Diplomarbeit

Constraint-basierte Generierung realitätsnaher Eisenbahnnetze

von
Björn Piesker

Universität Potsdam
Institut für Informatik
Professur Constraint-Programmierung

1. Gutachter:	Prof. Dr. Ulrich Geske (Universität Potsdam)
2. Gutachter:	Dr. Armin Wolf (Fraunhofer FIRST, Berlin)

Potsdam
15. Mai 2007

Inhaltsverzeichnis

1 Einleitung

Mit der zunehmenden Integration von rechnergestützten Anwendungen im Eisenbahnwesen, werden heutzutage viele betriebstechnische Aufgaben mit Hilfe von Computern bearbeitet. Unter diesen Aufgaben fallen Fahrplankonstruktion, Disposition, Personalplanung oder Simulationen zur Leistungsuntersuchung. Sowohl in Verkehrsunternehmen als auch in Forschungseinrichtungen werden neue Verfahren entwickelt, welche in Software zum Einsatz kommen sollen. Um die Effizienz der Verfahren zu verifizieren, werden diese Verfahren durch Simulation in verschiedenen Testszenarien überprüft. Ein grundlegender Bestandteil solcher Testszenarien ist die Infrastruktur des Eisenbahnnetzes auf der die Simulationen durchgeführt werden. Da Infrastrukturdaten im Allgemeinen nicht öffentlich zugänglich sind, besteht die Notwendigkeit, realitätsnahe Infrastrukturdaten zu erzeugen. Realitätsnahe Infrastrukturdaten zeichnen sich durch eine recht hohe Komplexität aus. Zudem ist die konkrete Form eines Schienennetzes an vielfältige, teilweise komplexe Randbedingungen geknüpft. Die manuelle Erzeugung von Eisenbahninfrastrukturdaten stellt kein adäquates Mittel dar, da bei der Menge zu erzeugender Daten nicht gewährleistet werden kann, dass keine Inkonsistenzen auftreten bzw. Randbedingungen immer Beachtung finden. Fehlerhafte bzw. unerwünschte Infrastrukturdaten wären die Folge. Neben der Fehleranfälligkeit durch manuelle Erstellung ist der zu betreibende Aufwand für einen Menschen unzumutbar. Aus diesem Grund bietet sich die automatisierte Bearbeitung von Problemen dieser Komplexität an. Ein geeignetes Mittel zur Lösung von Problemen, die durch Randbedingungen beschrieben werden, stellt die Constraint-Programmierung dar. Die Constraint-Programmierung ist ein Programmierparadigma, welches durch spezielle Verfahren und Herangehensweisen effizient Randbedingungen verarbeiten kann und dadurch auf die Lösung derartiger Probleme zugeschnitten ist. Das Ziel dieser Arbeit ist, eine Applikation mit Hilfe der Constraint-Programmierung zu entwickeln, die konsistente und vom Anwender, durch eine Spezifikation, charakterisierte Infrastrukturdaten generiert. Erläuternd dazu werden zunächst die Grundlagen beschrieben, die für die Bearbeitung dieser Aufgabe notwendig sind. Zuerst werden die Prinzipien und die Arbeitsweise der Constraint-Programmierung erläutert. Anschließend werden eisenbahntechnische Begriffe und Verfahren beschrieben, die sich auf die Eisenbahninfrastruktur beziehen bzw. Einfluss auf die Gestaltung der Infrastruktur haben. Im darauf folgenden Kapitel werden die in dieser Arbeit entwickelten Modelle vorgestellt, welche das Fundament der Implementierung bilden. An diesen Modellen werden die Teilprobleme des Generierungsprozesses und ihre Lösungen erläutert. Zudem

wird darauf eingegangen, inwieweit der Anwender in den Generierungsprozess eingreifen und die Eigenschaften des Eisenbahnnetzes durch die Spezifikation kontrollieren kann. Das anschließende Kapitel beschreibt die Implementierung der Modelle, stellt die wesentlichen Algorithmen dar, beschreibt aufgetretene Probleme während der Implementierung und erläutert die verwendeten Datenstrukturen. Danach werden die an der Applikation durchgeführten Tests ausgewertet und die Ergebnisse abschließend zusammengefasst.

2 Grundlagen

2.1 Constraint-Programmierung

Dieser Teil der Arbeit wird einen grundlegenden Einblick in die Constraint-Programmierung geben und notwendige Begriffe einführen, die zum besseren Verständnis dieser Arbeit führen sollen. Dieses Kapitel beschränkt sich nur auf Teilbereiche dieses Fachgebietes und verweist an entsprechendes Stellen auf weiterführende Literatur.

Die Constraint-Programmierung ist ein Programmierparadigma, welches Mitte der achtziger Jahre des 20. Jahrhunderts erstmals vorgestellt und theoretisch begründet wurde. In dieser Zeit wurden erste relevante Programmiersprachen für die Behandlung von Constraints entwickelt. Das wesentliche Merkmal der Entwicklung dieser Sprachen ist die Basierung auf dem logischen Programmierparadigma. Daher kann die Constraint-Programmierung heute als eine natürliche Erweiterung der logischen Programmierung angesehen werden. Aus diesem Grund wird auch oft von der so genannten *Constraint-Logischen Programmierung*(CLP) gesprochen.

Somit wurde die Deklarativität des logischen Programmierung mit der der effizienten Lösungsfindung mittels Constraints vereint. Das Vorhandensein des Backtracking-Mechanismus in logischen Programmiersprachen erleichtert die Programmierung einer Lösungssuche wesentlich. In der constraint-logischen Programmiersprache wird der Zustand der Constraintverarbeitung während des Backtracking zusätzlich zurückgesetzt.

Die constraintbasierte Programmierung ist besonders zum Lösen komplexer Suchprobleme, die durch Constraints formulierbar sind, geeignet. Typische Anwendungsgebiete sind beispielsweise Probleme aus dem Bereich der Unternehmensforschung (engl. operations research) wie Personaleinsatzplanung, Ablaufplanung, Verkehrsplanung und Kapazitätsplanung, in der Sprachverarbeitung die Konstruktion effizienter Parser, und aus dem Bereich Konfiguration, wie Schaltkreisentwurf, Konfiguration technischer Anlagen und Produkte.

Ein *Constraint*(Nebenbedingung, Zwang) bildet eine relationale Beziehung zwischen Variablen und kann eine Gleichung oder Ungleichung sein. Durch sie werden Bedingungen oder Einschränkungen von Objekten oder die Beziehungen zwischen diesen beschrieben. In der Praxis sind Constraints spezielle durch die jeweilige Programmiersprache gegebene prädikatenlogische Formeln. Ein

Constraint welches sich auf genau eine Variable bezieht bezeichnet man als *unär*. Bezieht es sich auf zwei Variablen, wird es *binäres* Constraint genannt.

2.1.1 Constraint-Löser

Die Verarbeitung und Verwaltung von Constraints erfolgt durch einen *Constraint-Löser* (engl. *constraint solver*). Dieser arbeitet während des Programmablaufs als Hintergrundprozess und wacht darüber, dass nur die Lösungen gefunden werden, welche die Constraints einhalten. In der Praxis sind Constraint-Löser Sammlungen über speziellen Klassen von Constraints. Beispielsweise gibt es Constraint-Löser für Boolesche Constraints, lineare arithmetische Constraints oder Finite-Domain-Constraints. Der Grund für diese Differenzierung sind die unterschiedlichen Algorithmen mit denen die Constraints der jeweiligen Klasse behandelt werden. Da die Realisierung der Diplomarbeit ausschließlich mit *Finite-Domain-Constraints*[1] durchgeführt wurde, sei bereits hier darauf hingewiesen, dass sich die folgenden Erläuterungen und Beispiele weitgehend auf diese Art von Constraints beziehen werden. Nähere Informationen über andere Typen von Constraints können unter anderem in [Apt03] eingeholt werden. Zusätzlich können Constraint-Löser nach bestimmten Eigenschaften unterteilt werden, welche sich auf ihren Leistungsfähigkeiten auswirken. Beispielsweise kann ein Constraint-Löser in der Lage sein, die ein Problem beschreibenden Constraints auf Erfüllbarkeit zu prüfen. Genau betrachtet bedeutet dies, dass unmittelbar nach dem Absetzen der Constraints durch den Constraint-Löser festgestellt werden kann, ob überhaupt eine Lösung existieren kann.

2.1.2 Arbeitsweise der Constraint-Programmierung

Der Vorteil der Constraint-basierten Programmierung gegenüber der klassischen Programmierung basiert auf dem unterschiedlichen Ansatz zur Verarbeitung von den im Programm gesetzten Bedingungen. In der klassischen Programmierung werden zuerst die von den Constraints einbezogenen Variablen mit Werten belegt und anschließend werden sie darauf geprüft ob sie diese Constraints mit den zugewiesenen Werten erfüllen. Ist dies nicht der Fall, werden diese Variablen mit anderen Werten belegt und anschließend neu geprüft. Im schlimmsten Fall kann dies das Prüfen aller kombinatorisch möglichen Lösungsvektoren dieser Variablen bedeuten. Im Gegensatz dazu werden bei der Constraint-Programmierung die Constraints möglichst weit vor den

1 Diese Bedingungen beziehen sich nur auf Variablen mit endlichen Wertebereiche (engl. finite domain). Die Wertebereiche sind in diesem Fall endliche Mengen von natürlichen Zahlen.

Wertzuweisungen der Variablen abgesetzt.

Die Constraints sind im Allgemeinen zu diesem Zeitpunkt noch nicht berechenbar, da die Variablen noch keine konkreten Werte enthalten, aber sie werden zurückgestellt und der Constraint-Löser wacht ab diesem Zeitpunkt über diese Variablen und versucht mit Hilfe von Konsistenztechniken den von den Variablen aufgespannten Suchraum einzuschränken. Der Grad der Suchraumeinschränkungen bzw. der Erkennung von Inkonsistenzen hängt überwiegend von den verwendeten Konsistenzverfahren ab. Auf solche Konsistenzverfahren wird in Abschnitt 1.1.4 anhand von Beispielverfahren noch genauer eingegangen. Ein typischer Vorteil der sich gegenüber der klassischen Programmierung ergibt, ist der Zeitpunkt wenn einer Variable ein Wert zugewiesen wird. Entweder kann der Wert als unzulässig erkannt werden und er wird sofort zurückgewiesen (in diesem Fall gehört dieser Wert zu keiner Lösung) oder die Zuweisung der Variablen führt unmittelbar zu einer Suchraumeinschränkung.

2.1.3 Constraint-Satisfaction-Problem

Ein Problem, welches mit Hilfe von Constraints modelliert werden kann, wird allgemein als *Constraint-Satisfaction-Problem*(CSP) bezeichnet. Diese Klasse von Problemen stellt für die Constraint-Programmierung ein fundamentales Konzept dar.

Formal gesehen ist ein CSP eine Menge von Variablen $X=\{x_1,\ldots,x_n\}$ und einer Menge von Constraints C über diesen Variablen. Jeder Variable x_i wird eine endliche *Domäne* (engl. *domain*) $D_i\in\{D_1,\ldots,D_n\}$ zugeordnet. Ein Constraint $c\in C$ bezieht sich indessen auf eine Teilmenge von Variablen aus X. Der Suchraum, den das CSP aufspannt, ist das kartesische Produkt der Domänen seiner Variablen $D_1\times\cdots\times D_n$. Das CSP hat mindestens eine Lösung, wenn ein n-Tupel $(d_1,\ldots,d_n)\in D_1\times\cdots\times D_n$ gibt, dass alle Constraints $c\in C$ erfüllt. In diesem Fall liegt ein *konsistentes* CSP vor, andernfalls ein *inkonsistentes*. Das generelle Ziel eines CSP ist es entweder eine Lösung, alle Lösungen oder eine optimale Lösung zu finden. Für das Auffinden der optimalen Lösung muss eine Zielfunktion definiert werden, welche hinreichende Aussagen über die Qualität der gefundenen Lösungen angibt.

Wie bereits oben erwähnt, ist der aufgespannte Suchraum das kartesische Produkt der Domänen. Zudem sei $|D_i|$ die Kardinalität der Domäne D_i. Dann existieren $|D_1|*\ldots*|D_n|$ mögliche unterschiedliche Tupel $(d_1,\ldots,d_n)$ mit $d_i\in D_i$. Bei n Variablen mit gleichgroßen Domänen $|D_i|=d$ gäbe es d^n unterschiedliche Kombinationen die Variablen mit Werte zu belegen. Die Komplexität dieses CSP wäre also exponentiell. Ein typisches CSP könnte beispielsweise so

aussehen.

Beispiel 2.1: $X=\{x_1, x_2, x_3\}$ seien drei Variablen und $D_1=\{1,2,3,4\}, D_2=\{1,2,3,4\}$ und $D_3=\{1,2,3,4\}$ die ihnen zugehörigen Domänen. Die dieses Problem modellierenden Constraints seien $C=\{\, x_3=x_1+x_2,\ x_1\neq x_2,\ x_3>x_2+1\,\}$. Demzufolge sind von den 64 potentiellen Variablenbelegungen $x_1=2, x_2=1, x_3=3$ und $x_1=3, x_2=1, x_3=4$ die zwei einzigen Lösungen die dieses CSP erfüllen.

2.1.4 Constraint-Propagation

Aufgrund des Fehlen eines universellen effizienten Algorithmus zum Lösen von CSPs, wurden in der Vergangenheit verschiedene Techniken entwickelt, die in Kombination miteinander Lösungen ermitteln können. Eine effiziente Bearbeitung von Finite-Domain-CSP wird heutzutage durch die so genannte *Constraint-Propagation* in Verbindung mit Suchverfahren realisiert. Die Grundidee der Constraint-Propagation ist es, das CSP auf ein einfacheres äquivalentes CSP zurückzuführen. Dabei bezieht sich die Vereinfachung auf die Verkleinerung des Suchraums des CSP, welcher durch die Domänen der Variablen aufgespannt wird. Dies ist sehr sinnvoll, da ein CSP für ein praktisch relevantes Problem im Allgemeinen einen sehr großen Suchraum besitzt. Anschließend wird der verkleinerte Suchraum nach Lösungen durchsucht. In der Regel ist diese Suche aufgrund eines stark verkleinerten Suchraums wesentlich schneller. Damit der Suchraum reduziert wird, müssen folglich die Domänen verkleinert werden. Hierfür bedient man sich der bereits angesprochener Konsistenztechniken. In der Folge soll exemplarisch das Prinzip der Suchraumeinschränkung an zwei ausgewählten Konsistenzverfahren näher erläutert werden. Diese beiden Verfahren wurden ausgewählt, weil sie die in der Praxis meist angewandten Verfahren zur Suchraumeinschränkung darstellen. Das erste Verfahren ist die *Knotenkonsistenz* und und das zweite Verfahren ist die *Kantenkonsistenz*.

