ausgeschaltet blieb und Diagonalen erlaubt waren. Dabei ist festzustellen, dass für die meisten Eingaben im Rahmen der Testzeit trotz Lazy-Constraints keine Lösung gefunden werden konnte.

In Tabelle 5.3 sieht man die Anzahl der Variablen und Constraints vor und nach dem Presolve-Algorithmus. Abbildung 5.8 zeigt die erste erhaltene Lösung für das Grundmodell für die Eingabe *Köln-München*. Man kann auf den ersten Blick erkennen, dass die Lösung schlecht ist, da die Zwischenstationen noch nach links verschoben und die Kantenlängen somit gekürzt werden können. In Abbildung 5.2 sieht man die letzte Lösung für dieselbe Eingabe. Bei genauerer Betrachtung erkennt man einen Doppelpfeil zwischen den Stationen *Köln* und *Köln Messe*. Dies ist kein Fehler. Das Modell schließt die Überlagerung von Kanten untereinander nicht aus. Es macht vom Informationsgehalt hier auch keinen Unterschied, da klar dargestellt wird, dass sowohl eine Hin- als auch eine Rückverbindung besteht und wann diese abfahren.

Tabelle 5.3.: Übersicht über die Variablen, Constraints und Laufzeiten für das Grundmodell

Eingabe	C_{init}	C_{pre}	V_{init}	V_{pre}	erste *(s)*	letzte *(s)*	gap *(%)*
Trivial	284	229	194	163	0	0	0
Ko-Mu	2688	2336	2580	2342	580	652	100
Ba-Pr	4050	3517	4470	4119	-	-	-
Fl-Ko	3950	3181	5618	5215	-	-	-
Lo-Pa	4130	3231	6210	5562	-	-	-
Ba-Mo	5572	4277	9538	8933	-	-	-
Ka-Tr	7974	6866	10.068	9406	-	-	-

Abbildung 5.8.: Beispiel: erste Lösung für das Grundmodell



5.3.3. Die Hauptpfad-Variante, mit Diagonalen

Im zweiten Experiment wurde die Hauptpfad-Verbesserung in das Modell eingeführt. Die Grundidee des Hauptpfades ist, hochgewichtige Kanten gerade zu zeichnen. Da die Kanten in den Eingabeinstanzen alle dieselben Kantengewichte haben, nämlich $g(e) = 1$, testen wir die Hauptpfad-Variante nur bezüglich der Effizienz, nicht bezüglich der Bevorzugung von besonderen Kanten.

Tabelle 5.4 zeigt die Variablen, Constraints und Laufzeiten für die Hauptpfad-Verbesserung. Die Hauptpfad-Variante brachte für die meisten Eingaben zwar eine Verbesserung hinsichtlich der Anzahl an Constraints und Variablen, allerdings konnte der ILP-Löser trotzdem für die meisten Eingaben immernoch keine Lösung finden. Die Laufzeit wurde zwar beschleunigt, aber es konnte trotzdem keine Optimalität für die Vebindung *Köln-München* bewiesen werden. In Abbildung 5.9 sieht man die erste erhaltene Lösung für die Verbindung *Köln-München* mit der Hauptpfad-Variante. Man erkennt, dass eine Zugkante von

Mannheim nach München deutlich verkürzt werden könnte, wenn sie aus der anderen Seite austräte. Abbildung 5.10 zeigt wieder die letzte Lösung für die Eingabe *Köln-München*, ebenfalls mit Hauptpfad-Variante. Die oben genannte Kante tritt nun rechts aus und ist dadurch kürzer. Deutlich zu sehen ist hier eine Umordnung der Stationen im Vergleich zum Grundmodell, sodass die Stationsknoten auf dem Hauptpfad alle auf derselben Höhe liegen. In diesem Fall sieht man ebenfalls zwei übereinander liegende Kanten. Dies stellt hier ebenso kein Problem dar, da keine Information verloren geht. Vergleicht man die Lösung mit der obigen Lösung in Abbildung 5.2 so erkennt man zwei statt einem Kantenknick. Die Lösung ist trotzdem gut, da eine Verschiebung des *"08:55 ICE"*-Knotens nach oben und der Abfahrtsknoten von *Köln Messe* nach unten zwar Knicke entfernt, die Kanten und die Stationshöhe aber deutlich größer macht. Dies würde auch eine größere, und somit aus Sicht des ILP-Lösers schlechtere, Lösung darstellen.

Tabelle 5.4.: Übersicht über die Variablen, Constraints und Laufzeiten für die Hauptpfad-Variante

Eingabe	C_{init}	C_{pre}	V_{init}	V_{pre}	erste *(s)*	letzte *(s)*	gap *(%)*
Trivial	348	11	194	99	0	0	0
Ko-Mu	2815	1846	2580	2192	55	73	81,2
Ba-Pr	4177	3014	4470	3957	-	-	-
Fl-Ko	4329	1632	5618	4598	-	-	-
Lo-Pa	4635	1187	6210	5127	-	-	-
Ba-Mo	5888	2962	9538	8496	-	-	-
Ka-Tr	8164	6041	10.068	9141	-	-	-

Abbildung 5.9.: Beispiel: erste Lösung für die Hauptpfad-Variante

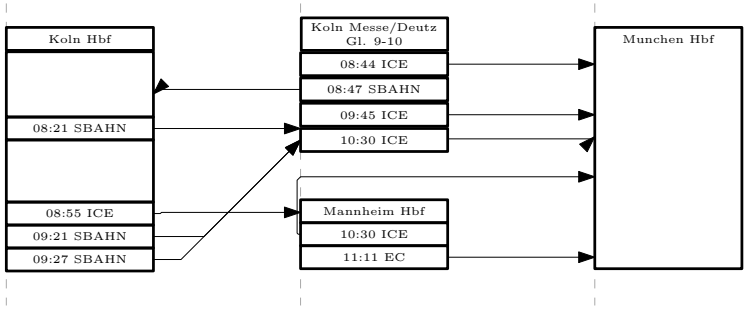
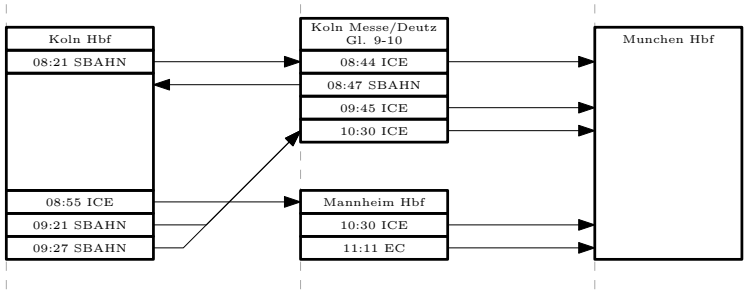


Abbildung 5.10.: Beispiel: letzte Lösung für die Hauptpfad-Variante



5.3.4. Variante ohne Diagonalen, ohne Hauptpfad

Im dritten Experiment wurden die Diagonalen entfernt. Die Hauptpfad-Variante wurde nicht angeschaltet. Dies brachte für alle Eingaben mindestens eine Lösung. Tabelle 5.5

zeigt die Anzahl der Variablen, Constraints und Laufzeiten für verschiedene Eingaben bei Entfernen der Diagonalen. Bemerkenswert ist, dass der Presolve-Algorithmus die überflüssigen Variablen und Constraints für die Trivialeingabe zwar berechnet, die erste Lösung für die Trivialinstanz aber schon gefunden wird, bevor der Presolve-Algorithmus endet. Der Löser ist für die Trivialeingabe mit dieser Variante also schneller als der Presolve-Algorithmus.

Abbildung 5.11 zeigt die erste erhaltene Lösung für die Verbindung *Köln-München*. Offensichtlich könnten die Kanten kürzer sein, wenn man die Stationen auf andere Gitterstäbe platziert und wenn einige Kanten auf der anderen Seite des Abfahrtsknotens austräten. In Abbildung 5.12 sieht man ein Lösungsbeispiel ohne Diagonalen der letzten erhaltenen Lösung für die Verbindung *Köln-München*. Da die Hauptpfad-Variante ausgeschaltet ist, platziert der ILP-Löser die Stationen wie im Grundmodell so, dass die Kanten möglichst kurz sind. Da die ersten beiden Zugkanten des Startknotens sich überschneiden und auch mit Umordnung der Stationen keine Überschneidungsfreiheit gegeben wäre, sind mindestens zwei Knicke vonnöten.