Die Funktionsweise dieser Konsistenztechniken wird nun mit Hilfe des folgenden Beispielproblems erläutert.

Beispiel 2.2: Gegeben sei ein Constraint-Satisfaction-Problem mit den Variablen $X=\{x, y, z\}$, den zugehörigen Wertebereichen $D_x=\{1,2,3,4,5\}, D_y=\{1,2,3,4,5\}, D_z=\{1,2,3,4,5\}$ und den Constraints $C=\{x>1, z<5, x\leq y, y<z\}$. In Abbildung 2.1 wird dieses Problem als ein so genanntes *Constraintnetz* dargestellt. Dieses Netz ist ein Graph, wobei jede Variable durch einen Knoten und jedes Constraint durch eine Kante dargestellt wird. Die unären Constraints werden

durch Schleifen an den jeweils zugeordneten Variablen symbolisiert und die binären Constraints als bidirektionale Kanten zwischen den, vom Constraint eingeschränkte, Variablen. Wahlweise können die binären Constraints auch als ungerichtete Kanten ohne Pfeile gezeichnet werden. Es ist leicht erkennbar, dass es für dieses relativ einfache CSP bereits $5^3 = 125$ mögliche Kombinationen gibt die Variablen zu belegen. Diese Kombinationen müssten alle über eine Suche getestet werden, wenn jede mögliche Lösung dieses Problems gefunden werden soll. Um dies zu vermeiden, sollte der Suchraum vor der Suche mit angemessenen Konsistenzverfahren verkleinert werden.

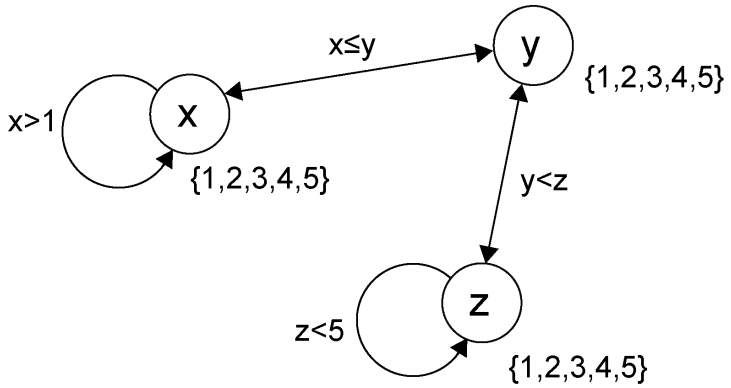

Abbildung 2.1: Darstellung eines Beispiel-CSP durch ein Constraintnetz

2.1.4.1 Knotenkonsistenz

Definition: Gegeben sei ein CSP mit der Variablenmenge $X = \{x_1, \ldots, x_n\}$, den Domänen $D = \{D_1, \ldots, D_n\}$ und den Constraints C. Sei $c \in C$ ein unäres Constraint und $x_i \in X$ die Variable, auf die sich dieses Constraint bezieht, dann ist das Constraint c *knotenkonsistent* (engl. node consistent), wenn das Constraint für jeden Wert aus der zu x_i gehörenden Domäne D_i erfüllt ist. Ein CSP ist genau dann knotenkonsistent, wenn alle unären Constraints dieses CSP knotenkonsistent sind. Ein CSP, welches keine einstelligen Constraints besitzt, ist immer knotenkonsistent.

Es ist leicht zu erkennen, dass das CSP in dem Beispiel laut der Definition nicht knotenkonsistent ist, da sowohl das unäre Constraint $x > 1$ als auch $z < 5$ nicht knotenkonsistent sind. Um Knotenkonsistenz zu erreichen, muss der Wert 1 zunächst aus dem Wertebereich D_x gestrichen werden, damit $x > 1$ erfüllt werden kann. Anschließend muss das Gleiche für den Wert 5 in D_z getan werden. Somit ergibt sich das folgende, in Abbildung 2.2 dargestellte, neue Constraintnetz. Die Knotenkonsistenz hat unmittelbar zur Folge, dass das Ermitteln aller Lösungen nur noch durch

$4*4*5=80$ Kombinationen anstatt der ursprünglichen 125 geschehen kann. Trotzdem wäre eine weitere Suchraumeinschränkung in diesem Fall erstrebenswert. Da nur einstellige Constraints mit diesem Verfahren betrachtet werden, sollten zusätzliche Konsistenztechniken (z.B. Kantenkonsistenz) verwendet werden, welche auf die Bearbeitung von mehrstelligen Constraints abzielen.

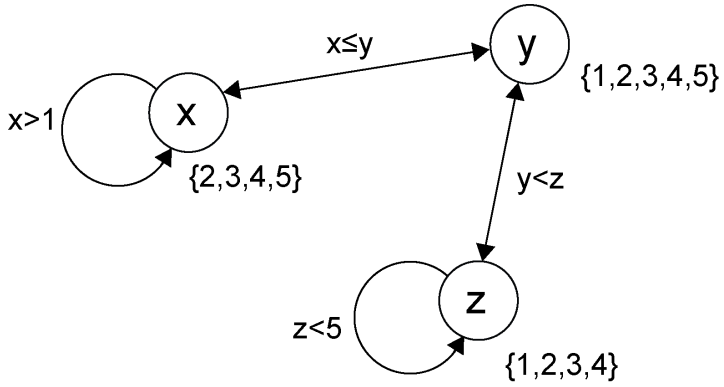

Abbildung 2.2: Ein knotenkonsistentes äquivalentes CSP

2.1.4.2 Kantenkonsistenz

Definition: Gegeben sei ein CSP mit der Variablenmenge $X=\{x_1,\ldots,x_n\}$, den Domänen $D=\{D_1,\ldots,D_n\}$ und den Constraints C. Sei $c\in C$ ein binäres Constraint und $\{x_i,x_j\}\subseteq X$ die beiden Variablen, dessen Relation durch dieses Constraint c eingeschränkt wird. Das binäre Constraint *c* ist genau dann *kantenkonsistent* (engl. arc consistent), wenn zu jedem Wert aus D_i ein Wert aus D_j existiert und umgekehrt auch zu jedem Wert aus D_j ein Wert aus D_i existiert, so dass dieses Constraint für diese Wertepaare erfüllt ist. Ein CSP ist kantenkonsistent, wenn alle binären Constraints dieses CSP kantenkonsistent sind.

In der Praxis existieren verschiedene Varianten des Kantenkonsistenz-Algorithmus. Die Bearbeitungsschritte werden anhand eines Beispiel-CSP aufgezeigt. In dem Beispiel wird zuerst ein beliebiges binäres Constraint ausgewählt, für welches die Kantenkonsistenz hergestellt werden soll. Ohne Beschränkung der Allgemeinheit wird hier das Constraint $x\le y$ gewählt. Die zu x und y korrespondierenden Domänen sind zu diesem Zeitpunkt $D_x=\{2,3,4,5\}$ und $D_y=\{1,2,3,4,5\}$. Die Werte dieser Domänen werden nun jeweils geprüft, ob für sie mindestens ein kompatibler Wert in der anderen Domäne existiert, mit dem diese beiden Werte das Constraint $x\le y$ erfüllen. Aufgrund

der Gleichheitsrelation von $x \leq y$ trifft dies für die Werte 2, 3, 4 und 5 beider Domänen zu. Nur für den Wert $1 \in D_y$ existiert kein kompatibler Wert in D_x. Deshalb wird er aus der Domäne entfernt. Da für $y=1$ das Constraint $x \leq y$ nicht erfüllt werden kann, ist $y=1$ auch in keiner Lösung enthalten. Nach diesem Abarbeitungsschritt ist das Constraint $x \leq y$ zunächst kantenkonsistent. Danach wird das noch unbearbeitete binäre Constraint $y<z$ geprüft. Die zu betrachtenden Domänen sind zu diesem Zeitpunkt $D_y=\{2,3,4,5\}$ und $D_z=\{1,2,3,4\}$. Als Konsequenz dessen, werden aus der Domäne D_y die Werte 4 und 5 wegen fehlenden kompatiblen Werten in D_z gestrichen. Umgekehrt werden aus D_z die Werte 1 und 2 gelöscht, weil es keine passenden Werte in D_y gibt, mit denen das Constraint eingehalten werden kann. Somit ist auch der Kantenkonsistenztest für $y<z$ vorerst abgeschlossen. Die Domänen sind zu diesem Zeitpunkt $D_x=\{2,3,4,5\}$, $D_y=\{2,3\}$ und $D_z=\{3,4\}$. Das Verfahren ist aber noch nicht beendet, weil im letzten Konsistenztest die Domäne von *y* reduziert wurde. Dies hat zur Folge, dass alle binären Constraints, welche sich auf *y* beziehen nochmal auf Kantenkonsistenz geprüft werden müssen. Denn die Reduzierung von D_y könnte wiederum die Werte anderer Domänen, die an diesen binären Constraints teilhaben, inkonsistent werden lassen. In diesem Beispiel-CSP ist es einzig das Constraint $x \leq y$, welches nochmals geprüft werden muss. Daraufhin erkennt dieser Test, dass die Belegungen $x=4$ und $x=5$ bei $D_y=\{2,3\}$ ungültig sind und entfernt die zwei Werte. Weil der Wertebereich von *y* nicht verändert wurde und *x* nicht Teil eines anderen binären Constraints ist, müssen daraufhin keine weiteren binären Constraint geprüft werden. Dieses Konsistenzverfahren ist somit abgeschlossen. Abbildung 2.3 zeigt das kantenkonsistente CSP zugehörige Constraintnetz.

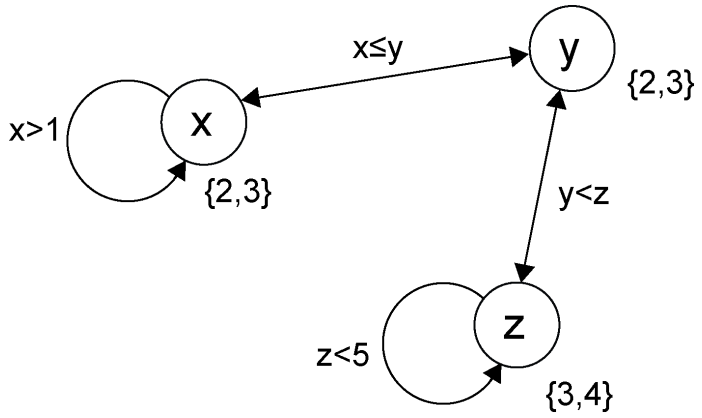

Abbildung 2.3: Ein knoten- und kantenkonsistentes äquivalentes CSP

Durch die Anwendung der beiden Konsistenztechniken wurde der Suchraum auf $2^3=8$ verschiedene

Kombiationen eingeschränkt, den Variablen Werte zuzuordnen. Dies beschleunigt die anschließende Suche gegenüber den ursprünglichen 125 Kombinationsmöglichkeiten erheblich. Schlussendlich besitzt das CSP die 4 Lösungen, $(x=2, y=2, z=3)$, $(x=2, y=2, z=4)$, $(x=2, y=3, z=4)$ und $(x=3, y=3, z=4)$. Es lässt sich an diesem Beispiel gut erkennen, dass die Konsistenzprüfung im Allgemeinen nicht alle ungültigen Belegungen eliminiert. Daher müssen anschließend mit Hilfe von Suchverfahren die Lösungen ermittelt werden. Knoten- und Kantenkonsistenz eines Problems impliziert zudem keine Lösbarkeit des Problems. Abbildung 2.4 demonstriert anschaulich ein CSP bei dem die Knoten- und Kantenkonsistenzprüfung ineffektiv ist. Dieses CSP ist in seiner Darstellung bereits knoten- und kantenkonsistent, aber trotzdem unerfüllbar.

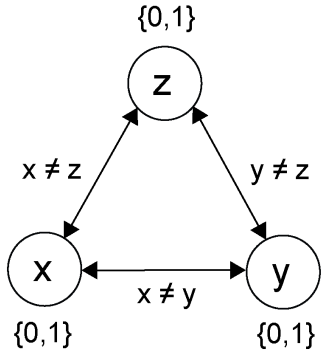

Abbildung 2.4: Ein knoten- und kantenkonsistentes, aber unerfüllbares CSP, Quelle:[Wolf07]

Sollten dagegen bereits bei der Konsistenzprüfung alle Werte einer Domäne gelöscht werden, so bedeutet dies, dass für dieses CSP keine Lösung existiert. Die entsprechende Variable kann mit keinem Wert die Constraints erfüllen. Der Grad der Suchraumeinschränkung ist abhängig von den eingesetzten Konsistenztechniken. Es ist in anderen Fällen effizienter andere Techniken anzuwenden. Knotenkonsistenz berücksichtigt nur die unären Constraints und Kantenkonsistenz nur binäre Constraints. Constraints mit höherer Stelligkeit wie zum Beispiel $x=y+z$ werden ignoriert. Zudem werden die Constraints mit ihren Variablen hier nur isoliert für sich auf Konsistenz geprüft. Es handelt sich in diesen Fällen um die so genannte *lokale Konsistenz*. Es existieren aufwändigere Verfahren, um stärkere Konsistenzeigenschaften zu prüfen, wie das Verfahren zur Prüfung der so genannten *Pfadkonsistenz*(engl. path consistency) oder *Grenzenkonsistenz*(engl. bounds consistency).

2.1.5 Suche

Um die Lösungen eines CSP zu finden, kann man sich verschiedener Verfahren bedienen. Zum einen wäre da, das *generate-and-test*-Paradigma, mit dem systematisch alle möglichen Wertbelegungen generiert werden und daraufhin geprüft werden ob diese Belegungen jedes Constraint erfüllt. Eine effektivere Methode verwendet das Backtracking-Paradigma, welches der meist angewendete Algorithmus zur systematischen Suche ist. Backtracking versucht schrittweise, durch wiederholte Wertzuweisung der Variablen, eine Teillösung zu einer Gesamtlösung zu vervollständigen. (vgl. [Bar98])

In dieser Arbeit bot es sich an, ausschließlich die *Tiefe-zuerst-Suche* für die Lösungssuche zu verwenden, denn es bestand aufgrund der annehmbaren Laufzeiten keinen sonderlichen Bedarf eine aufwändigere Suche implementieren zu müssen. Durch die Anwendung eines constraint-logischen Programmiersystems, das Tiefe-zuerst-Suche und Backtracking als Programmabarbeitungsmethoden verwendet, wurde die Realisierung von Suchverfahren zusätzlich erleichtert. In der Constraint-Programmierung hat sich für die Lösungssuche mit Hilfe von Backtracking der Begriff *Labeling* etabliert.

Im Folgenden soll dieses Suchverfahren näher erläutert werden. Dafür ist es hilfreich, sich den zu durchsuchenden Suchraum als eine Baumstruktur vorzustellen. Dabei besitzt der Baum maximal eine Tiefe, die der Anzahl zu belegender Variablen gleicht. Mit Tiefe sei hier die Anzahl der Kanten gemeint, die einen Pfad von der Wurzel zu einem Knoten bzw. Blatt bilden. Die Wurzel symbolisiert das aus der Konsistenzprüfung resultierende CSP, für das eine oder mehrere Lösungen gefunden werden sollen. Die Knoten des Baumes repräsentieren Teillösungen des CSP und die Blätter stellen die Gesamtlösungen des CSP dar. Die Kanten des Baumes repräsentieren Entscheidungen über die Belegungen der Variablen. Aus diesem Grund wird ein derartiger Baum auch als *Entscheidungsbaum* bezeichnet. In der vorliegende Arbeit wird sich auf Entscheidungen beschränkt, die jeweils genau einer Variable einen Wert aus ihrer Domäne zuweisen. Der Entscheidungsbaum wird hierfür in mehrere Ebenen unterteilt. Einer Ebene wird eine Menge von Knoten zugeordnet. Die Knoten einer Ebene besitzen Pfade gleicher Länge zum Wurzelknoten. Der Wurzelknoten allein bildet die Ebene 0. Die Kinderknoten der Wurzel werden der Ebene 1 zugeordnet. Die Kinder der Knoten aus Ebene 1 bilden die Ebene 2, usw. Zusätzlich wird jede Ebene mit einer Variablen des CSP assoziiert. Wobei die Kanten, welche von den Knoten einer Ebene ausgehen, die Entscheidung über die Belegung der zugehörigen Variable mit einem Wert

bedeuten. Abbildung 2.5 zeigt einen solchen Entscheidungsbaum für das im vorhergehenden Abschnitt bereits auf Knoten- und Kantenkonsistenz geprüfte Beispielproblem. Die Domänen der 3 Variablen x,y und z sind $D_x=\{2,3\}$, $D_y=\{2,3\}$ und $D_z=\{3,4\}$. Die verbleibenden 8 möglichen Kombinationen sind im Entscheidungsbaum an den Blättern direkt ablesbar.

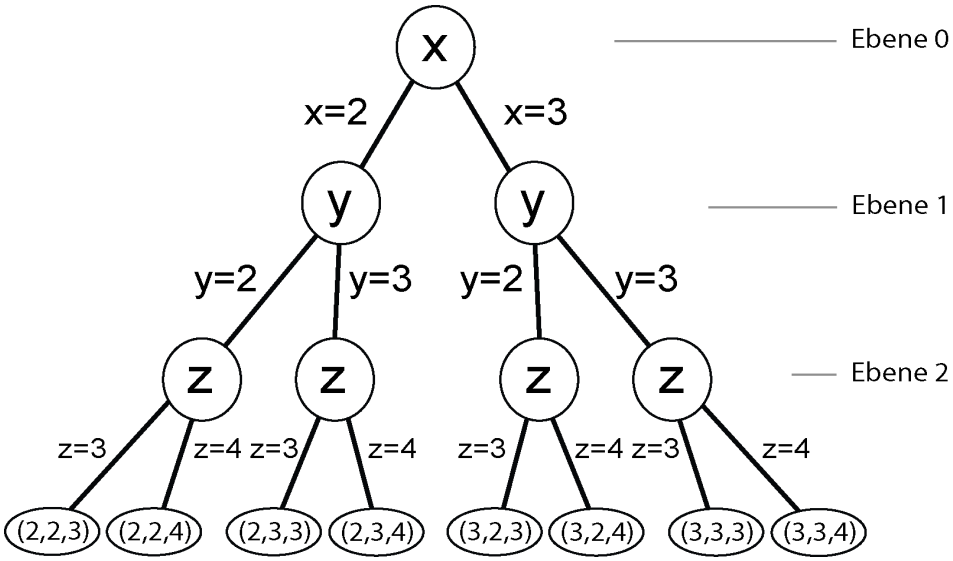

Abbildung 2.5: Ein Entscheidungsbaum für das Beispiel-CSP

Ausgehend vom Wurzelknoten, kann das Traversieren des Baumes bis zu einem Blatt, als eine schrittweise Belegung aller Variablen des CSP angesehen werden. Da infolge der Wertzuweisung der Variablen unmittelbar eine Suchraumeinschränkung stattfindet, wird der Constraint-Löser nach jeder Entscheidung geweckt und es werden unter Berücksichtigung der aktuellen Entscheidung, der abgesetzten Constraints und der bisher getroffenen Entscheidungen, den Wertzuweisungen, die geforderten Konsistenzeigenschaften erneut geprüft. Dies wiederum kann gegebenenfalls zu weiteren Domänenreduzierungen führen. Wird die geforderte Konsistenz des CSP hergestellt, kann daraufhin eine Wertzuweisung für die Variable des nachfolgenden Kindknoten getroffen werden. Dies wiederholt sich solange, bis ein Blattknoten erreicht und somit eine Lösung gefunden worden ist. Tritt andererseits bei der Domäneneinschränkung eine Inkonsistenz in Form einer leeren Domäne auf, wird die Traversierung im Entscheidungsbaum auf den Vorgängerknoten zurückgesetzt. Dies bedeutet, dass die aus der unmittelbar bevor getroffenen Entscheidung gefolgerten Suchraumeinschränkungen durch den Constraint-Löser zurückgenommen werden. Nach der Rücksetzung wird für den Vorgängerknoten, sofern vorhanden, eine andere bisher noch nicht gewählte Entscheidung getroffen. Ist dies nicht mehr möglich, wird soweit zu einem Knoten zurückgesetzt bis eine andere Entscheidung gewählt werden kann. Ist dies nicht möglich und das

Zurücksetzen endet bei dem Wurzelknoten, bedeutet dies, dass dieses CSP keine Lösung besitzt und die Suche beendet ist. Ein entscheidender Vorteil, der unmittelbar aus der Wertzuweisung folgenden Konsistenzprüfung ist der, dass nur die Werte für weitere Zuweisungen zur Auswahl stehen, welche die Konsistenz unter den bisherigen Constraints und Entscheidungen erfüllen. Somit kann die Traversierung von Teilbäumen eventuell frühzeitig abgebrochen werden. Um die Suchraumeinschränkung während der Wertzuweisungen zu veranschaulichen zeigt Abbildung 2.6 den mit Hilfe von Kantenkonsistenz reduzierten Entscheidungsbaum für das Beispiel-CSP. Die durch die Konsistenzprüfung verbleibenden Entscheidungsmöglichkeiten sind dunkel und die entfernten Entscheidungsmöglichkeiten hell gekennzeichnet. In diesem Beispiel ist gut zu erkennen, dass für die Teillösungen $x=2, y=3$ und $x=3, y=3$ die Teilbäume für $z=3$ aufgrund der Kantenkonsistenz mit dem Constraint $y<z$ entfernt worden sind. Zusätzlich wurde der Teilbaum für die Kombinationen $x=3, y=2$ gelöscht, da der Wert 2 aus der Domäne von y entfernt wird. Denn für $x=3$ erfüllt dieser Wert das Constraint $x \leq y$ nicht. Wie man an diesem Beispiel gut sehen kann, existieren Fälle für die Kantenkonsistenz eine effiziente Suchraumeinschränkung darstellt, denn die hier verbliebenen Teilbäume führen zu einer der vier möglichen Lösungen. In der Praxis gibt es allerdings auch oftmals Beispiele, wo aufwendigere Konsistenzverfahren verwendet werden müssen.

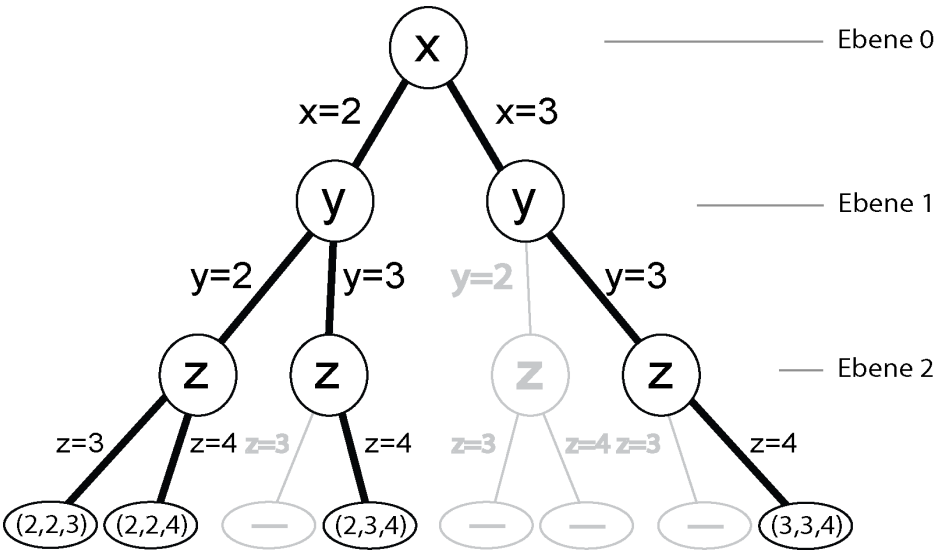

Abbildung 2.6: Ein durch Kantenkonsistenz eingeschränkter Entscheidungsbaum

Schließlich bleibt aber noch die Frage offen, wie durch den Entscheidungsbaum traversiert werden soll. So bemerkt Wolf [Wolf07] an, dass sich für die Suche nach Lösungen lediglich Heuristiken zur Traversierung des Suchraums formulieren lassen, die sich bei der Lösung vieler Probleme bewährt

haben. Dabei garantieren ausgewählte Suchstrategien weder das schnelle Finden einer Lösung noch das Erkennen der Unlösbarkeit eines Problems. Um eine Suche durchführen zu können, muss zum einen geklärt werden in welcher Reihenfolge die Variablen(V*ariable Ordering*) betrachtet und zum anderen in welcher Reihenfolge sie mit ihren Werten(V*alue Ordering*) belegt werden sollen.

2.1.5.1 Variable Ordering

Bei der Bestimmung, in welcher Reihenfolge die Variablen mit Werten belegt werden sollen, werden *dynamische* und *statische* Strategien unterschieden. Als dynamisch werden Strategien bezeichnet, bei denen sich die Variablenauswahl erst anhand des aktuellen Suchzeitpunktes bestimmen lässt. Dies wird durch die bei der Suche aktivierte Konsistenzherstellung bestimmt. Weil sich die Domänen während der Suche verändern können, kann sich auch die Wahl einer Variable aufgrund ihrer Domäneneigenschaften mit jedem Suchschritt dynamisch verändern. Bei statischen Strategien wird die Variablenreihenfolge bereits vor der Suche festgelegt und bleibt während der Suche unverändert. Typische Strategien für die Wahl der Variablenreihenfolge sind:

- *left-to-right* - Wähle aus einer Sequenz die Variablen von links nach rechts entsprechend ihren Auftreten.
- *first-fail* - Wähle die Variable mit den wenigsten Werten in der Domäne. Diese Wahl ist dynamisch und hängt stark davon ab welche Domänen durch die Konsistenzverfahren zu welchem Zeitpunkt der Suche reduziert wurden.
- *most-constrained* - Wähle dynamisch die Variable, die mit den meisten Constraints in Verbindung steht. Bei dieser Heuristik wird angenommen, dass diese Variable am schwierigsten zu lösen ist und sich ihre Festlegung vorzeitig auf die Einschränkung der Domänen anderer Variablen auswirkt.
- *min-/max-value* - Die Variable mit dem aktuell kleinsten bzw. größten Wert wird zuerst belegt.

Der Entscheidungsbaum aus Abbildung 2.6 zeigt eine Suche mit der statischen *left-to-right*-Heuristik. Die Variablen x, y und z werden nacheinander – in der Abbildung von oben nach unten - mit Werten belegt.

2.1.5.2 Value Ordering

Einen entscheidenden Einfluss auf den Erfolg einer Lösungssuche üben neben der Wahl der Variablen die Entscheidungen aus, in welcher Reihenfolge die Werte der Variablen gewählt werden. Die Strategie mit welcher ein Wert zuerst gewählt wird, kann die Geschwindigkeit der Suche und die Qualität der gefundenen Lösung beeinflussen. Dies ist besonders wichtig, wenn nur eine Lösung gesucht wird. Werden dagegen alle Lösungen gesucht oder wird keine Lösung gefunden spielt diese Reihenfolge keine Rolle. In diesen Fällen muss der komplette Suchraum durchsucht werden. Typische Strategien für die Wahl der Wertereihenfolge werden in der Regel durch Constraint-Löser unterstützt. Oft soll der kleinste, größte oder mittlere Wert der betrachteten Domäne der Variablen zugeordnet werden. In einer anderen Strategie soll von einem angegebenen Wert der nächstliegende ausgesucht werden.

2.1.6 globale Constraints

In der Praxis stellen Constraint-Löser neben unären und binären Constraints oftmals auch so genannte globale Constraints zur Verfügung. Globale Constraints stellen ein mächtiges Werkzeug zur Beschreibung von in vielen Bereichen im wieder auftretenden komplexen Problemen dar. Diese Probleme sind durch mitgelieferte generische Prädikate formulierbar. Um den Grad der Abstraktion und Mächtigkeit dieser Prädikate möglichst hoch zu halten, verfügen diese Prädikate in der Regel über eine Vielzahl an Parameter. Durch die Parameter können somit speziell an das jeweilige Problem angepasste Constraints abgesetzt werden. Globale Constraints nutzen speziell entwickelte Algorithmen zur effizienteren Suchraumeinschränkung und Speicherersparnis.

Da in der Arbeit auf globale Constraints zurückgegriffen wurde, werden in diesem Abschnitt diejenigen Constraints vorgestellt, deren Verwendung für das Programm am bedeutsamsten sind. In der praktischen Arbeit wurde das Constraint-Programmiersystem *CHIP* verwendet. *CHIP* steht für *Constraint Handling in Prolog* und wurde von der Firma *COSYTEC* entwickelt. Wie aus dem Namen schon zu erkennen ist wird das Fundament durch die logische Programmiersprache Prolog gebildet. Daher entspricht die Syntax der im folgenden vorgestellten Prädikate auch dieser Programmiersprache.