Tabelle 5.5.: Übersicht über die Variablen, Constraints und Laufzeiten für die Variante ohne Diagonalen

Eingabe	C_{init}	C_{pre}	V_{init}	V_{pre}	erste *(s)*	letzte *(s)*	gap *(%)*
Trivial	304	120	194	138	0	0	0
Ko-Mu	2888	1256	2580	2092	0	10	100
Ba-Pr	4350	1919	4470	3744	4	155	100
Fl-Ko	4210	2010	5618	4890	5	2606	100
Lo-Pa	4390	2092	6210	5444	10	185	100
Ba-Mo	5912	2736	9538	8516	160	1005	100
Ka-Tr	8574	4101	10.068	8664	380	800	100

Abbildung 5.11.: Beispiel: erste Lösung für die Variante ohne Diagonalen

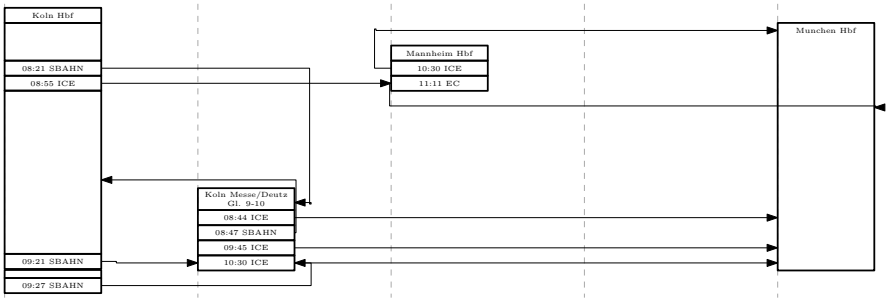
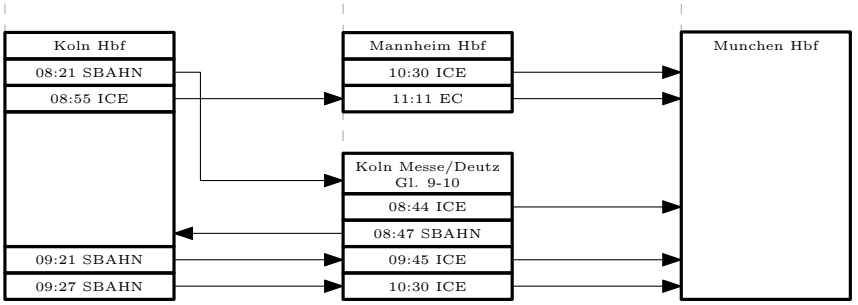


Abbildung 5.12.: Beispiel: letzte Lösung für die Variante ohne Diagonalen



5.3.5. Variante mit Blöcken, ohne Hauptpfad, mit Diagonalen

In einem weiteren Experiment wurde der Einfluss der Blockbildung im Eingabegraphen auf das ILP-Modell getestet. Da durch die Blockbildung nur der Eingabegraph, nicht aber das ILP-Modell selbst geändert wird, betrachten wir nur Eingaben, bei denen eine Blockbildung überhaupt möglich ist. Dabei bezeichne m' die neue Anzahl an Zugkanten bzw. Abfahrtsknoten im verringerten Blockgraph. In Tabelle 5.6 sieht man die durch die Blockgenerierung neuen Zahlen für Constraints und Variablen vor und nach dem Presolve-Algorithmus. Abbildung 5.13 zeigt die erste erhaltene Lösung der Blockvariante für die Verbindung *Köln-München*. Wie in obigen Erstlösungen schon gesehen, können die Kanten auch hier wieder durch Umplatzierung der Stationen verkürzt werden. In Abbildung 5.14 sieht man die letzte erhaltene Lösung für die Block-Variante für die Verbindung *Köln-München*. Vergleicht man diese Lösung mit der Lösung aus dem Grundmodell, so erkennt man deutlich, welche Kanten in Blöcke zusammengefasst wurden. Interessant ist, dass die Stationsumordnung nun automatisch erfolgt, da die zwei Kanten, die in den vorherigen Beispielen für zwei Knicke verantwortlich waren, nun zu einer Kante verschmelzen.

Tabelle 5.6.: Übersicht über die Variablen, Constraints und Laufzeiten für die Blockbildung im Grundmodell

Eingabe	n	m'	C_{init}	C_{pre}	V_{init}	V_{pre}	erste *(s)*	letzte *(s)*	gap *(%)*
Ko-Mu	4	8	2180	1874	2080	1881	105	1030	24,9
Ba-Pr	5	13	3542	3071	3890	3577	-	-	-
Fl-Ko	8	11	3442	2734	4798	4433	-	-	-
Lo-Pa	9	11	3622	2788	5310	4907	-	-	-
Ka-Tr	6	13	3656	3100	4458	4119	-	-	-

Abbildung 5.13.: Beispiel: erste Lösung für die Blockvariante

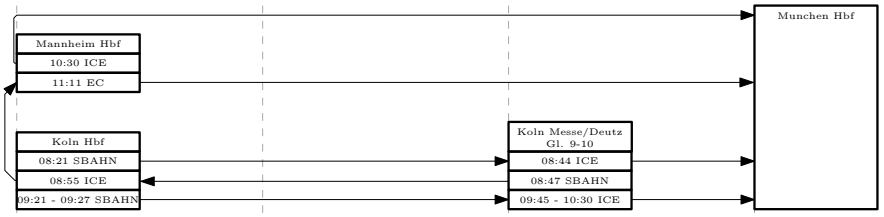
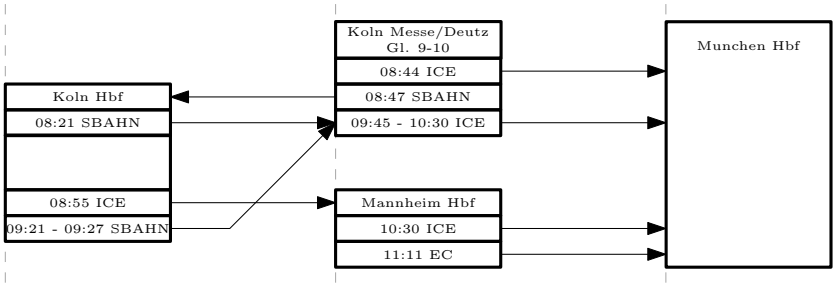


Abbildung 5.14.: Beispiel: letzte Lösung für die Blockvariante



5.3.6. Kombination der Hauptpfad-Variante ohne Diagonalen

Das nächste Experiment bestand aus einer Kombination von der Hauptpfad-Variante und dem Entfernen von Diagonalen. Auch hier konnte für die meisten Eingaben eine Lösung gefunden werden. In Tabelle 5.7 sieht man die Anzahl der Variablen, der Constraints und

die Laufzeiten für die erste und letzte Lösung der verschiedenen Eingaben. In Abbildung 5.15 sieht man die erste erhaltene Lösung für diese Kombination. Sie unterscheidet sich nicht stark von der letzten erhaltenen Lösung, allerdings sieht man einige überflüssige Kantenknicke. Die Kante von Köln nach Mannheim kann verkürzt werden, indem sie auf der anderen Seite austritt. Abbildung 5.16 zeigt die letzte erhaltene Lösung für die Verbindung *Köln-München* mit aktivierter Hauptpfad-Variante und ohne Diagonalen. Offensichtlich werden die Stationen wieder so umgeordnet, dass alle Stationsknoten des Hauptpfades auf derselben Höhe liegen. Hier besteht der Fall, dass übereinander liegende Kanten einen Verlust an Information aufzeigen. Es ist nicht sofort erkenntlich, welche Kante von wo nach wo geht.

Tabelle 5.7.: Übersicht über die Variablen, Constraints und Laufzeiten für die Kombination aus Hauptpfad und entfernten Diagonalen

Eingabe	C_{init}	C_{pre}	V_{init}	V_{pre}	erste *(s)*	letzte *(s)*	gap *(%)*
Trivial	386	6	194	98	0	0	0
Ko-Mu	3015	1093	2580	1992	0	1	81,7
Ba-Pr	4477	1807	4470	3632	0	15	87,3
Fl-Ko	4589	986	5618	4552	-	-	-
Lo-Pa	4895	711	6210	5002	-	-	-
Ba-Mo	6228	1904	9538	8200	20	2681	75,8
Ka-Tr	8764	3398	10.068	8466	15	75	91,8

Abbildung 5.15.: Beispiel: erste Lösung für die Hauptpfad-Variante ohne Diagonalen

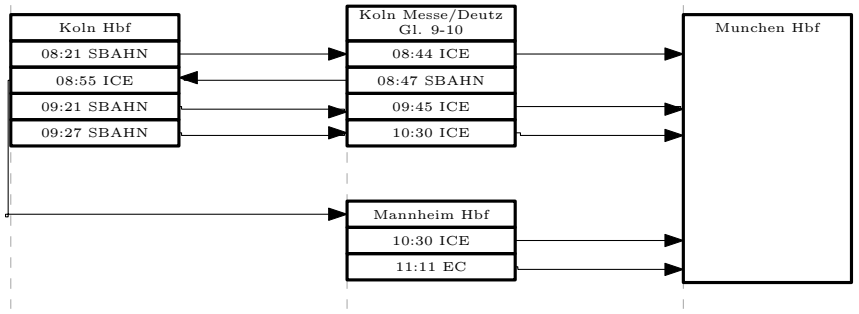
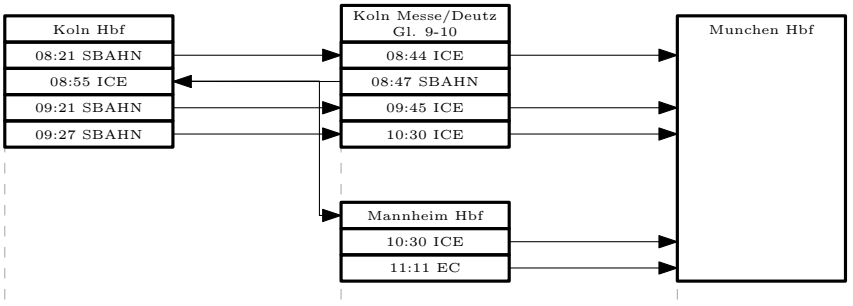


Abbildung 5.16.: Beispiel: letzte Lösung für die Hauptpfad-Variante ohne Diagonalen



5.3.7. Kombination der Block- und Hauptpfad-Varianten, ohne Diagonalen

Wie in der einzelnen Block-Variante wurden in diesem Experiment nur Eingaben getestet, die überhaupt Blöcke ermöglichen. Abbildung 5.17 zeigt die erste erhaltene Lösung für die Verbindung *Köln-München*. Wie auf den ersten Blick erkennbar ist, sind die vertikalen

Abstände zu groß, so dass die vertikalen Kanten extrem lang werden. Dies kann durch Verschieben der Stationen nach oben verbessert werden. Abbildung 5.18 zeigt die letzte erhaltene Lösung für dieselbe Testinstanz. Die Stationen wurden nach oben verschoben und die Kanten sind dadurch kürzer. Wie in der Hauptpfad-Variante ohne Diagonalen ohne Blöcke (Abb. 5.16) besteht auch hier das Problem des Informationsverlustes, da mehrere Kanten übereinander liegen.