2.1.6.1 Das diffn-Constraint

Mit dem *diffn*-Constraint kann eine Menge von *m* *n*-dimensionalen Rechtecken in einem *n*-

dimensionalen Raum überlappungsfrei platziert werden. Aufgrund der Tatsache, dass sich zu dieser Forderung noch weitere Bedingungen wie etwa gewünschte Volumen der Rechtecke ergänzend formulieren lassen, existieren für dieses Constraint mehrere Versionen. Hier wird lediglich die Grundversion dieses Constraints verwendet. Wenn die Forderung gestellt wird, dass sich *m* *n*-dimensionale Rechtecke in einem *n*-dimensionalen Raum überlappungsfrei platziert werden sollen, wird das globale Constraint mit einer Liste von *m* Teillisten verwendet. Jede Teilliste repräsentiert genau ein Rechteck. Das i-te n-dimensionale Rechteck wird mit $[O_{i1},\ldots,O_{in},L_{i1},\ldots,L_{in}]$ beschrieben. Die Variablen $O_{i1},\ldots,O_{in}$ geben die Koordinaten des Eckpunktes des i-ten Rechtecks an, welcher sich am dichtesten zum Ursprungspunkt $(0,\ldots,0)$ befindet. $L_{i1},\ldots,L_{in}$ beschreiben die Ausdehnungen des Rechtecks in den *n* Dimensionen. Diese Parameter dürfen hier sowohl Variablen als auch natürliche Zahlen sein. Zusätzlich müssen $L_{i1},\ldots,L_{in}$ größer Null sein, d.h. sie müssen echte Ausdehnungen vom Ursprung weg in den jeweiligen Dimensionen darstellen. Die in [Co03b] angegebene mathematische Formel, durch die die im *diffn*-Constraint geforderte Überlappungsfreiheit beschrieben wird, lautet:

$$\forall i\in[1,m],\ \forall j\in[1,m],\ i\neq j,\ \exists k\in[1,n]: O_{ik}\geq O_{jk}+L_{jk}\ \vee\ O_{jk}\geq O_{ik}+L_{ik}$$

Dies kann so interpretiert werden, dass zwei voneinander verschiedene Rechtecke mit den Indizes *i* bzw. *j* sich in mindestens einer Dimension *k* soweit unterscheiden müssen, dass eines der beiden Rechtecke frühestens dort beginnt wo das andere aufhört. Beispielsweise wird mit $O_{ik}\geq O_{jk}+L_{jk}$ ausgedrückt, dass das Rechteck *i* weiter vom Ursprung entfernt beginnt als das Rechteck *j* mit seiner Ausdehnung in der *k*-ten Dimension endet.

In dieser Arbeit müssen lediglich Überlappungsprobleme für den 2-dimensionalen Raum gelöst werden. Dies hat den zusätzlichen Vorteil, dass solche Probleme gut veranschaulicht werden können. Abbildung 2.7 zeigt ein Beispiel mit drei überlappungsfreien Rechtecken im zweidimensionalen Raum.

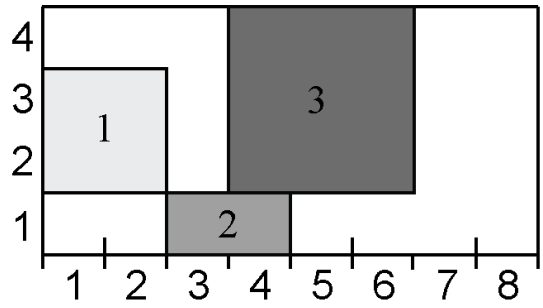

Abbildung 2.7: diffn([[1,2,2,2],[3,1,2,1],[4,2,3,3]]), Quelle:[Co03b]

2.1.6.2 Das among-Constraint

Das *among*-Constraint ist ein Werkzeug für die Formulierung von Forderungen zu Auftretenshäufigkeiten über *Finite-Domain*-Variablen. Mit Hilfe dieses Constraints kann das Auftreten von Variablen mit bestimmten Werten aus einer Menge von Variablen erzwungen werden. Diese Art von Bedingung ist oftmals Bestandteil der Modellierung in dieser Arbeit. Beispielsweise lässt sich die Bedingung formulieren, dass aus einer Menge von 10 Variablen mindestens 2 und maximal 5 Variablen ungerade Werte haben dürfen. Es könnte auch verlangt werden, dass aus einer Menge von Variablen genau eine Variable existieren muss, die den Wert Null annimmt. Durch die unscharfe Formulierung dieser Bedingungen, dass beliebige Variablen einer Menge diese Bedingungen erfüllen können, wird im Allgemeinen der Suchraum der Variablen nicht bereits beim Absetzen dieser Constraints eingeschränkt. Dies ist aufgrund der angegebenen Informationen meist nicht möglich. Erst im Zusammenspiel mit anderen Constraints bzw. mit weiteren Forderungen für das Auftreten anderer Werte kann die Propagation erfolgreich genutzt werden. Sollte dies nicht der Fall sein, kann spätestens bei der Wertbelegung während der Suche von den Konsistenztechniken dieses Constraints profitiert werden. Dort kann für das vorher erwähnte Beispiel bei einer Belegung einer Variable mit dem Wert Null sofort darauf geschlossen werden, dass die verbleibenden Variablen den Wert Null nicht annehmen dürfen. Für dieses Constraint gibt es verschiedene Aufrufvarianten. Die folgende Beschreibung erläutert die Variante, die in der Applikation zum Einsatz kam. Die Syntax dieses Prädikats ist in [Co03b] mit

$$among(N, [X_1, \ldots, X_s], [C_1, \ldots, C_s], [V_1, \ldots, V_m], Event)$$

angegeben. Der Parameter N ist eine *Finite-Domain*-Variable. Durch ihren Wertebereich wird die Anzahl der Variablen aus der Menge $[X_1, \ldots, X_s]$ begrenzt, die die geforderten Werte $[V_1, \ldots, V_m]$

enthalten müssen. $[C_1,\ldots,C_s]$ ist eine Liste von natürlichen Zahlen, welche für die Evaluierung des Constraints auf die Variablen addiert werden. Diese Konstanten wurden nicht benötigt und wurden deswegen immer als Liste von Nullen angegeben. $[V_1,\ldots,V_m]$ ist die Liste von Werten, welche nur die, in ihrer Anzahl eingeschränkten, Variablen annehmen dürfen. *Event* ist ein systemtechnischer Parameter zur Regulierung der Suchraumeinschränkung dieses Constraints und dessen Bedeutung hier vernachlässigt werden. Mathematisch kann die durch die Parameter geforderte absolute Häufigkeit so formuliert werden:

$$a \le N \le b,\ a \in \mathbb{N},\ b \in \mathbb{N},\ X=\{X_1+C_1,\ldots,X_s+C_s\},\ V=\{V_1,\ldots,V_m\},$$
$$T=\{t \mid t \in X \wedge t \in V\},\ |T|=N$$

Eine typische Anwendung eines *among*-Constraints zeigt folgendes Beispiel.

Beispiel 2.3: Von den 5 Variablen $[X_1, X_2, X_3, X_4, X_5]$ sollen zwei oder drei Variablen mit den Werten 8 oder 9 in einer Lösung enthalten sein. Das zugehörige Constraint mit $N \in [2,3]$ lautet:

$$among(N,[X_1,X_2,X_3,X_4,X_5],[0,0,0,0,0],[8,9],all)$$

Häufig werden in komplexeren Szenarien Forderungen gestellt, dass verschiedene Werte unterschiedlich oft auftreten sollen. Um dies zu beschreiben müssen mehrere *among*-Constraints verwendet werden. Aus diesem Grund stellt *CHIP* eine erweiterte Variante zur Verfügung, die mit einem Aufruf mehrere Aufrufe des vorgestellten *among*-Constraints ermöglicht. Laut [Co03b] ist diese Variante nicht nur eine kompaktere Darstellung, sondern sie reduziert den Speicherverbrauch der Constraints gegenüber der Nutzung mehrerer einzelner *among*-Constraints drastisch ein und ermittelt bessere Resultate in der Propagation. Die Syntax zu dieser erweiterten Version, welche *k* Aufrufe des vorher beschriebenen *among*-Constraints repräsentiert ist:

$$among([N_1,\ldots,N_k],[X_1,\ldots,X_s],[C_1,\ldots,C_s],[[V_{11},\ldots,V_{1m}],\ldots,[V_{k1},\ldots,V_{kn}]],Event)$$

Die Parameter N_i und $[V_{i1},\ldots,V_{im}]$ beschreiben das i-te *among*-Constraint über den Variablen $[X_1,\ldots,X_s]$. Das folgende Beispiel verdeutlicht die Anwendung der beiden Varianten.

Beispiel 2.4: Gegeben sind fünf Variablen $[X_1, X_2, X_3, X_4, X_5]$ und jede Variable besitzt eine Domäne über $[6,7,8,9]$. In den Lösungen sollen mindestens zwei und maximal drei Variablen den Wert 6 annehmen. Der Wert 7 soll genau einmal in der Lösung vorkommen und die Werte 8 und 9 sollen maximal zweimal vorkommen. Die drei Constraints,

$$among(N_{1,}[X_1,X_2,X_3,X_4,X_5],[0,0,0,0,0],[6],all) \quad mit \quad N_1 \in [2,3]$$
$$among(N_{2,}[X_1,X_2,X_3,X_4,X_5],[0,0,0,0,0],[7],all) \quad mit \quad N_2 \in [1]$$
$$among(N_{3,}[X_1,X_2,X_3,X_4,X_5],[0,0,0,0,0],[8,9],all) \quad mit \quad N_3 \in [0,1,2]$$

können durch den folgenden Aufruf ersetzt werden.

$$among([N_{1,}N_{2,}N_3],[X_1,X_2,X_3,X_4,X_5],[0,0,0,0,0],[[6],[7],[8,9]],all)$$
$$mit\ N_1 \in [2,3],\ N_2 \in [1]\ und\ N_3 \in [0,1,2]$$

Die Tupel (7,6,6,8,9), (7,6,6,6,9) oder (9,6,7,6,9) erfüllen beispielsweise diese Bedingungen. Aber es existieren noch weitere Lösungen. Anzumerken sei, dass N_3 in diesem Beispiel in keiner Lösung Null sein kann, denn die Werte 6 und 7 dürfen zusammen maximal viermal vorkommen. Die fünfte Variable muss deshalb einen der beiden verbleibenden Werte 8 oder 9 annehmen. Da N_3 die absolute Häufigkeit von 8 und 9 angibt, ist N_3 mindestens Eins.

2.2 Eisenbahntechnische Grundlagen

Da sich diese Arbeit mit einer Problematik des Eisenbahnwesens befasst, liefert dieser Abschnitt eine kurze Einführung in die fachspezifische Begriffswelt des Schienenverkehrs. Zum einen werden Prinzipien zur Regelung und Sicherung der Zugfolge dargelegt und zum anderen werden Begriffe – besonders aus dem Bereich Infrastruktur- definiert. Kaum ein anderer technischer Bereich besitzt eine so umfangreiche Nomenklatur wie das Bahnwesen. Durch den historischen Entstehungsprozess der Eisenbahn entstanden auch immer wieder neue Begriffe, welche den Neueinstieg in dieses Fachgebiet mitunter kompliziert wirken lassen oder sogar erschweren. So kann es vorkommen, dass für die selbe Definition mehrere Begriffe existieren. Beispielsweise kann dies von einer Definitionsänderung eines Begriffes kommen, der daraufhin dieselbe Bedeutung wie ein bereits existierender Begriff erhält. In [Pa04] wird dies an der historisch entstandenen Begriffsdoppelung der Begriffe Zugfolgestelle und Blockstelle gezeigt. Weiterhin lassen sich einige Begriffe aufgrund ihrer aufgestellten Definitionen nicht sauber voneinander trennen. (vgl. [Pa04], S.13) Hinzu kommt noch, dass das Eisenbahnwesen keiner internationale Vereinheitlichung unterliegt. Dies äußert sich nicht nur in betrieblichen Regelwerken der einzelnen Bahnen, sondern auch bereits in fundamentalen Definitionen des Bahnwesens. Diese Arbeit orientiert sich in erster Linie an den in Deutschland bestehenden Begriffen und Prinzipien. Das Fundament für den Bau und Betrieb von Eisenbahnen in Deutschland bildet eine Rechtsverordnung, die so genannte *Eisenbahn-Bau- und Betriebsordnung* (EBO). Sie gibt den gesetzlichen Rahmen vor, in dem die Bahnen ihre

betrieblichen Regelwerke zu gestalten haben. In ihr werden grundlegende Definitionen, Grundsätze der Betriebsführung, technische Normen und Anforderungen an das Personal festgelegt. Die größte deutsche Bahngesellschaft, welche sich nach der EBO richtet ist die *Deutsche Bahn AG* (DB AG).

Unterteilung der Gleise im Eisenbahnnetz

Ein Eisenbahnnetz wird in *Bahnhöfe* und „*freie Strecken*“ eingeteilt. Die Gleise der freien Strecke sind Hauptgleise. In §4 (11) EBO wurden *Hauptgleise* definiert als Gleise, die planmäßig von Zügen befahren werden dürfen. Im Bahnhofsbereich wird zwischen Haupt- und *Nebengleise* unterschieden. *Durchgehende Hauptgleise* sind die Hauptgleise der freien Strecke und deren Fortsetzung in den Bahnhöfen. Weitere sich in Bahnhöfen befindende Hauptgleise sind die *Überholungsgleise*. Das sind Gleise zur betrieblichen und verkehrlichen Nutzung. Die restlichen Gleise sind *Nebengleise*. Auf Nebengleisen finden im allgemeinen Rangierfahrten unter anderem zum Umstellen, Abstellen und Aufstellen von Zügen statt. Abbildung 2.8 veranschaulicht eine Unterteilung dieser Gleise.

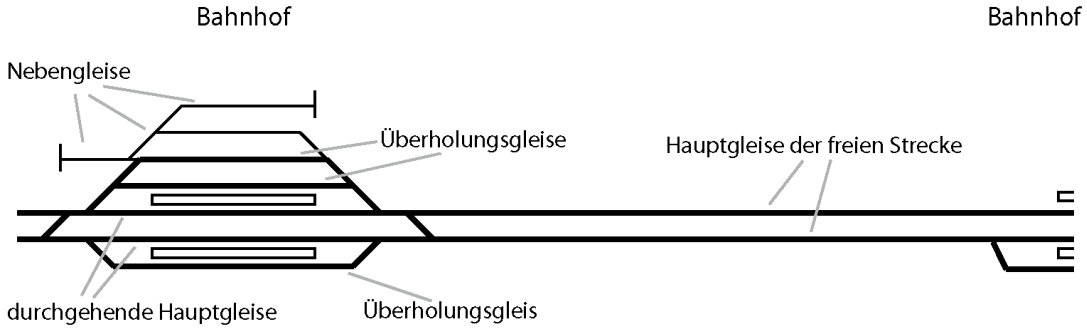

Abbildung 2.8: Unterteilung der Gleise in Haupt- und Nebengleise

Bahnhöfe

Bahnhöfe werden in §4 der EBO als Bahnanlagen mit mindestens einer Weiche definiert. Züge dürfen in Bahnhöfe beginnen, enden, ausweichen oder wenden. Die Grenzen zwischen den Bahnhöfen und der freien Strecke werden im allgemeinen durch Einfahrsignale oder Trapeztafeln, sonst den Einfahrweichen gebildet.

Weichen

Damit ein Zug von einem Gleis auf ein anderes Gleis fahren kann, muss er über Weichen (engl. switch) fahren. In der Praxis existieren aufgrund bestimmter Anforderungen verschiedene Arten von

Weichen (Abbildung 2.9). Die *einfache Weiche* besteht aus einem geraden Stammgleis und einem gekrümmten abzweigenden Gleis. *Bogenweichen* bestehen aus zwei gekrümmten Gleisen. *Doppelweichen* sind zwei ineinander geschobene einfache Weichen und besitzen somit zwei Abzweigungen. Sie werden beispielsweise nur dann verlegt, wenn der Platz für das Verlegen zweier einfacher Weichen nicht zur Verfügung steht. *Kreuzungsweichen* bestehen aus einer Gleisüberschneidung und ein oder zwei Gleisübergängen.

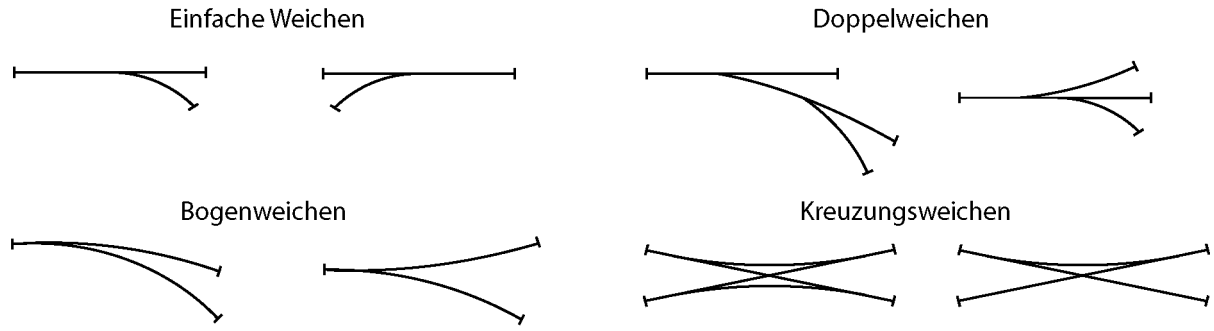

Abbildung 2.9: Weichenarten

Einfache Weichen werden aufgrund ihrer gegenüber anderen Weichenarten niedrigen Anschaffungs- und Unterhaltungskosten am häufigsten verwendet. Zudem ist man bemüht, die Anzahl der Weichen klein zu halten bzw. bestehende Weichen durch einfache Weichen zu ersetzen. Dies gilt laut [Fie05] und [Fen03] besonders für Doppel- und Kreuzungsweichen. Eine Weiche wird *„spitz befahren“*, wenn der Zug vom Stammgleis in Richtung der Gleisverzweigungen der Weiche fährt. Dagegen wird eine Weiche *„stumpf befahren“*, wenn der Zug ausgehend von einem der Weichenschenkel die Weiche überquert. Im Gegensatz zu Kurven wird der Bogen einer Weiche mit Ausnahme der *Bogenweiche* nicht überhöht. Folglich wird das Abzweiggleis, abhängig vom Bogenradius seiner Krümmung, mit einer gegenüber dem Stammgleis reduzierten Geschwindigkeit befahren. Daher kann eine Weiche, über die abgezweigt wird, als eine Langsamfahrstelle angesehen werden. Nach [Fie05] lässt sich die Geschwindigkeit für einen Bogenradius r mit $v = 2{,}91 * \sqrt{(r)};\ \ r[m], v[km/h]$ berechnen. Die folgende Tabelle zeigt bei der DB AG standardmäßig verwendete Weichenradien und ihre zulässigen Geschwindigkeiten.

Radius	190 m	300 m	500 m	760 m	1200 m	2500 m
zul. Geschwindigkeit	40 km/h	50 km/h	65 km/h	80 km/h	100 km/h	130 km/h

Tabelle 1: zulässige Geschwindigkeiten in Weichen

Zwischen den beiden Schenkeln einer Weiche befindet sich dessen *Grenzzeichen*[2]. Bis zu diesem

2 In der Praxis wird das Grenzzeichen durch einen rot-weißen Pflock gekennzeichnet.

Zeichen können zwei Züge je einen der zwei Weichenverzweigungen befahren, ohne mit dem anderen Zug zu kollidieren.

2.2.1 Verfahren zur Abstandshaltung

Im Schienenverkehr treten zwei wichtige Probleme auf, die durch besondere Maßnahmen gelöst werden müssen. Zum einen ist es die Spurführung des Zuges an den Schienen und zum anderen ist es der aufgrund geringer Haftreibung entstehende lange Bremsweg eines Zuges. Um eine sichere Spurführung gewährleisten zu können, müssen spezielle Techniken zur Regelung und Sicherung der Fahrwegelemente, z.B. Weichen, verwendet werden. Der Haftreibungsbeiwert von Stahlrädern auf Stahlschienen ist laut [Pa04] um ein achtfaches kleiner als im Straßenverkehr. Die dadurch resultierenden langen Bremswege der Züge übersteigen i.a. die Sichtweite des Zugführers um ein Vielfaches. Daher müssen für die sichere Abwicklung des Betriebes besondere Verfahren zur Regelung und Sicherung der Zugfolge bestehen.

Im folgenden werden die theoretisch möglichen Verfahren zur Abstandshaltung aufgezeigt. Dies soll für ein besseres Verständnis für die Notwendigkeit bestehender Infrastrukturelemente beitragen. Dabei ist bei zwei aufeinander folgenden Zügen der räumliche Abstand zwischen dem Zugschluss des voraus fahrenden Zuges und der Spitze der nachfolgenden Zuges gemeint.

Fahren im relativen Bremswegabstand

Nach dieser Methode fahren zwei direkt aufeinander folgende Züge in einem Mindestabstand, welcher der Differenz der Bremswege der beiden Züge entspricht (Abbildung 2.10). Diese Technik zur Abstandshaltung ist auch im Straßenverkehr üblich. Allerdings ist dieses Verfahren im Schienenverkehr schwerer realisierbar, da umzustellende Weichen zwischen den Zügen bzw. ein Unfall des voraus fahrenden Zuges ortsfeste Gefahrenpunkte darstellen. Um vor diesen Punkten sicher zum Halten zu kommen, muss der volle Bremsweg zur Verfügung stehen.

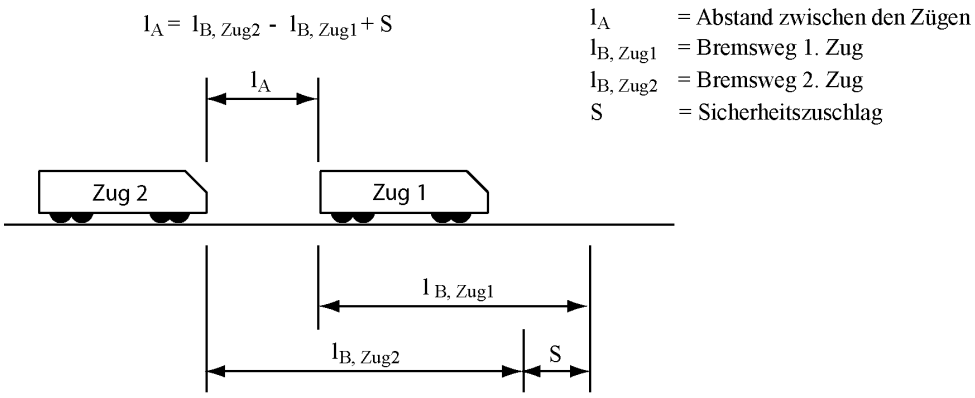

Abbildung 2.10: Zugfolge im relativen Bremswegabstand, Quelle:[Pa04]

Fahren im absoluten Bremswegabstand

Bei diesem Verfahren folgt der nachfolgende Zug mindestens in dem Abstand seines kompletten geschwindigkeitsabhängigen Bremsweges (Abbildung 2.11). “Der Zugschluss des vorausfahrenden Zuges bildet einen wandernden Gefahrpunkt, vor dem ein folgender Zug mit Sicherheit zum Halten kommen muss. Wenn zwischen zwei aufeinander folgenden Zügen Weichen umgestellt werden, wechselt der Gefahrpunkt vom Schluss des vorausfahrenden Zuges auf den ortsfesten Gefahrpunkt der Weiche. Solange die Weiche nicht in der für den nachfolgenden Zug korrekten Stellung liegt, wird diesem Zug eine notwendige Bremsung gemeldet. Nach dem Stellen der Weiche darf die Bremsung des nachfolgenden Zuges aufgehoben werden. Somit ist immer ein ausreichender Bremsweg vorhanden.“ ([Pa04], S.39ff)

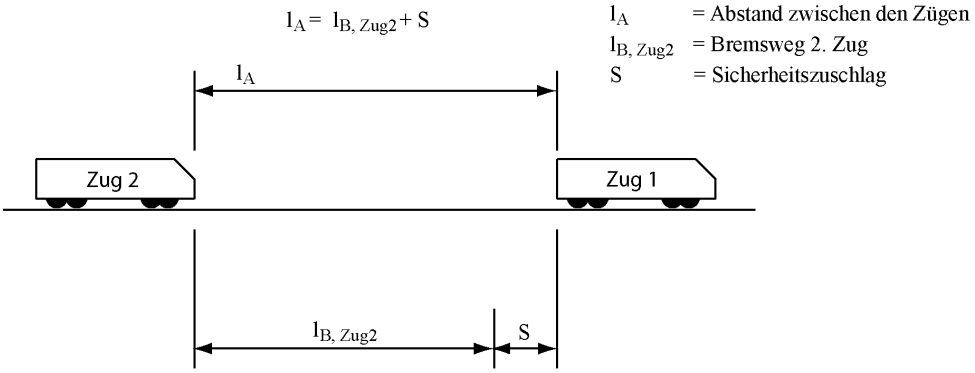

Abbildung 2.11: Zugfolge im absoluten Bremswegabstand, Quelle:[Pa04]

Fahren im Raumabstand

Für das Fahren im Raumabstand wird im Gegensatz zu den beiden vorher beschriebenen Verfahren kein geschwindigkeitsabhängiger, sondern immer ein fester Abstand zwischen zwei aufeinander folgenden Zügen gehalten. Dieser Abstand entspricht mindestens dem Bremsweg der maximal

zulässigen Geschwindigkeit. Zusätzlich kann hier noch zwischen *Fahren im festen Raumabstand* und *Fahren im wandernden Raumabstand* unterschieden werden. In der Praxis wird die Zugfolgesicherung oftmals durch ortsfeste Signalisierung realisiert. Die Gleisfreimeldung erfolgt insgesamt für den Gleisabschnitt zwischen den zwei Signalen. Die Länge des Gleisabschnittes, welche durch den Abstand seiner begrenzenden Signale bestimmt wird, wird zum Folgeabstand hinzugerechnet (Abbildung 2.12). Ein derartiger Gleisabschnitt wird in der Fachsprache *Blockabschnitt*, *Blockstrecke* oder auch *Zugfolgeabschnitt* genannt. In der Abbildung ist gut zu erkennen, dass die Blockabschnittslänge l_{Block} den Zugfolgeabstand direkt beeinflusst. Durch die Verkürzung der Blockabschnitte und der daraus resultierenden verkürzten Zugfolgezeit kann daher eine Leistungssteigerung der Strecke erzielt werden. Infolgedessen werden aber aufgrund der verkürzten Signalabstände mehr Signale benötigt, was sich in erhöhten Streckenkosten widerspiegelt. Wegen der ortsfesten Blockabschnitte wird die beschriebene Fahrweise als das Fahren im festen Raumabstand bezeichnet.

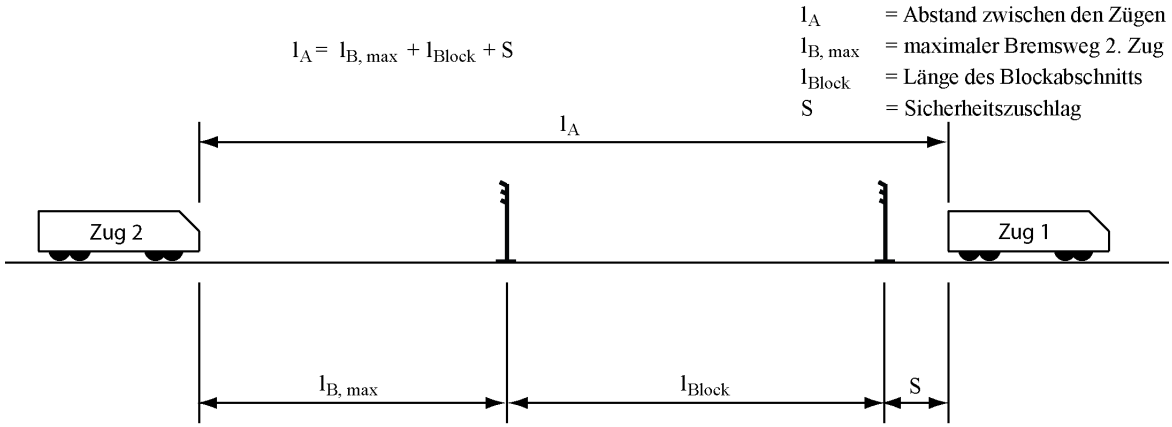

Abbildung 2.12: Zugfolge im festen Raumabstand, Quelle:[Pa04]

Beim Fahren im wandernden Raumabstand wird die Strecke kontinuierlich freigegeben, was bedeutet, dass die Blockabschnittslängen l_{Block} Null sind. Wird beim Fahren im wandernden Raumabstand nicht mehr der für die maximal zulässige Geschwindigkeit geltende Bremsweg zur Abstandshaltung benutzt, sondern nur noch der geschwindigkeitsabhängige Bremsweg, so gleicht dies wiederum dem Fahren im absoluten Bremswegabstand.