Abbildung 5.17.: Beispiel: erste Lösung für die Dreier-Kombination aus Blöcken, Hauptpfad und entfernten Diagonalen

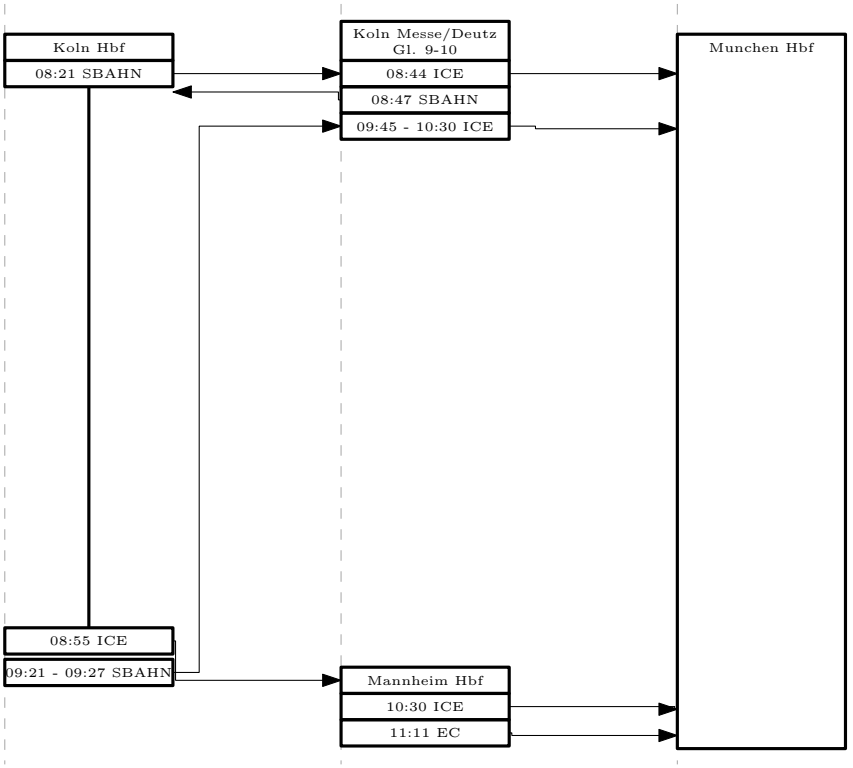
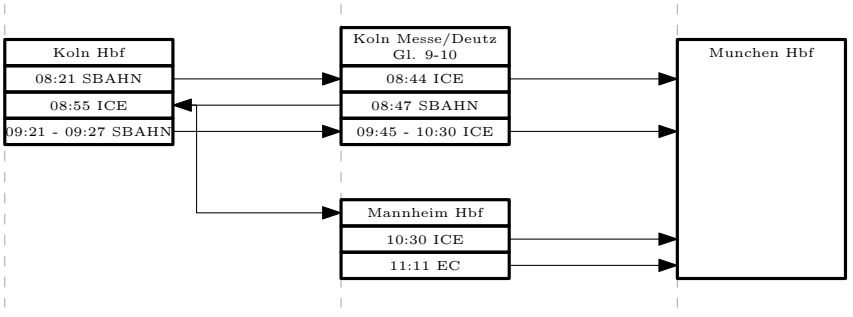


Abbildung 5.18.: Beispiel: letzte Lösung für die Dreier-Kombination aus Blöcken, Hauptpfad und entfernten Diagonalen



5.3.8. Unendlich-Norm im Vergleich zur Euklidischen Norm

Ein weitere Frage war, in wie weit die Wahl der Abstandsnorm die Ausgabe für einzelne Graphen beeinflusst. Vermutet wurde eine Bevorzugung der Diagonalen in der Unendlich-Norm. Als Vergleich dient die jeweils letzte Lösung der Eingabe *Köln-München* im Grundmodell, einmal in der Unendlich-Norm (Abbildung 5.19) und einmal in der Euklidischen Norm (Abbildung 5.20). Man erkennt in der Unendlich-Norm bereits die längere Diagonale.

Tabelle 5.8.: Übersicht über die Variablen, Constraints und Laufzeiten für die Kombination aus Blöcken, Hauptpfad und entfernten Diagonalen

Eingabe	C_{init}	C_{pre}	V_{init}	V_{pre}	erste *(s)*	letzte *(s)*	gap *(%)*
Ko-Mu	2467	828	2080	1584	0	0	77,2
Ba-Pr	3929	1532	3890	3142	0	170	84,7
Fl-Ko	4041	700	4798	3874	-	-	-
Lo-Pa	4347	448	5310	4194	-	-	-
Ka-Tr	4106	1424	4458	3634	0	15	80,6

Während Funktionalitätstests im Rahmen der Implementierung wurde dieser Verdacht weiter bestätigt. Effekte auf das Laufzeitverhalten wurden nicht getestet.

Abbildung 5.19.: Köln-München in der Unendlich-Norm

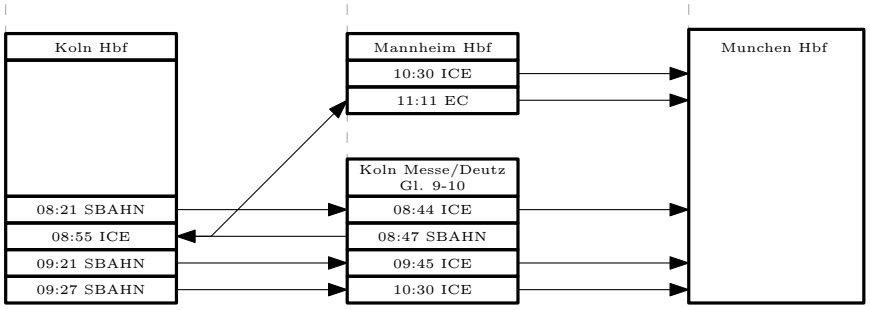
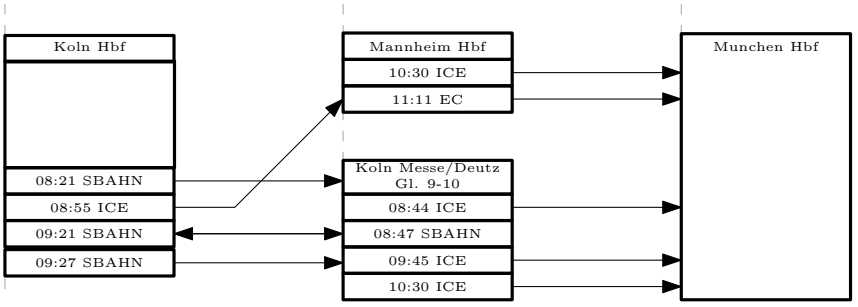


Abbildung 5.20.: Köln-München in der Euklidischen Norm



5.4. Interpretation der Ergebnisse

Um verschiedene Varianten vergleichen zu können, sehen wir uns zunächst die Anzahl an Variablen und Constraints an, die vom Presolve-Algorithmus von vorne herein entfernt werden. Die unten stehenden Tabellen 5.9 und 5.10 geben an, wieviel Prozent der angelegten Variablen und Constraints nach dem Presolve-Algorithmus noch berechnet werden müssen.

Wir sehen, dass im Grundmodell nur circa rund 20% aller Constraints von vorne herein ausgeschlossen werden. Für die Hauptpfad-Variante schwanken die aussortierten Constraints stark, je nach Eingabe. Ein Extrembeispiel ist die Trivialeingabe, dort fallen fast alle Constraints weg. Entfernt man die Diagonalen, so kann der Presolve-Algorithmus in unseren Experimenten stets über die Hälfte aller Constraints ausschließen. Dies entspricht dem erwarteten Ergebnis, da die Hälfte der Richtungen wegfallen und die Mehrheit der Constraints für die Kantenmodellierung genutzt wird. Die Blockbildung hat in unseren Experimenten nicht den gewünschten Effekt. Sie erzielt in Bezug auf den Presolve-Algorithmus ähnliche Ergebnisse wie das Grundmodell.