2.2.2 Fahren im festen Raumabstand

Aufgrund dessen, dass sich das Fahren im festen Raumabstand als das Standardverfahren zur Zugfolgesicherung im Eisenbahnbetrieb entwickelt hat und sich dies unmittelbar in der Infrastruktur

widerspiegelt, wird dieses Verfahren und deren Realisierung hier näher beschrieben. Infrastrukturseitig werden die Gleise beim Fahren im festen Raumabstand durch ortsfeste *Hauptsignale* in Blockabschnitte unterteilt. Hauptsignale auf der freien Strecke, die die Einfahrt in einen Blockabschnitt sichern werden auch *Blocksignale* genannt. Die Blockabschnitte besitzen eine Mindestlänge, welche dem für die jeweilige Strecke festgelegte längste Bremsweg gleicht. Dieser Bremsweg, auch *Regelbremsweg* genannt, ist bei der Deutschen Bahn AG nach EBO §35 (4) für *Hauptbahnen* auf 1000 m und für *Nebenbahnen* auf 700 m festgelegt. Aus ökonomischen Gründen werden laut [Fie05] in der Praxis auf freien Strecken oftmals Blockabschnitte mit 2000 m bis 4000 m gebildet. Grundsätzlich darf sich in jedem Blockabschnitt maximal ein Zug befinden. Damit ein Zug in einen Blockabschnitt einfahren darf, muss der Blockabschnitt frei von Fahrzeugen sein (vgl. EBO $4 (3)), der *Durchrutschweg* hinter dem Signal am Ende des Blockabschnitts muss frei sein und der voraus fahrende Zug muss durch ein Halt zeigendes Signal gedeckt werden. Der *Durchrutschweg*(DRW) wird im Bahnwesen auch als *Schutzstrecke* oder *Gefahrpunktabstand* bezeichnet. Dieser Gleisabschnitt stellt einen Sicherheitszuschlag gegen das Verbremsen eines Zuges dar und beträgt bei der DB AG in der Regel 200 m. Da die Einfahrt in einen Blockabschnitt an diese Bedingungen gebunden ist, wird somit sichergestellt, dass der voraus gefahrene Zug den Blockabschnitt und den Durchrutschweg vollständig geräumt haben muss. Technisch wird diese Gleisfreimeldung mit Hilfe einer *Signalzugschlussstelle* am Ende des Durchrutschweges realisiert. Auf die einzelnen Details der Gleisfreimeldung wird hier aber nicht weiter eingegangen. Für eine ausführliche Beschreibung der zugrunde liegenden Technik der Gleisfreimeldeanlagen wird hier auf [Fen03] verwiesen. Abbildung 2.13 zeigt ein typische Anordnung der Fahrwegelemente für das Fahren im festen Raumabstand bei der Deutschen Bahn AG.

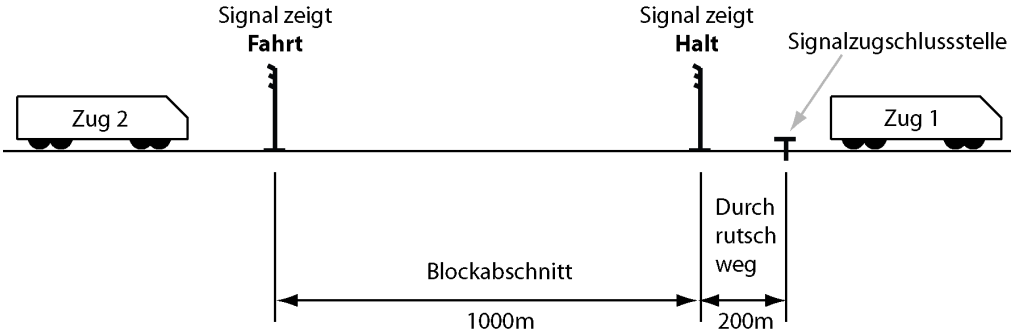

Abbildung 2.13: Fahren im festen Raumabstand bei der Deutschen Bahn AG

Wird ein Gleis im Zweirichtungsbetrieb, also in beide Richtungen befahren, muss für das Einfahren in einem Blockabschnitt noch zusätzlich die Gegenfahrt anderer Züge ausgeschlossen werden.

Die genauen Standorte der Hauptsignale sind davon abhängig, wo sich die von ihnen gedeckten Gefahrenpunkte befinden. Hauptsignale werden so weit davor aufgestellt, dass ein ausreichender Durchrutschweg vor dem maßgebenden Gefahrenpunkt endet. *Gefahrenpunkte* sind

- der Anfang der ersten hinter dem Signal liegenden spitz befahrenen Weiche
- das Grenzeichen einer hinter dem Signal liegenden Weiche oder Kreuzung
- der Zugschluss oder die Spitze eines haltenden Zuges
- eine Rangierhalttafel

Vorsignalisierung

Damit ein Zug vor einem Halt zeigenden Signal zum stehen kommt muss der Zug rechtzeitig abgebremst werden. Dies kann sich aufgrund langer Bremswege schwierig gestalten, da die Hauptsignale in dem Moment des Einleiten der Bremsung normalerweise außerhalb der Sichtweite des Triebwagenführers liegen. Aus diesem Grund wurden die Vorsignale eingeführt. Ein Vorsignal steht im ausreichendem Abstand, vor dem ihm zugeordneten Hauptsignal und zeigt dessen Signalbegriff an. Für gewöhnlich steht das Vorsignal im Abstand des Regelbremsweges vor dem Hauptsignal, d.h. abhängig von der auf dem Streckenabschnitt zulässigen Höchstgeschwindigkeit v und dem Gefälle der Strecke (vgl. EBO \$14 (13)). Bei der DB AG beträgt der Vorsignalabstand in der Ebene 1000 m auf Hauptbahnen mit $v > 100\,km/h$, 700 m auf Hauptbahnen mit $v \leq 100\mathrm{km}/h$ und 400 m auf Nebenbahnen mit $v \leq 80\,km/h$. Um den Zugführer bei kurzen Hauptsignalabständen von 1000 m bis 1300 m nicht mit kurz aufeinander folgenden Haupt- und Vorsignalen zu belasten, wird das Hauptsignal und das folgende Vorsignal an einem Signalmast angebracht. Nur in Sonderfällen ist es erlaubt, den Vorsignalabstand über das 1,5-fache des Regelbremsweges, d.h. mehr als 1500 m zu setzen (vgl. [DB98]). Abbildung 2.14 a zeigt die Signalstandorte bei getrennten Haupt- und Vorsignalen und Abbildung 2.14 b zeigt kombinierte Haupt- und Vorsignale.

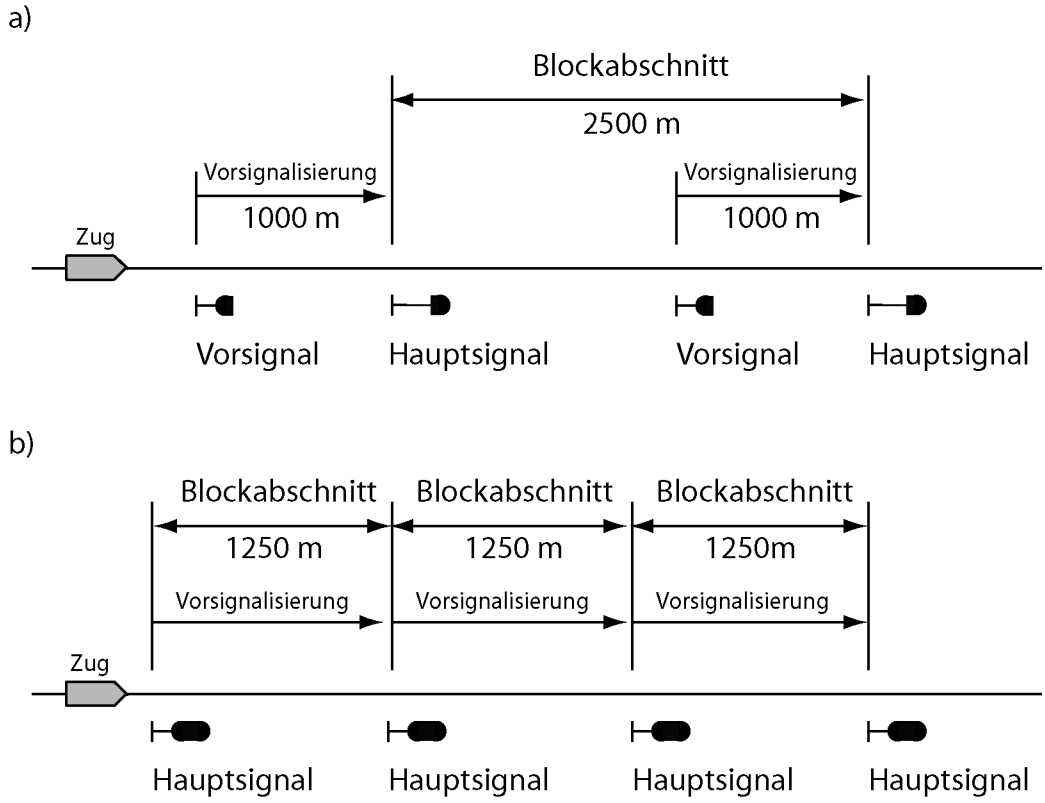

Abbildung 2.14: (a) getrennte Vor- und Hauptsignale, (b) kombinierte Signale

Die maximal zulässige Geschwindigkeit, die ein Zug in Deutschland bei ortsfester Signalisierung fahren darf, beträgt 160 km/h, da die Bremsung bei dieser Geschwindigkeit noch im Vorsignalabstand von 1000 m eingehalten werden kann. Höhere Geschwindigkeiten benötigen einer anderen Zugsicherungstechnik.

2.2.3 Bahnanlagen

In diesem Abschnitt werden Begriffe verschiedener Infrastruktureinrichtungen vorgestellt, welche im Laufe dieser Arbeit verwendet werden. Die Funktion der Bahnanlagen wird in EBO §4 festgelegt.

Blockstellen

Blockstellen sind Bahnanlagen, die einen Blockabschnitt begrenzen. Eine Blockstelle kann zugleich als Bahnhof, Abzweigstelle, Überleitstelle, Anschlussstelle, Haltepunkt, Haltestelle oder Deckungsstelle eingerichtet sein.

Überleitstellen

Überleitstellen sind Blockstellen der freien Strecke, wo Züge auf ein anderes Gleis derselben Strecke übergehen können. Bei zweigleisigen Strecken wird dieser Übergang über Verbindungen, zwischen zwei, parallel verlaufenden Gleisen ermöglicht. Überleitstellen sind mit Blocksignale auszurüsten (Abbildung 2.15).

Abbildung 2.15: Überleitstellen mit einfachen und doppelten Gleiswechsel

Abzweigstellen

Abzweigstellen sind Blockstellen der freien Strecke, wo Züge von einer Strecke auf eine andere Strecke übergehen können. Abzweigstellen sind mit Blocksignale auszurüsten, womit nicht nur die angrenzenden Blockabschnitte der Strecke begrenzt werden, sondern auch die Blockabschnitte der abzweigenden Strecke (Abbildung 2.16).

Haltepunkte

Haltepunkte sind weichenlose Bahnanlagen, an denen Züge planmäßig halten, beginnen oder enden dürfen.

Haltestellen

Haltestellen sind Abzweigstellen, Überleitstellen oder Anschlussstellen, die mit einem Haltepunkt örtlich verbunden sind (Abbildung 2.16).

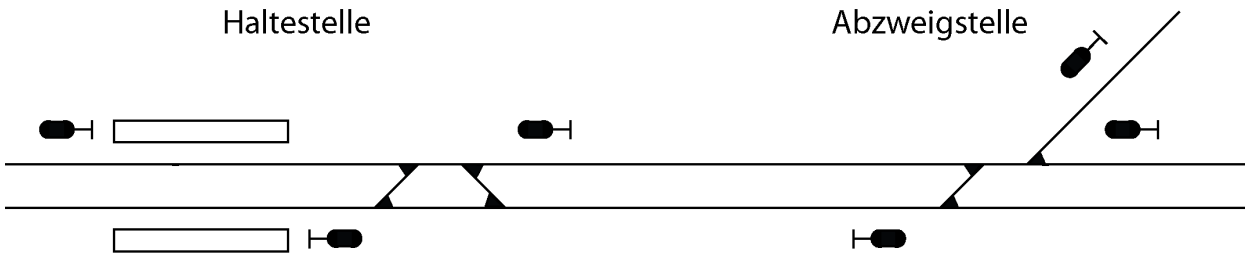

Abbildung 2.16: Eine Haltestelle mit Überleitstelle und eine Abzweigstelle

Deckungsstellen

Deckungsstellen sind Bahnanlagen der freien Strecke, die den Bahnbetrieb besonders an Kreuzungen, beweglichen Brücken, Gleisverschlingungen und Baustellen absichern.

Gleisverschlingungen sind Gleise die bei beengten Verhältnissen zu dicht aneinander liegen und sich somit die Züge berühren würden. Signale, die an Deckungsstellen angeordnet werden, heißen *Deckungssignale*.

2.2.4 Änderungen der Zugfolge

Überholung

Als *Überholung* bezeichnet man das Ausweichen eines Zuges, um einen schnelleren nachfolgenden Zug der gleichen Fahrtrichtung, die Vorbeifahrt zu ermöglichen. Wird der Zug ohne das er Halten muss überholt, wird dies als *„fliegende“ Überholung* bezeichnet. Fliegende Überholungen setzen ein Überholungsgleis voraus, dessen Länge den geplanten Geschwindigkeiten und Zuglängen entspricht.

Kreuzung

Kreuzen wird das Ausweichen zweier in entgegengesetzter Fahrtrichtung fahrender Züge auf einer eingleisigen Strecke genannt. Wird bei entsprechend langem Ausweichgleis das Kreuzen der beiden Züge durchgeführt, ohne dass einer der Züge halten muss, spricht man von einer *„fliegenden“ Kreuzung*.

Bei der Erstellung des Fahrplans wird versucht, Kreuzungen und Überholungen von Zügen mit einem Verkehrshalt im Bahnhof zu kombinieren, um die Haltezeit für Personal- und Fahrgastwechsel nutzen zu können. Zudem wird das Halten in einem Bahnhof von den Passagieren als weniger störend empfunden als das Halten auf freier Strecke.

Begegnung

Eine Begegnung ist die Vorbeifahrt eines Zuges an einem Zug der Gegenrichtung auf einer zweigleisigen Strecke

2.2.5 Der Fahrstraßenbegriff

Damit die sichere Fahrt eines Zuges gewährleistet werden kann, bedient man sich dem Prinzip der so genannten *Fahrstraße*. Eine Fahrstraße wird ein technisch gesicherter Fahrweg genannt. Fahrstraßen, die für Zugfahrten zugelassen werden, werden auch als *Zugstraßen* bezeichnet. *Rangierstraßen* werden hingegen die für Rangierfahrten zugelassenen Fahrstraßen genannt. Im Folgenden wird hier auf die Eigenschaften von Zugstraßen eingegangen, da Rangierstraßen in der

Regel eine abgeschwächte Form der Zugstraßen darstellen und deren Betrachtung für diese Arbeit nur eine untergeordnete Rolle spielen.

Eine Fahrstraße beginnt stets an einem Signal. Für gewöhnlich ist dies ein Hauptsignal. Das Ende einer Fahrstraße ist abhängig davon, ob für den Fahrweg ein Zielsignal vorhanden ist.

Wenn ein Zielsignal vorhanden ist, ist das Ende einer Fahrstraße der dahinter liegende Durchrutschweg. Weil der Durchrutschweg ebenfalls Teil der Fahrstraße ist, werden deswegen Weichen und Kreuzungen die im Durchrutschweg liegen können mit in die Fahrstraßensicherung einbezogen. Ist für eine Fahrstraße kein Zielsignal vorhanden, ist das Fahrstraßenende das Ende eines Fahrstraßenknotens. Ein Beispiel für eine Fahrstraße mit einem Zielsignal ist die für eine Einfahrt in einen Bahnhof gebildete *Einfahrstraße* (Abbildung 2.17 a). Ein weiteres Beispiel sind die *Blockfahrstraßen*. Sie sind die zwischen Blockabschnitte gesicherten Fahrwege auf der freien Strecke (Abbildung 2.17 b).