Tabelle 5.9.: Übersicht über die Anzahl der Variablen und Constraints für die Einzelvariationen nach dem Presolve-Algorithmus in Prozent

Eingabe	Grundmodell		Hauptpfad		ohne Diagonalen		Blöcke	
	C	V	C	V	C	V	C	V
Trivial	80,63	84,02	3,16	51,03	39,47	71,13	keine Blöcke	
Ko-Mu	86,90	90,78	65,58	84,96	43,49	81,09	85,96	90,43
Ba-Pr	86,84	92,15	72,16	88,52	44,11	83,76	86,70	91,96
Fl-Ko	80,53	92,83	37,70	81,84	47,74	87,04	79,43	92,39
Lo-Pa	78,23	89,57	25,61	82,56	47,65	87,67	76,97	92,41
Ba-Mo	76,76	93,66	50,31	89,08	46,28	89,28	keine Blöcke	
Ka-Tr	86,10	93,42	74,00	90,79	47,83	86,05	84,79	92,39

Tabelle 5.10.: Übersicht über die Anzahl der Variablen und Constraints für die Einzelvariationen nach dem Presolve-Algorithmus in Prozent

Eingabe	Hauptpfad, ohne Diagonalen		Blöcke, Hauptpfad, ohne Diagonalen	
	C	V	C	V
Trivial	1,55	50,50	keine Blöcke	
Ko-Mu	36,25	77,21	33,56	76,15
Ba-Pr	40,36	81,25	38,99	80,77
Fl-Ko	21,49	81,03	17,32	78,98
Lo-Pa	14,53	80,55	10,31	78,98
Ba-Mo	30,57	85,97	keine Blöcke	
Ka-Tr	38,77	84,09	34,68	81,52

Die Kombination aus Hauptpfad und Entfernen der Diagonalen erzielt sehr gute Ergebnisse bezüglich der überschüssigen Constraints. Auch hier werden stets mehr als die Hälfte aussortiert. Dies liegt vermutlich wieder an den entfernten Diagonalen. Die Werte sind jedoch stets etwas besser als die reine Variante ohne Diagonalen. Dies liegt dann an der Festlegung einzelner Variablen durch die Hauptpfad-Variante. Vergleicht man die zweite Kombination mit der ersten, so stellt man fest, dass die Werte noch ein klein wenig verbessert werden. Dies liegt offensichtlich an der kleineren Eingabeinstanz.

Betrachten wir nun die gemessenen Laufzeiten. Tabelle 5.11 zeigt nochmals eine Übersicht über die einzelnen Varianten. Tabelle 5.12 zeigt die Laufzeiten der Kombinationen in der Übersicht. Offensichtlich kann für das Grundmodell nur für sehr kleine Eingaben eine Lösung gefunden werden. Für die Hauptpfad-Variante gilt dasselbe. Hier ist ersichtlich, dass die Berechnung für kleine Eingaben deutlich schneller wird. Die Berechnung für größere Eingaben wurde vermutlich auch beschleunigt. Um diese Variante allgemein als gute Lösung zu bezeichnen, fehlen jedoch Lösungen für mittlere und große Eingaben. Auch die Reduzierung der Kanten durch Blöcke führt nicht die gewünschte Beschleunigung herbei.

Werden die Diagonalen entfernt, so wurde in dem Experiment der Einzelvariante ohne Diagonalen für jede Eingabe eine Lösung gefunden. Interessant ist, dass der ILP-Löser für die größte Instanz mit Zehntausend Variablen am längsten gebraucht hat, um die erste Lösung zu finden, wie es erwartet war. Jedoch konnte der ILP-Löser im Gegensatz zu kleineren Eingaben nach einer gewissen Zeitspanne keine weiteren Lösungen für die größte Testinstanz mehr finden. Eine mögliche Erklärung könnte sein, dass größere Eingaben dem ILP-Löser nicht genug Möglichkeiten bieten, einzelne Stationen unterschiedlich zu platzieren. Kleinere Eingaben könnten dagegen mehrere Platzierungsmöglichkeiten bieten

Tabelle 5.11.: Übersicht über die Laufzeiten der einzelnen Varianten

Eingabe	Grundmodell		Hauptpfad		ohneDiagonalen		Blöcke	
	erste	letzte	erste	letzte	erste	letzte	erste	letzte
Trivial	0	0	0	0	0	0	keine Blöcke	
Ko-Mu	580	652	55	73	0	10	0	27
Ba-Pr	-	-	-	-	4	155	-	-
Fl-Ko	-	-	-	-	5	2606	-	-
Lo-Pa	-	-	-	-	10	185	-	-
Ba-Mo	-	-	-	-	160	1005	keine Blöcke	
Ka-Tr	-	-	-	-	380	800	-	-

und daher mehr verschiedene Lösungen produzieren. Allgemein kann man jedoch sagen, dass die erhaltenen Lösungen der Variante ohne Diagonalen sehr gut sind. Die Diagonalen weg zu lassen, scheint also eine gute Verbesserung zu sein.

Tabelle 5.12.: Übersicht über die Laufzeiten der Kombinationen

Eingabe	Kombo Hauptpfad, ohne Diagonalen		Kombo Blöcke, Hauptpfad, ohne Diagonalen	
	erste	letzte	erste	letzte
Trivial	0	0	keine Blöcke	
Ko-Mu	0	1	0	0
Ba-Pr	0	15	0	170
Fl-Ko	-	-	-	-
Lo-Pa	-	-	-	-
Ba-Mo	20	2681	keine Blöcke	
Ka-Tr	15	75	0	15

Fügt man der Variante ohne Diagonalen nun noch die Hauptpfad-Variante hinzu, so können für bestimmte Eingaben wiederum keine Lösungen generiert werden. Eine denkbare Erklärung hierfür ist, dass die Hauptpfad-Variante den Canvas vergrößert und dadurch zu viele Platzierungsmöglichkeiten bestehen. Um dies zu verifizieren müssten weitere Tests mit verschiedenen Hauptpfad-Längen durchgeführt werden. Dazu war im Rahmen dieser Arbeit jedoch keine Zeit vorhanden. Auch die Dreier-Kombination aus Hauptpfad-Variante, Blocken und Entfernen von Diagonalen brachte keine Lösungen für die entsprechenden Eingabeinstanzen. Die Berechnung der Eingabeinstanz *Karlsruhe-Trier* wurde beschleunigt. Dies liegt daran, dass in diesem Graph im Gegensatz zu den anderen Eingabegraphen sehr viele Blöcke gebildet werden konnten und die Anzahl der Zugkanten dadurch stark verringert wurde.

Kommen wir abschließend zum Thema Optimalität. Wie bereits oben erwähnt, konnte für keine der Eingaben außer der Trivialeingabe und für keine Modellvariante die für den ILP-Löser eindeutig optimale Lösung berechnet werden. Diese rät der Löser zunächst als heuristische Lösung, d.h. als untere Schranke. Die Anzeige *gap* soll den Benutzer darüber informieren, wie weit die zur Zeit beste Lösung, d.h. die obere Schranke, noch von der heuristischen Lösung entfernt ist. Das heißt für $gap = 0\%$ ist die gefundene Lösung minimal in Bezug auf das angegebene Modell. Der Wert $gap = 100\%$ bedeutet also genau das Gegenteil, nämlich dass die aktuelle Lösung noch weit von der heuristischen Lösung entfernt ist. Dazu sei gesagt, dass die heuristische Lösung des ILP-Lösers in den Tests oft sehr unrealistische Werte annahm. Die minimale Kantenlänge zwischen zwei nebeneinander

liegenden Stationen beträgt in den obigen Lösungen 64 pts. Da alle Kantenlängen in die Minimierung mit eingehen, beträgt die minimale heuristische Lösung also mindestens

$$64 \cdot m$$

Hinzu kommen noch Kantenknicke und Stationshöhen. Der ILP-Löser konnte diesen Wert jedoch nie annähernd bestimmen. Die Anzeige des *gap* verliert also stark an Bedeutung und kann kaum in eine Bewertung der gefundenen Lösungen mit einbezogen werden.

6. Fazit

Fassen wir nun abschließend noch einmal zusammen. Die Motivation für diese Arbeit war, dass die meisten Algorithmen der klassischen Graphenvisualisierung in zu unflexibel für die Visualisierung des Abfahrtsgraphen sind, oder wir können sie nicht so auf unseren Abfahrtsgraphen anwenden, um das von uns gewünschte Ergebnis zu erzielen. Wir wollten versuchen, diese Flexibilität durch die Modellierung eines Graphen und seiner Darstellung als ILP-Modell zu erhalten. Dazu kombinieren wir die klassische Graphenvisualisierung mit der Optimierungstheorie.

Im Rahmen der Arbeit wurde festgestellt, dass die Modellierungsmöglichkeiten eines ganzzahligen linearen Programms uns genug Freiheit im Design lassen, um unseren Anforderungen gerecht zu werden. Wir können Stationen mit ihren Stationsknoten und Abfahrtsknoten als Boxen zeichnen. Wir können diese auf der Zeichenfläche anordnen und verschieben. Wir können sie beliebig groß machen oder die Höhe der Stationen minimieren. Die Kanten zwischen den Stationen können wir als Polylinien zeichnen. Wir können beliebige Kantenknicke festlegen, ob dies nun 45°- oder 90°-Winkel sind. Außerdem können wir die Kanten beliebig oft knicken lassen, indem wir Dummy-Knoten einfügen oder wir können Knicke minimieren. Des Weiteren sind octolineare Richtungen möglich, die wir bei Bedarf auch auf ein orthogonales System reduzieren können, indem wir die Diagonalen entfernen. Wir können den Kanten die Freiheit geben, rechts oder links aus ihrem Abfahrtsknoten auszutreten, genauso wie sie an beiden Seiten in ihre Zielstation einlaufen können. Wir haben gelernt, dass wir, wenn wir die Kanten geschickt wählen, auch nicht triviale, nicht lineare Funktionen wie die Euklidische oder die Unendlich-Norm als ILP modellieren können. Der große Vorteil ist, dass wir ganze Mengen von Constraints jederzeit an- und ausschalten können und das Modell dadurch sehr leicht variieren können. Wir erhalten also ein deutlich größeres Maß an Flexibilität.

Doch ist es sinnvoll, diesen Ansatz andern algorithmischen Ansätzen vorzuziehen? Was müssen wir für diese Flexibilität aufgeben? In den Laufzeitexperimenten haben wir gesehen, dass die Berechnung eines Ausgabebildes für ein einfaches Grundmodell für mittlere und große Eingaben sehr langsam ist. Besonders die Kreuzungsfreiheit von Kanten mit Stationen erhöht die Rechenzeit beachtlich. Dabei ist die Kreuzungs- oder Überlagerungsfreiheit von Kanten untereinander noch nicht modelliert. Callback-Funktionen und Lazy-Constraints erweisen sich als deutliche Verbesserung. Sie sind für unser Modell ein Muss.

Versuchen wir unser Modell zu optimieren, so stellen wir schnell fest, dass die Octolinearität der Kanten ein Hindernis für eine schnelle Laufzeitberechnung ist. Lassen wir die

Diagonalen weg und reduzieren die Richtungen somit von acht auf vier, bekommen wir plötzlich Lösungen für all unsere Eingaben. Somit sind wir wieder bei der orthogonalen Graphenvisualisierung. Dieses Teilgebiet ist jedoch schon gut erforscht und es gibt bessere und schnellere Algorithmen mit einer besseren Darstellung eines orthogonalen Graphen als das in dieser Arbeit vorgestellte Modell. Es lohnt sich also nicht, diesen Ansatz für rein orthogonale Graphen zu verwenden.

Ist das Modell verbesserungsfähig? Sicherlich gibt es noch einige Varianten, die zu testen sich lohnen würde. Das Modell ist insgesamt sehr intuitiv aufgebaut. Vielleicht ist es möglich, die Berechnung noch zu beschleunigen, indem man Kantengruppen genauer bestimmt und einschränkt. Auch denkbar wäre, die Grundidee der Lagenlayouts einzuarbeiten und eine Hierarchie für die Stationen festzulegen. So könnte man die Koordinaten womöglich weiter einschränken. Die Hauptpfad-Variante geht schon in diese Richtung, sie reicht aber offensichtlich nicht aus.

Bezüglich realer Anwendungen für den Abfahrtsgraph ist dieser Ansatz zur Zeit mit dem aktuellen Modell wohl auch wenig sinnvoll. Ein Benutzer, der wissen möchte, welchen Zug er als nächstes nehmen sollte, wenn er einen Anschlusszug verpasst, möchte nicht eine Stunde oder länger auf seine Antwort warten. Es ist im Moment nicht möglich, Ausgaben für beliebige Eingaben effizient und innerhalb einer benutzerfreundlichen Wartezeit zu berechnen. Für dynamische Benutzeranfragen an den Abfahrtsgraph ist die ILP-Modellierung also die falsche Vorgehensweise. Eine mögliche Anwendung besteht vielleicht für statische Anfragen. Denkbar wäre eine statische Vorberechnung für oft benutzte Strecken, z.B. zwischen Großstädten, sodass nur noch die kleineren Zwischenstationen berechnet werden müssten.

Literaturverzeichnis

[DBETT98] Giuseppe Di Battista, Peter Eades, Roberto Tamassia und Ioannis G. Tollis: *Graph Drawing - Algorithms for the Visualization of Graphs.* Prentice Hall, 1998, ISBN 0133016153.

[DH96] Ron Davidson und David Harel: *Drawing Graphs Nicely using Simulated Annealing.* ACM Transactions on Graphics, 15(4):301–331, 1996.

[DSW14] Julian Dibbelt, Ben Strasser und Dorothea Wagner: *Delay-Robust Journeys in Timetable Networks with Minimum Expected Arrival Time.* In: *Proceedings of the 14th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'14)*, OpenAccess Series in Informatics (OASIcs), 2014.

[Ead84] Peter Eades: *A Heuristic for Graph Drawing.* Congressus Numerantium, 42:149–160, 1984.

[EFK01] Markus Eiglsperger, Sándor P. Fekete und Gunnar W. Klau: *Orthogonal Graph Drawing.* In: Michael Kaufmann und Dorothea Wagner (Herausgeber): *Drawing Graphs: Methods and Models*, Band 2025 der Reihe *Lecture Notes in Computer Science*, Seiten 121–171. Springer, 2001, ISBN 3-540-42062-2. `http://springerlink.metapress.com/content/kbnwla90v8vk7rnk/`.

[ES90] Peter Eades und Kozo Sugiyama: *How to Draw a Directed Graph.* Journal of Information Processing, 13(4):424–437, 1990.

[GJ79] Michael R. Garey und David S. Johnson: *Computers and Intractability. A Guide to the Theory of $\mathcal{NP}$-Completeness.* W. H. Freeman and Company, 1979.

[Nöl05] Martin Nöllenburg: *Automated Drawing of Metro Maps.* Diplomarbeit, Universität Karlsruhe, August 2005.

[STT81] Kozo Sugiyama, S. Tagawa und M. Toda: *Methods for Visual Understanding of Hierarchical System Structures.* IEEE Transactions on Systems, Man and Cybernetics, SMC-11(2):109–125, 1981. `http://dx.doi.org/10.1109/TSMC.1981.4308636`.

[Tam13] Roberto Tamassia (Herausgeber): *Handbook of Graph Drawing and Visualization.* CRC Press, 2013.

Anhang

A. Auflistung der ILP-Gleichungen

Dieser Abschnitt enthält eine Auflistung aller in das ILP eingefügten Gleichungen für das Grundmodell. Nicht mit aufgelistet sind Zahlenbereiche und Konstanten.

A.1. Stationen

$$
\begin{aligned}
&x(v_1) = T \\
&\forall v_i \neq v_n : \; x(v_i) \leq x(v_n) \\
&\forall v_i : x(v_i) = \phi_i * Gw + T \\
&\forall v_i : x(v_i) + B \leq Cw \\
&\forall v_i : y(v_i) + H \leq Ch
\end{aligned}
$$

$$
\begin{aligned}
&0 \leq a_{i_j} \leq hvar_i \\
&y(v_i) = \; y(v_{i_1}) + H + hvar_i - a_{i_1} + b_{i_1} \\
&y(v_{i_j}) = \; y(v_{i_{j+1}}) + H + hvar_i - a_{i_{j+1}} + b_{i_{j+1}}
\end{aligned}
$$

$$
\begin{aligned}
&\psi(i,u)_0 + \psi(i,u)_1 + \psi(i,u)_2 && +\psi(i,u)_3 = 1 \\
&y(v_u) + H + Gap + 1 - y(v_{i_{max}}) && \leq M \cdot (1 - \psi(i,u)_0) \\
&x(v_u) - x(v_i) && \leq M \cdot (1 - \psi(i,u)_0) \\
&x(v_i) - x(v_u) && \leq M \cdot (1 - \psi(i,u)_0) \\
&y(v_i) + H + Gap + 1 - y(v_{u_{max}}) && \leq M \cdot (1 - \psi(i,u)_1) \\
&x(v_u) - x(v_i) && \leq M \cdot (1 - \psi(i,u)_1) \\
&x(v_i) - x(v_u) && \leq M \cdot (1 - \psi(i,u)_1) \\
&x(v_u) + 1 - x(v_i) && \leq M \cdot (1 - \psi(i,u)_2) \\
&x(v_i) + 1 - x(v_u) && \leq M \cdot (1 - \psi(i,u)_3) \\
&\forall u, i : u \neq i : && \psi(u,i)_0 = \psi(i,u)_1 \\
&\forall u, i : u \neq i : && \psi(u,i)_2 = \psi(i,u)_3
\end{aligned}
$$

A.2. Zugankunft und -abfahrt

$$
\begin{aligned}
&x_1(e_{i_j,u}) - x(v_i) && \leq M \cdot (1 - \alpha(e_{i_j,u})) \\
&x(v_i) - x_1(e_{i_j,u}) && \leq M \cdot (1 - \alpha(e_{i_j,u})) \\
&x_1(e_{i_j,u}) - (x(v_i) + B) && \leq M \cdot \alpha(e_{i_j,u}) \\
&x(v_i) + B - x_1(e_{i_j,u}) && \leq M \cdot \alpha(e_{i_j,u}) \\
&y_1(e_{i_j,u}) && = y(v_{i_j}) + 1/2 \cdot H
\end{aligned}
$$

$$
\begin{aligned}
& y_2(e_{i_j,u}) \geq y(v_{u_{max}}) && \\
& y_2(e_{i_j,u}) \leq y(v_u) + H && \\
& x_2(e_{i_j,u}) - x(v_u) && \leq M \cdot (1 - \beta(e_{i_j,u})) \\
& x(v_u) - x_2(e_{i_j,u}) && \leq M \cdot (1 - \beta(e_{i_j,u})) \\
& x_2(e_{i_j,u}) - (x(v_u) + B) && \leq M \cdot \beta(e_{i_j,u}) \\
& x(v_u) + B - x_2(e_{i_j,u}) && \leq M \cdot \beta(e_{i_j,u})
\end{aligned}
$$

$$
\begin{aligned}
& r_{i_j,u} = \sum_q r_q(e_{i_j,u}) \\
& -z_q(e_{i_j,u}) \leq r_q(e_{i_j,u}) \\
& r_q(e_{i_j,u}) \leq z_q(e_{i_j,u})
\end{aligned}
$$

$$
\begin{aligned}
& 3 - s_1(e_{i_j,u}) && \leq M \cdot (1 - \alpha(e_{i_j,u})) \\
& s_1(e_{i_j,u}) - 5 && \leq M \cdot (1 - \alpha(e_{i_j,u})) \\
& s_1(e_{i_j,u}) - 1 && \leq M \cdot \alpha(e_{i_j,u}) \\
& -1 - s_1(e_{i_j,u}) && \leq M \cdot \alpha(e_{i_j,u}) \\
& s_l(e_{i_j,u}) - 1 && \leq M \cdot (1 - \beta(e_{i_j,u})) \\
& -1 - s_l(e_{i_j,u}) && \leq M \cdot (1 - \beta(e_{i_j,u})) \\
& 3 - s_l(e_{i_j,u}) && \leq M \cdot \beta(e_{i_j,u}) \\
& s_l(e_{i_j,u}) - 5 && \leq M \cdot \beta(e_{i_j,u}) \\
& s_{q+1}(e_{i_j,u}) = s_q(e_{i_j,u}) + r_{q+1}(e_{i_j,u}) && \\
& s_0(e_{i_j,u}) = \alpha(e_{i_j,u}) \cdot 4 && \\
& s_l(e_{i_j,u}) = \sum_q r_q(e_{i_j,u}) + s_0(e_{i_j,u}) &&
\end{aligned}
$$