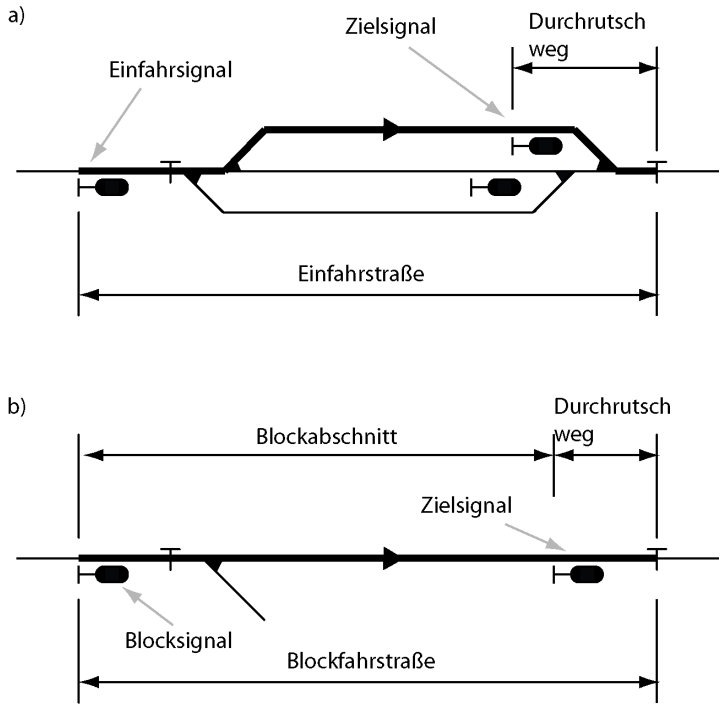

Abbildung 2.17: Fahrstraßen mit Zielsignal

Ausfahrstraßen sind die Fahrstraßen der Ausfahrten aus Bahnhöfe und besitzen kein Zielsignal (Abbildung 2.18). Ausfahrstraßen enden hinter dem Weichenbereich des Bahnhofs. Anstatt eines Zielsignals wird eine Ausfahrstraße durch eine *Fahrstraßenzugschlussstelle* begrenzt, welche hinter der letzten Weiche der Weichenstraße liegt. Die Fahrstraßenzugschlussstelle ersetzt in diesem Fall das eigentliche Blocksignal hinter den Bahnhofsweichen, welches aber aufgrund der kurzen Distanz zum Ausfahrsignal weggelassen wird. Analog zur Signalzugschlussstelle muss ein Zug die

Fahrstraßenzugschlussstelle komplett passiert haben, damit die eingestellte Fahrstraße wieder aufgelöst werden kann. Die direkt hinter der Fahrstraßenzugschlussstelle gebildete anschließende Blockfahrstraße bleibt unabhängig vom Räumen der Ausfahrstraße weiterhin von dem Ausfahrsignal gedeckt.

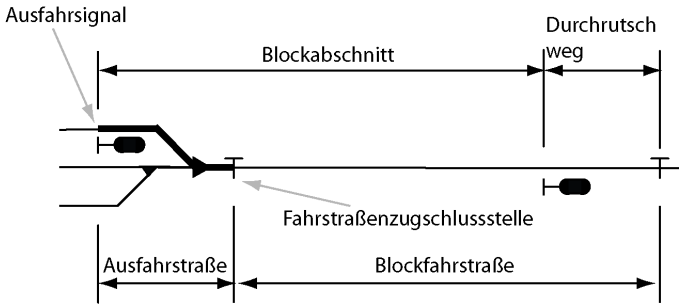

Abbildung 2.18: Eine Fahrstraße ohne Zielsignal

In [Pa04] werden folgende Sicherheitsanforderungen an eine Fahrstraße für eine sichere Zugfahrt aufgelistet:

- vor einer Zugfahrt muss die richtige Endposition aller auf dem Fahrweg liegenden beweglichen Fahrwegelemente (i.d.R. Weichen) festgestellt werden.
- das Umstellen der beweglichen Fahrwegelemente unter dem fahrenden Zug muss verhindert werden.
- die Zulassung von gefährdenden Fahrten muss verboten sein.
- Fahrzeuge müssen daran gehindert werden, anderweitig in den Fahrweg eintreten zu können.
- die Zugfahrt muss verboten werden, wenn der zu befahrene Gleisabschnitt bereits belegt ist.

Um diese erforderlichen Bedingungen einzuhalten, werden bei den Bahnen spezielle technische Verfahren angewendet. Ist bei einfachen Infrastrukturverhältnissen keine entsprechende Technik vorhanden, wird die Sicherung der Fahrstraßen durch betriebliche Maßnahmen ersatzweise gesichert. In der Eisenbahn-Bau- und Betriebsordnung wurde für die Fahrstraßensicherung der Begriff der *Signalabhängigkeit* eingeführt. Dieser besagt, dass eine Fahrstraße genau dann signalabhängig ist, wenn das dem zu sichernden Fahrweg zugehörige Signal nur dann auf Fahrt gestellt werden kann, wenn die im Fahrweg liegenden Weichen gestellt und verschlossen sind. Signalabhängigkeit bedeutet weiterhin, dass die Weichen einer Fahrstraße solange verschlossen bleiben wie das Signal den Fahrtbegriff anzeigt. Die Bedingung des Fahren im festen Raumabstand

besagt jedoch, dass ein sich im Blockabschnitt befindender Zug durch das Halt zeigende Startsignal gedeckt werden muss, um den vorgeschriebenen Folgefahrschutz zu gewährleisten. Daher ist es möglich, das Startsignal der Fahrstraße bereits wieder auf Halt zu setzen, bevor der Zug alle Weichen der Fahrstraße passiert hat. Aus diesem Grund muss die Fahrstraße in einen zusätzlichen *Fahrstraßenverschluss* gebracht werden, der die Weichen erst dann wieder freigibt, wenn der Zug die jeweiligen Weichen passiert hat oder an seinem Zielort angekommen ist. Um die dritte Bedingung zu erfüllen, dürfen Fahrten die sich gegenseitig gefährden nicht gleichzeitig zugelassen werden. Die Fahrstraßen solcher *feindlicher Fahrten* werden als *feindliche Fahrstraßen* bezeichnet. Feindliche Fahrstraßen, welche sich in der Stellung mindestens einer signalabhängigen Weiche unterscheiden, sind aufgrund der Signalabhängigkeit bereits ausgeschlossen. Die Stellung dieser Weiche sorgt in diesem Fall für die ausschließliche Fahrtfreigabe nur einer der beiden Fahrstraßen. Abbildung 2.19 zeigt exemplarisch eine Weiche, die eine Einfahrstraße und eine Ausfahrstraße gegenseitig ausschließt. Aufgrund dessen, dass diese Weiche im Durchrutschweg der Einfahrstraße liegt, muss diese Weiche auch für die Einfahrt gestellt und verschlossen werden. Entweder muss sie für die Einfahrt vom Stammgleis her stumpf befahren werden können oder sie muss für die Ausfahrt vom Abzweiggleis stumpf befahren werden können.

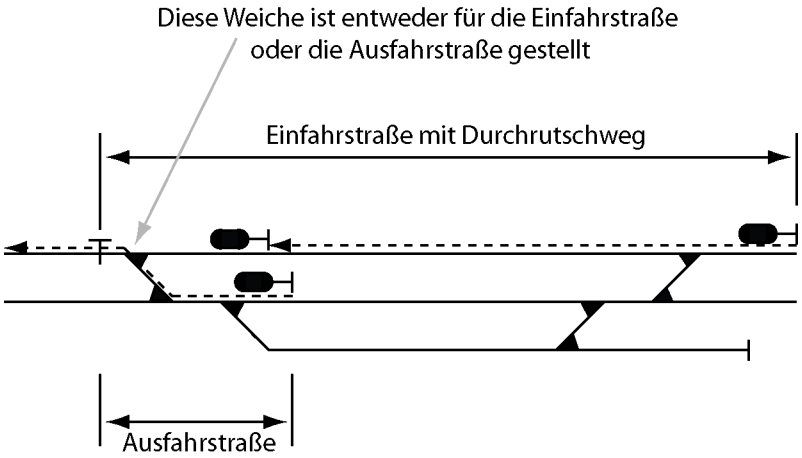

Abbildung 2.19: Ein einfacher Ausschluss

Andere feindliche Fahrstraßen, die sich von Natur aus nicht selbst ausschließen, sind mit speziellen Fahrstraßenausschlüssen zu behandeln. So müssen beispielsweise Gegeneinfahrten auf dasselbe Gleis eines Bahnhofs mit einem Gegenfahrschutz in der Sicherungslogik des Bahnhofsstellwerks verhindert werden. Abbildung 2.20 zeigt einen Bahnhof, in dem solch eine Gegenfahrt auftreten könnte. Das Stellwerk muss in diesem Fall das gleichzeitige Zulassen des Fahrtbegriffes der beiden Einfahrsignale verbieten.

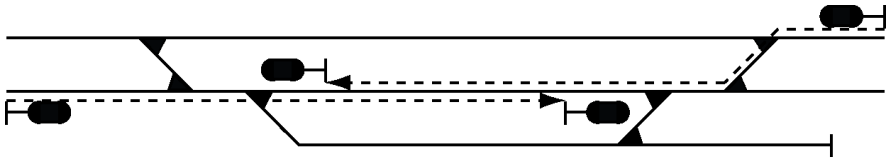

Abbildung 2.20: Ausschluss einer Gegenfahrt, Quelle: [Pa04]

Aus der weiteren Forderung, Fahrzeuge daran zu hindern in den zu sichernden Fahrweg eintreten zu können leiten sich Flankenschutzmaßnahmen ab. Die Aufgabe des *Flankenschutzes* besteht darin, die in die Fahrstraße einmündende Fahrten, auch als *Flankenfahrten* bezeichnet, abzuwenden.
Flankenfahrten sind feindliche Fahrten, unbeabsichtigt abrollende Wagen oder das Strecken von Zügen nach dem Halten. Flankenfahrten die selbst eine Fahrstraße benötigen, können bereits durch den vorher beschriebenen Fahrstraßenausschluss verhindert werden. So bilden beispielsweise die Einfahrstraße und die Ausfahrstraße aus Abbildung 2.19 gegenseitig Flankenfahrten in der in beiden Fahrstraßen liegenden Weiche.
Flankenschutz wird nach [Pa04] in *unmittelbaren Flankenschutz* und *mittelbaren Flankenschutz* eingeteilt. Der unmittelbare Flankenschutz wird durch technische Flankenschutzeinrichtungen realisiert. Diese sind Schutzweichen, Halt zeigenden Signale oder Gleissperren.

- Eine *Schutzweiche* ist eine Weiche außerhalb des zu sichernden Fahrweges, über die eine feindliche Fahrt laufen könnte. Sie wird deshalb während der Fahrstraßenbildung in abweisender Stellung verschlossen um die Flankenfahrt abzuwehren.
- Halt zeigende Signale sind z.B. die Sperrsignale, welche nur für Rangierfahrten gelten und in den Nebengleisen der Bahnhöfe aufgestellt werden.
- *Gleissperren* werden nur in Nebengleisen eingerichtet und würden dort eine Flankenfahrt zum Entgleisen bringen bevor sie in den zu sichernden Fahrweg eintritt. Gleissperren schützen daher vor gefährlichen Rangierfahrten und unbeabsichtigt abrollende Wagen.

Mittelbarer Flankenschutz kann bei mangelnder technischer Ausstattung für Rangierfahrten und unbeabsichtigt ablaufende Wagen durch Rangier- und Abstellverbot durchgeführt werden. Nach dem Halten eines Zuges kann das Strecken des Zuges entweder durch Fahrwegweichen mit Flankenschutzfunktion oder durch Streckschutzlängen auf dem Einfahrgleis gesichert werden. Die Streckschutzlänge stellt eine Verlängerung des Abschnitts hinter der Verzweigungsweiche dar, der vor dem Halt frei gefahren werden muss. Erst wenn diese Streckschutzlänge frei gefahren worden ist, darf die nächste Fahrstraße eingestellt werden. Somit ist sicher gestellt, dass haltende Züge nicht zu nah an der verzweigenden Weiche stehen bleiben. Abbildung 2.21 zeigt eine angeordnete

Streckschutzlänge durch eine Verlängerung des Freimeldeabschnitts um 4 Meter.

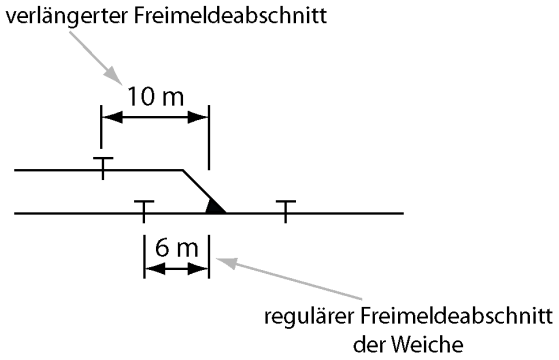

Abbildung 2.21: Streckschutzlänge, Quelle: [Pa04]

Damit eine Zugfahrt letztendlich nur in unbesetzte Gleisabschnitte zugelassen werden darf, muss bei der Fahrstraßensicherung eine *Gleisfreimeldung* stattfinden. Diese Forderung entfällt für Rangierfahrten, weil sie auf Sicht fahren. Die Gleisfreimeldung wird heutzutage hauptsächlich mit technischen Gleisfreimeldeanlagen durchgeführt. Diese Anlagen prüfen unmittelbar vor dem auf Fahrt Stellen des Startsignals der Fahrstraße, ob der vorliegende Fahrweg tatsächlich frei ist. Unter einfachen Verhältnissen wurde die Gleisfreimeldung in Bahnhöfen früher oft durch Hinsehen des Personals geprüft.

3 Modellierung

In diesem Kapitel wird das der Applikation zugrunde liegende Modell erläutert. Ziel dieser Arbeit ist es eine Software zu entwickeln, die Infrastrukturdaten eines angenommenen Eisenbahnnetzes generiert. Die grundlegende Zielsetzung ist es, die Modellierung so abstrakt wie möglich zu gestalten. Dadurch müssen für die Anwendung dieses Programms nur wenige Fachkenntnisse im Eisenbahnwesen vorausgesetzt werden. Weitere Vorteile einer abstrakten Betrachtung sind eine leichtere Wartbarkeit und eine höhere Flexibilität in der Weiterentwicklung der Software. Aufgrund der Anwendung dieses Programms für realitätsnahe Netze ist es teilweise notwendig fachspezifisches und technisches Wissen in die Modellierung mit einfließen zu lassen. An den jeweiligen Stellen wird erläuternd auf dieses Wissen eingegangen. Es wird als zweckmäßig angesehen kontrollierend in die Generierung der Eisenbahninfrastrukturdaten eingreifen zu können. Aufgrund dessen berücksichtigen die Modelle vom Anwender übergebene Parameter bei der Erzeugung der Daten. Die Übergabe der Parameter ist zu dem ein weiterer Grund die im Folgenden beschriebenen Modelle abstrakt zu halten, denn eine zu fachspezifische Modellierung würde den Umfang der erforderlichen Parameter unangemessen steigern.