A.3. Koordinatenverknüpfungen

Eine Vollständige Auflistung aller Koordinatenveränderungen verknüpft mit den für die acht möglichen Richtungen stehenden Binärvariablen. Jede Variable bezieht sich auf ihre Zugkante $e_{i_j,u}$.

$$
\begin{aligned}
& \sum_p \gamma_{q,p} = 1 \\
& s_q = \sum_{p \in \{0..7\}} \gamma_{q,p} * p + \delta_q * 8
\end{aligned}
$$

$$
\begin{aligned}
& x(d_1) = x_1(e) && \text{//setzt den ersten "'Dummy"' als Startpunkt} \\
& y(d_1) = y_1(e) && \\
& x(d_{Q+2}) = x_2(e) && \text{//setzt den letzten "'Dummy"' als Endpunkt} \\
& y(d_{Q+2}) = y_2(e) &&
\end{aligned}
$$

$$
\begin{aligned}
x(d_q) + 1 - x(d_{q+1}) &\leq M(1-\gamma_{q,0}) \text{ //erzeugt } x(d_q) < x(d_{q+1}) \\
y(d_q) - y(d_{q+1}) &\leq M(1-\gamma_{q,0}) \text{ //erzeugt } y(d_q) = y(d_{q+1}) \\
y(d_{q+1}) - y(d_q) &\leq M(1-\gamma_{q,0}) \\
(x(d_q) + y(d_q) + 1) - (x(d_{q+1}) + y(d_{q+1})) &\leq M(1-\gamma_{q,1}) \text{ //erzeugt } z_1(d_q) < z_1(d_{q+1}) \\
(x(d_q) - y(d_q)) - (x(d_{q+1}) - y(d_{q+1})) &\leq M(1-\gamma_{q,1}) \text{ //erzeugt } z_2(d_q) = z_2(d_{q+1}) \\
(x(d_{q+1}) - y(d_{q+1})) - (x(d_q) - y(d_q)) &\leq M(1-\gamma_{q,1})
\end{aligned}
$$

$$
\begin{aligned}
y(d_q) + 1 - y(d_{q+1}) &\leq M(1-\gamma_{q,2}) \text{ //erzeugt } y(d_q) < y(d_{q+1}) \\
x(d_q) - x(d_{q+1}) &\leq M(1-\gamma_{q,2}) \text{ //erzeugt } x(d_q) = x(d_{q+1}) \\
x(d_{q+1}) - x(d_q) &\leq M(1-\gamma_{q,2})
\end{aligned}
$$

$$
\begin{aligned}
(x(d_{q+1}) - y(d_{q+1}) + 1) - (x(d_q) - y(d_q)) &\leq M(1-\gamma_{q,3}) \text{ //erzeugt } z_2(d_{q+1}) < z_2(d_q) \\
(x(d_q) + y(d_q)) - (x(d_{q+1}) + y(d_{q+1})) &\leq M(1-\gamma_{q,3}) \text{ //erzeugt } z_1(d_q) = z_1(d_{q+1}) \\
(x(d_{q+1}) + y(d_{q+1})) - (x(d_q) + y(d_q)) &\leq M(1-\gamma_{q,3})
\end{aligned}
$$

$$
\begin{aligned}
x(d_{q+1}) + 1 - x(d_q) &\leq M(1-\gamma_{q,4}) \text{ //erzeugt } x(d_{q+1}) < x(d_q) \\
y(d_q) - y(d_{q+1}) &\leq M(1-\gamma_{q,4}) \text{ //erzeugt } y(d_q) = y(d_{q+1}) \\
y(d_{q+1}) - y(d_q) &\leq M(1-\gamma_{q,4})
\end{aligned}
$$

$$
\begin{aligned}
(x(d_{q+1}) + y(d_{q+1}) + 1) - (x(d_q) + y(d_q)) &\leq M(1-\gamma_{q,5}) \text{ //erzeugt } z_1(d_{q+1}) < z_1(d_q) \\
(x(d_q) - y(d_q)) - (x(d_{q+1}) - y(d_{q+1})) &\leq M(1-\gamma_{q,5}) \text{ //erzeugt } z_2(d_q) = z_2(d_{q+1}) \\
(x(d_{q+1}) - y(d_{q+1})) - (x(d_q) - y(d_q)) &\leq M(1-\gamma_{q,5})
\end{aligned}
$$

$$
\begin{aligned}
y(d_{q+1}) + 1 - y(d_q) &\leq M(1-\gamma_{q,6}) \text{ //erzeugt } y(d_{q+1}) < y(d_q) \\
x(d_q) - x(d_{q+1}) &\leq M(1-\gamma_{q,6}) \text{ //erzeugt } x(d_q) = x(d_{q+1}) \\
x(d_{q+1}) - x(d_q) &\leq M(1-\gamma_{q,6})
\end{aligned}
$$

$$
\begin{aligned}
(x(d_q) - y(d_q) + 1) - (x(d_{q+1}) - y(d_{q+1})) &\leq M(1-\gamma_{q,7}) \text{ //erzeugt } z_2(d_q) < z_2(d_{q+1}) \\
(x(d_q) + y(d_q)) - (x(d_{q+1}) + y(d_{q+1})) &\leq M(1-\gamma_{q,7}) \text{ //erzeugt } z_1(d_q) = z_1(d_{q+1}) \\
(x(d_{q+1}) + y(d_{q+1})) - (x(d_q) + y(d_q)) &\leq M(1-\gamma_{q,7})
\end{aligned}
$$

A.4. Längenberechnung

Alle Variablen beziehen sich auf ihre Zugkante $e_{i_j,u}$.
Berechnung der Unendlich-Norm:

$$
\begin{aligned}
-l_{x,q} &\leq x(d_q) - x(d_{q+1}) \\
x(d_q) - x(d_{q+1}) &\leq l_{x,q} \\
-l_{y,q} &\leq y(d_q) - y(d_{q+1}) \\
y(d_q) - y(d_{q+1}) &\leq l_{y,q} \\
l_{y,q} &\leq m_q \\
l_{x,q} &\leq m_q \\
l &= \sum_q m_q
\end{aligned}
$$

Berechnung der euklidischen Längen für alle acht möglichen Richtungen:

$$
\begin{aligned}
x(d_{q+1}) - x(d_q) - m_q &\leq M \cdot (1 - \gamma_{q,0}) \\
m_q - (x(d_{q+1}) - x(d_q)) &\leq M \cdot (1 - \gamma_{q,0}) \\
\sqrt{2} \cdot x(d_{q+1}) - \sqrt{2} \cdot x(d_q) - m_q &\leq M \cdot (1 - \gamma_{q,1}) \\
m_q - (\sqrt{2} \cdot x(d_{q+1}) - \sqrt{2} \cdot x(d_q)) &\leq M \cdot (1 - \gamma_{q,1}) \\
y(d_{q+1}) - y(d_q) - m_q &\leq M \cdot (1 - \gamma_{q,2}) \\
m_q - (y(d_{q+1} - y(d_q)) &\leq M \cdot (1 - \gamma_{q,2}) \\
\sqrt{2} \cdot x(d_q) - \sqrt{2} \cdot x(d_{q+1}) - m_q &\leq M \cdot (1 - \gamma_{q,3}) \\
m_q - (\sqrt{2} \cdot x(d_q) - \sqrt{2} \cdot x(d_{q+1})) &\leq M \cdot (1 - \gamma_{q,3}) \\
x(d_q) - x(d_{q+1}) - m_q &\leq M \cdot (1 - \gamma_{q,4}) \\
m_q - (x(d_q) - x(d_{q+1})) &\leq M \cdot (1 - \gamma_{q,4}) \\
\sqrt{2} \cdot x(d_q) - \sqrt{2} \cdot x(d_{q+1}) - m_q &\leq M \cdot (1 - \gamma_{q,5}) \\
m_q - (\sqrt{2} \cdot x(d_q) - \sqrt{2} \cdot x(d_{q+1})) &\leq M \cdot (1 - \gamma_{q,5}) \\
y(d_q) - y(d_{q+1}) - m_q &\leq M \cdot (1 - \gamma_{q,6}) \\
m_q - (y(d_q) - y(d_{q+1})) &\leq M \cdot (1 - \gamma_{q,6}) \\
\sqrt{2} \cdot x(d_{q+1}) - \sqrt{2} \cdot x(d_q) - m_q &\leq M \cdot (1 - \gamma_{q,7}) \\
m_q - (\sqrt{2} \cdot x(d_{q+1}) - \sqrt{2} \cdot x(d_q)) &\leq M \cdot (1 - \gamma_{q,7}) \\
l(e_{i_j,u}) = \sum_q m_q(e_{i_j,u})
\end{aligned}
$$

A.5. Sperrflächen

Gleichungen für die (Lazy-) Constraints für die Kreuzungsfreiheit von Kante $e_{i_j,u}$ und Station k. Die Halbebenen-Variablen tragen stets die Indices i_j, k. Koordinaten der Dummyknoten beziehen sich auf die Zugkante.

$$
\begin{aligned}
N + NO + O + SO + S + SW + W + NW &\geq 1 \\
y(v_k) + H + \epsilon - y(d_q) &\leq M \cdot (1 - N) \\
y(v_k) + H + \epsilon - y(d_{q+1}) &\leq M \cdot (1 - N) \\
(x(v_k) + B) + (y(v_k) + H) + \epsilon - (x(d_q) + y(d_q)) &\leq M \cdot (1 - NO) \\
(x(v_k) + B) + (y(v_k) + H) + \epsilon - (x(d_{q+1}) + y(d_{q+1})) &\leq M \cdot (1 - NO) \\
x(v_k) + B + \epsilon - x(d_q) &\leq M \cdot (1 - O) \\
x(v_k) + B + \epsilon - x(d_{q+1}) &\leq M \cdot (1 - O) \\
(x(v_k) + B) - y(v_{k_{max}}) + \epsilon - (x(d_q) - y(d_q)) &\leq M \cdot (1 - SO) \\
(x(v_k) + B) - y(v_{k_{max}}) + \epsilon - (x(d_{q+1}) - y(d_{q+1})) &\leq M \cdot (1 - SO) \\
y(d_q) + \epsilon - y(v_{k_{max}}) &\leq M \cdot (1 - S) \\
y(d_{q+1}) + \epsilon - y(v_{k_{max}}) &\leq M \cdot (1 - S) \\
(x(d_q) + y(d_q)) + \epsilon - (x(v_k) + y(v_{k_{max}})) &\leq M \cdot (1 - SW) \\
(x(d_{q+1}) + y(d_{q+1})) + \epsilon - (x(v_k) + y(v_{k_{max}})) &\leq M \cdot (1 - SW) \\
x(d_q) + \epsilon - x(v_k) &\leq M \cdot (1 - W) \\
x(d_{q+1}) + \epsilon - x(v_k) &\leq M \cdot (1 - W) \\
(x(d_q) - y(d_q)) + \epsilon - (x(v_k) - y(v_k) - H) &\leq M \cdot (1 - NW) \\
(x(d_{q+1}) - y(d_{q+1})) + \epsilon - (x(v_k) - y(v_k) - H) &\leq M \cdot (1 - NW)
\end{aligned}
$$

B. Beipiellösungen

In diesem Abschnitt finden sich weitere Beispiellösungen für die in Kapitel 5.3 getesteten Eingaben. Für das Grundmodell, für die Hauptpfad-Variante und für die Block-Variante gibt es die Lösungsbilder bereits innerhalb der Arbeit, da für die größeren Eingaben keine Lösung gefunden wurden. Es folgen Lösungsausgaben für die anderen Varianten, sofern vorhanden.

B.1. Variante ohne Diagonalen

Abbildung B.1.: Basel-Prenzlau, ohne Hauptpfad, ohne Diagonalen, erste Lösung

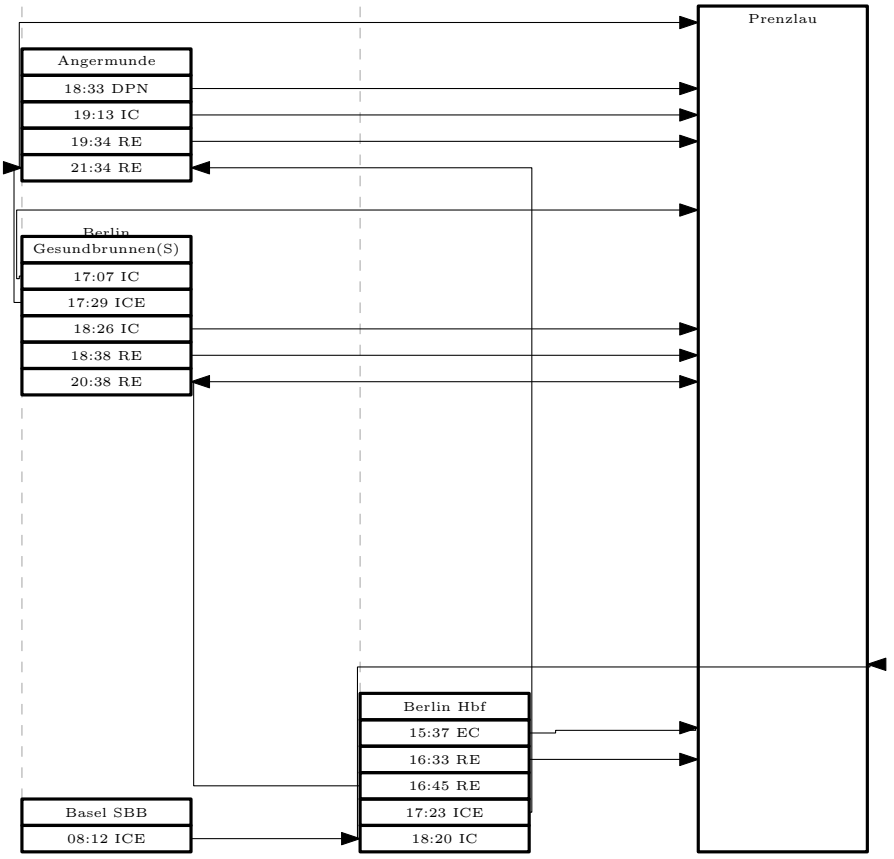


Abbildung B.2.: Basel-Prenzlau, ohne Hauptpfad, ohne Diagonalen, letzte Lösung

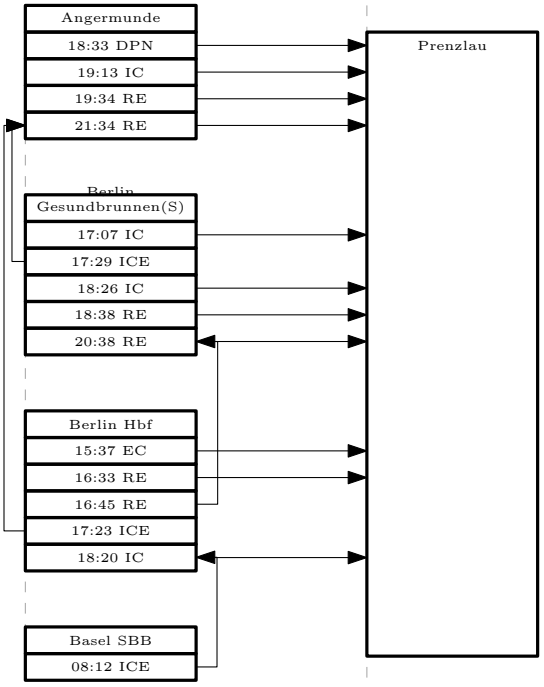


Abbildung B.3.: Bari-Moskau, ohne Hauptpfad, ohne Diagonalen, erste Lösung

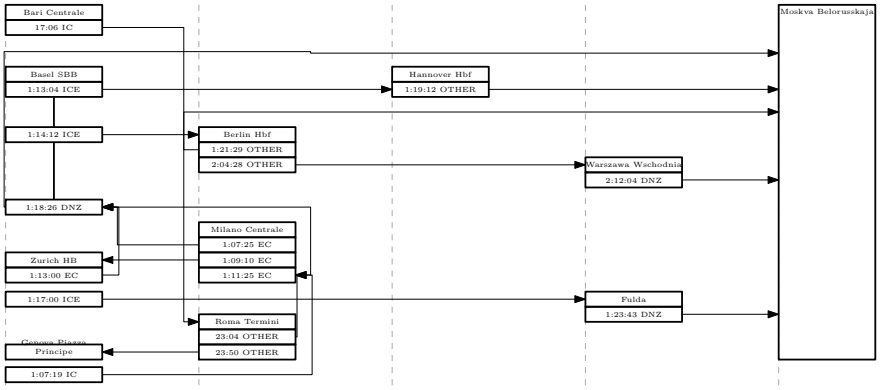


Abbildung B.4.: Bari-Moskau, ohne Hauptpfad, ohne Diagonalen, letzte Lösung

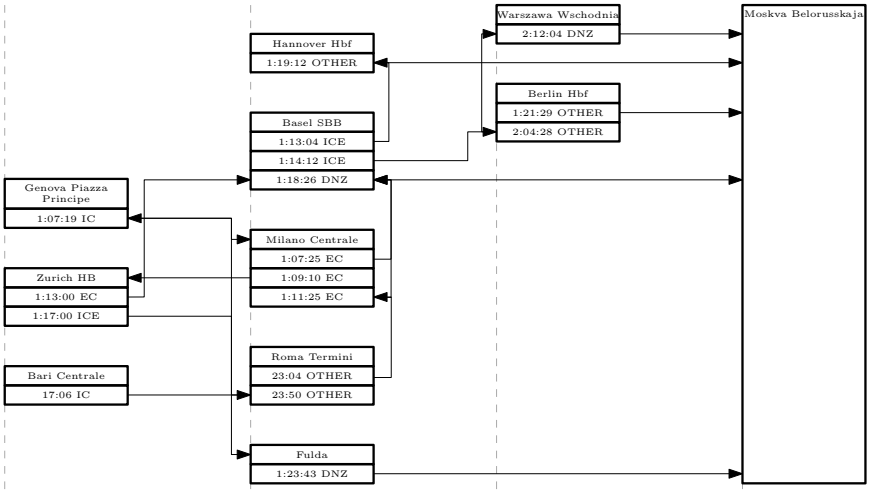


Abbildung B.5.: Karlsruhe-Trier, ohne Hauptpfad, ohne Diagonalen, erste Lösung

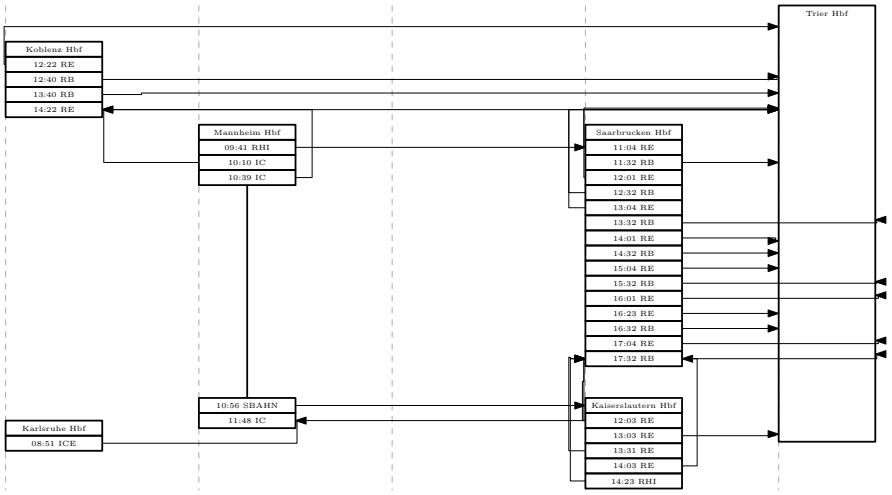
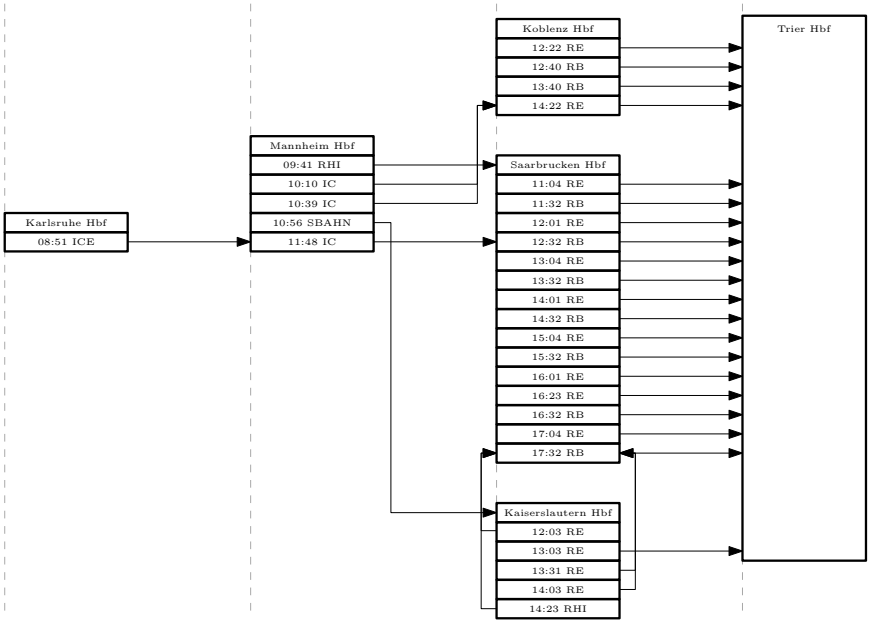


Abbildung B.6.: Karlsruhe-Trier, ohne Hauptpfad, ohne Diagonalen, letzte Lösung



B.2. Blöcke, Hauptpfad, ohne Diagonalen

Abbildung B.7.: Bari-Moskau, Blöcke, Hauptpfad, ohne Diagonalen, erste Lösung

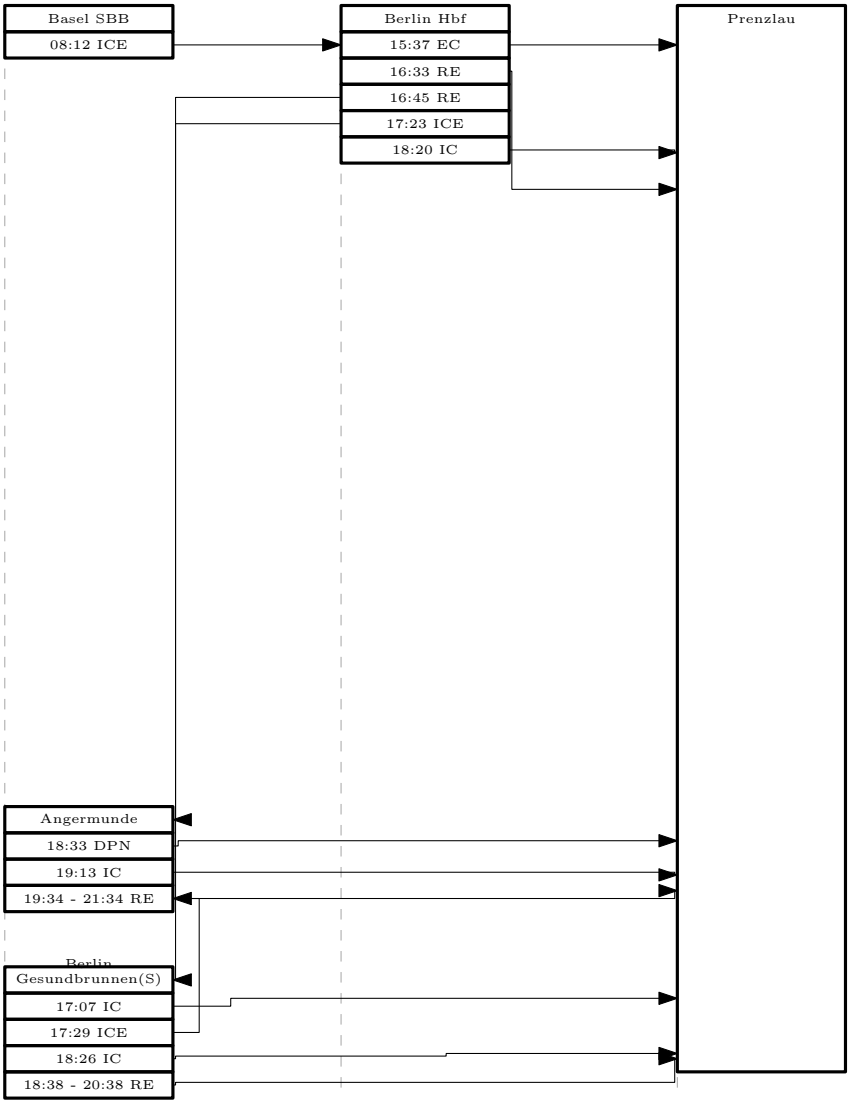


Abbildung B.8.: Bari-Moskau, Blöcke, Hauptpfad, ohne Diagonalen, letzte Lösung

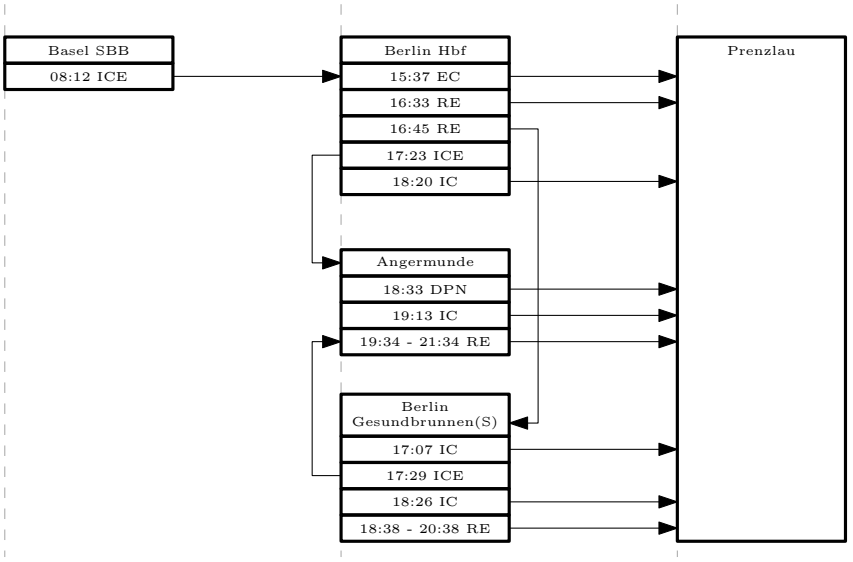


Abbildung B.9.: Karlsruhe-Trier, Blöcke, Hauptpfad, ohne Diagonalen, erste Lösung

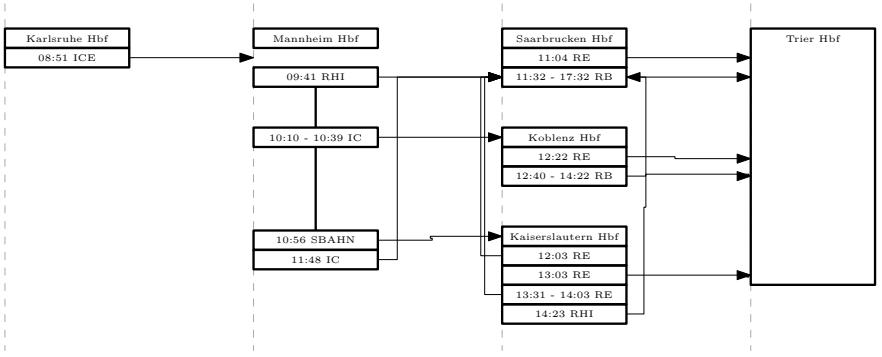


Abbildung B.10.: Karlsruhe-Trier, Blöcke, Hauptpfad, ohne Diagonalen, letzte Lösung