Aufgrund der Komplexität des Erzeugungsprozesses bietet sich die Unterteilung der Generierung in Teilprobleme an. Die Abarbeitung dieser Teilprobleme wird durch Modelle beschrieben, die im fortschreitendem Generierungsprozess an Abstraktion verlieren. Diese Unterteilung ist eine Voraussetzung für eine flexible Weiterentwicklung dieser Software, denn die Lösungssuche für ein Teilproblem kann durch ein zusätzliches Modell geändert, verbessert oder ersetzt werden. Das Hinzufügen neuer Verarbeitungsschritte zur Erzeugung weiterer Infrastrukturdaten ist auch möglich.

Um Eisenbahninfrastrukturdaten zu generieren, muss zuerst geklärt werden was unter einem Eisenbahnnetz verstanden werden soll. Da das Eisenbahnwesen historisch gewachsen ist, entstanden in vielen Bereichen des Bahnwesens evolutionär gewachsene Sonderformen. Diese sind auch in der Infrastruktur anzutreffen. In [Pa99] wird daher auch die Infrastruktur der Eisenbahn als unsystematisch charakterisiert. In der vorliegenden Arbeit wird in der Modellierung von diesen Sonderformen abstrahiert und es werden nur reguläre Eigenschaften der Infrastruktur berücksichtigt.

Unterteilung des Eisenbahnnetzes

Wie in Kapitel 2.2 beschrieben, wird ein Eisenbahnnetz nach der EBO in Bahnhöfe und freie

Strecken eingeteilt. Eine weitere Unterteilung des Netzes ist nicht genauer definiert worden. Um eine determinierte und effiziente Erzeugung von Infrastrukturdaten zu erreichen, bedarf es weiterer Definitionen. Dafür wird der Begriff eines Knoten im Eisenbahnnetz benötigt. Leider ist dieser Ausdruck in der Eisenbahnwelt nicht einheitlich definiert worden. Diese Arbeit orientiert sich an dem in [Wei02] verwendeten Begriff eines Eisenbahnnetzknoten.

Definition: Demnach sind *Knoten* im Eisenbahnnetz alle Bahnhöfe und Abzweigstellen, die die Strecken sowohl infrastrukturell als auch betrieblich miteinander verknüpfen. Die infrastrukturellen Verknüpfungen geschehen mittels Weichen und die betrieblichen Verknüpfungen stellen Zugübergänge zwischen mehreren Strecken in einem Bahnhof oder an einer Abzweigstelle dar.

Aufgrund dieser Definition werden die Bahnhöfe in Knoten- und Zwischenbahnhöfe unterteilt. **Definition:** *Knotenbahnhöfe* sind die Bahnhöfe, die der vorangegangenen Definition eines Knoten im Eisenbahnnetz genügen. Als Sonderform sollen abweichend von der Definition des Eisenbahnnetzknoten auch Kopfbahnhöfe, die mit genau einer Strecke verbunden sind und diese Strecke begrenzen, als Knotenbahnhöfe gelten. Die Bahnhöfe, die keine Knotenbahnhöfe sind, heißen *Zwischenbahnhöfe*. Zwischenbahnhöfe sind demnach die auf den Strecken liegenden Bahnhöfe ohne Abzweigungen zu weiteren Strecken. Sie werden auch als *Unterwegsbahnhöfe* bezeichnet.

Beschreibung des Eisenbahnnetz durch einen Graphen

Topologisch kann ein Eisenbahnnetz als ein Graph $G=(N, E)$ angesehen werden. $N=\{n_1,\ldots,n_k\}$ ist die Menge der Knoten und $E=\{e_1,\ldots,e_l\}$ die Menge der Kanten des Graphen. Eine Kante entspricht einer freien Strecke des Netzes. Auf den Strecken wird in der Regel im Linienbetrieb gefahren. Aufgrund der Hin- und Rückrichtung des Linienverkehrs werden die Kanten des Graphen G als ungerichtet betrachtet. Der Graph G besitzt keine Schleifen, weil eine Strecke immer zwei unterschiedliche Eisenbahnnetzknoten verbindet. Knoten des Graphen entsprechen den Eisenbahnnetzknoten. Dies sind alle Knotenbahnhöfe und Abzweigstellen. Die *Kardinalität* bzw. der *Grad* eines Knoten $n_i \in N$ wird im Folgenden als $card\,(n_i)$ angegeben. Sie gibt die Anzahl der mit dem Knoten n_i verbundenen Kanten an.

Beispiel 3.1: Abbildung 3.1 zeigt ein Eisenbahnnetz mit fünf Knotenbahnhöfen, einer Abzweigstelle und zwei Unterwegsbahnhöfe. Abbildung 3.2 zeigt den zu diesem Eisenbahnnetz gehörigen Graphen. An diesem Beispiel ist gut zu erkennen, dass die Kanten des Graphen die

modellierten Strecken des Eisenbahnnetzes darstellen und die Knoten entweder Abzweigstellen oder Knotenbahnhöfe sind. Der Knoten n_4 repräsentiert die Abzweigstelle und $n_{1,} n_2, n_3, n_5 \, und \, n_6$ die Knotenbahnhöfe. Die Knoten besitzen die Grade $card\,(n_1)=3, card\,(n_2)=2, card\,(n_3)=2,$ $card\,(n_4)=3, card\,(n_5)=3 \, und \, card\,(n_6)=1$. Der Graph erfasst die zwei Unterwegsbahnhöfe nicht. In der Praxis hat dies den Vorteil, dass der Graph eine wesentlich geringere Komplexität annimmt als ein zugehöriges Eisenbahnnetz, da in der Realität wesentlich mehr Unterwegsbahnhöfe als Knotenbahnhöfe existieren.

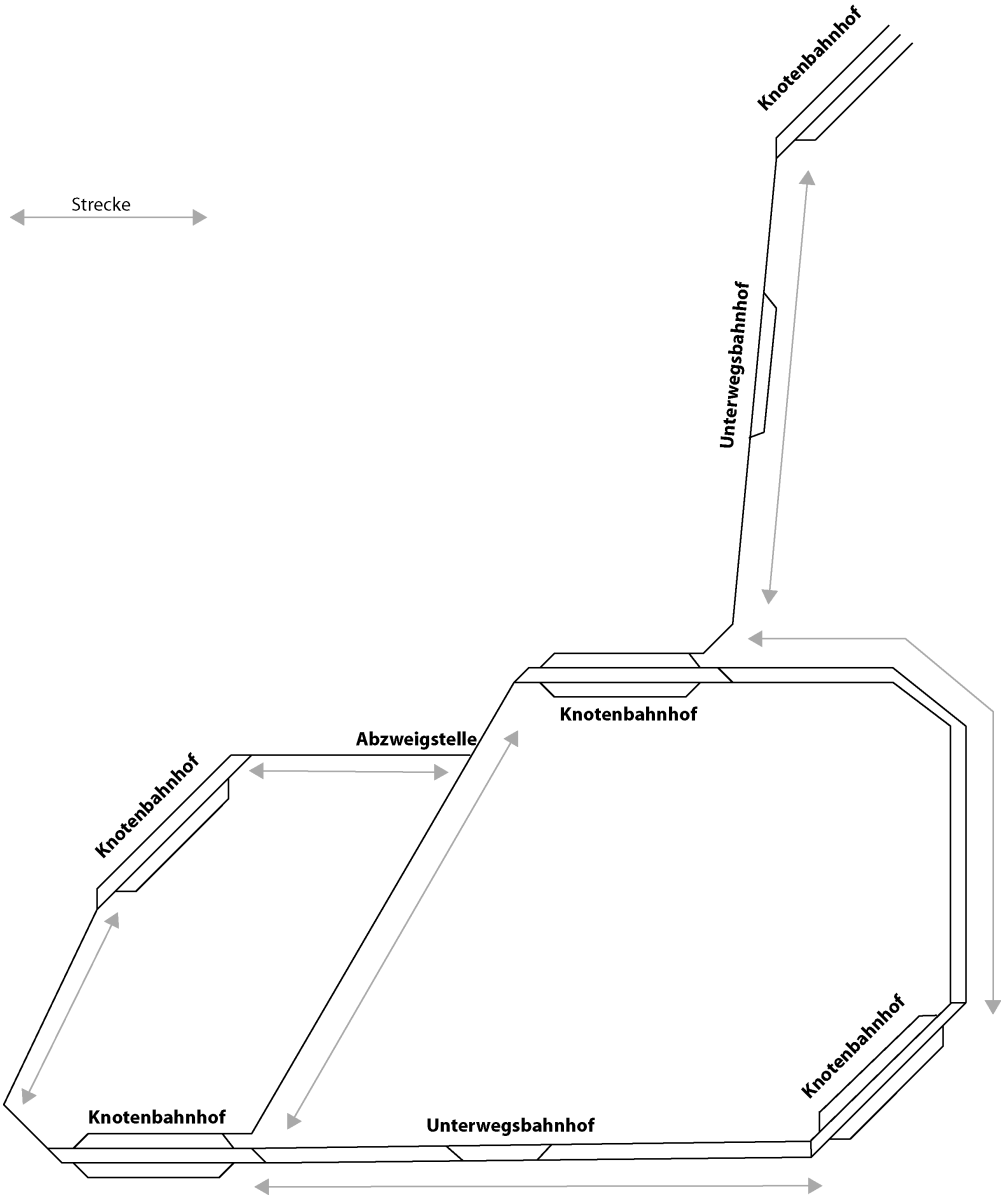

Abbildung 3.1: Ein Eisenbahnnetz

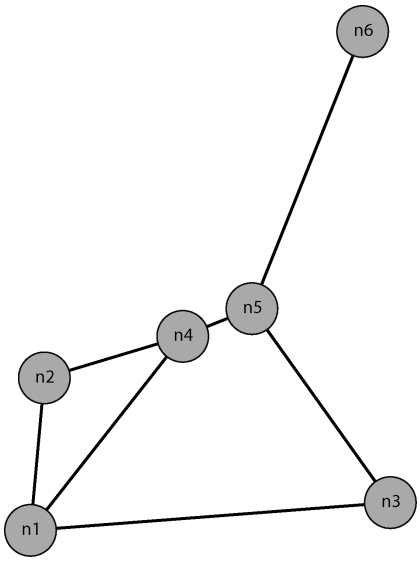

Abbildung 3.2: Der zum Eisenbahnnetz zugehörige Graph

Im fortlaufenden Generierungsprozess werden Daten generiert und den Knoten und Kanten eines Graphen sukzessiv hinzugefügt. Jeder Erzeugungsschritt konkretisiert das Schienennetz mit den erzeugten Daten weiter. Im Folgenden werden die einzelnen Phasen des Generierungsprozesses erläutert.

Der Charakter eines Schienennetzes wird entscheidend von der Topologie seines korrespondierenden Graphen bestimmt. Aus diesem Grund wird in den anschließenden zwei Phasen ein Graph gebildet, der die vom Anwender gewünschten räumlichen Eigenschaften erfüllt. Die Topologie wird durch verschiedene Größen ausgedrückt und kann durch verschiedene Bedingungen eingeschränkt werden. So lässt sich beispielsweise festlegen, ob ein Schienennetz regionales oder landesweites Ausmaß besitzt. Genauso kann die Dichte, die Ausdehnung oder die Gesamtlänge eines Eisenbahnnetzes durch die Topologie spezifiziert werden. Die Anzahl der Knotenbahnhöfe ist ein entscheidender Parameter. Die Generierung der topologischen Eigenschaften des Graphen lässt sich in zwei Teilprobleme formulieren. Zuerst muss eine den Anforderungen entsprechende Positionierung der Knoten des Netzes gefunden werden. Anschließend werden die positionierten Knoten so mit Kanten verbunden, dass die Verbindungen den spezifizierten Kriterien entsprechen. Anschließend wird das Eisenbahnnetz, basierend auf dem Graphen, konkretisiert.

3.1 Positionierung der Knoten

Das erste Teilproblem beschäftigt sich mit der korrekten Positionierung der Knoten eines Netzes.

Die Knoten sind korrekt positioniert, wenn ihre Lage alle angegebenen Kriterien erfüllt. In dieser Arbeit beschränkt sich die räumliche Positionierung des Schienennetzes auf den zweidimensionalen Raum. Es wird daher vorausgesetzt, das sich die Infrastruktur in einer Ebene befindet. Die Betrachtung für den dreidimensionalen Raum wäre ein möglicher Ansatzpunkt für weitere Arbeiten. Für die Betrachtung im zweidimensionalen Raum wird ein kartesisches Koordinatensystem mit diskreten Koordinaten genutzt. Eine Diskretisierung der Koordinaten ist für die Topologie ausreichend, weil das Maß einer Längeneinheit (LE) im Koordinatensystem für eine angemessene Genauigkeit im Nachhinein geändert werden könnte. Die zweidimensionalen Koordinaten werden durch die Angabe der horizontalen und anschließend der vertikalen Koordinatenkomponente angegeben. Der Bereich in dem die Koordinaten positioniert werden, wird durch den Anwender durch zwei Werte spezifiziert. Die beiden Werte geben die horizontale und die vertikale Ausdehnung des Positionierungsbereichs an.

Für die Platzierung der Knoten werden im folgenden Graphen ausschließlich Knotenbahnhöfe betrachtet. Abzweigstellen sind in diesem Fall nicht relevant, da sich ihre Lage in der Regel aus der Lage der Strecken zwischen den Knotenbahnhöfen ergibt. Abzweigstellen werden dazu genutzt, zwei Strecken zu verbinden, die aus ökonomischen und ökologischen[3] Gründen über einen gemeinsamen Streckenabschnitt verfügen.

3.1.1 Die Mindestabstände der Knoten

Ein Kriterium für ein Schienennetz ist der Mindestabstand der Knotenbahnhöfe zueinander. Knotenbahnhöfe befinden sich hauptsächlich in Städten. Aufgrund der Siedlungsstruktur der Ortschaften sind Knotenbahnhöfe in der Regel nicht unmittelbar benachbart, sondern weisen eine gewisse Distanz zueinander auf. Dieses Maß wird vom Anwender als positiver ganzzahliger Parameter übergeben. Die Zahl beschreibt den Mindestabstand in Längeneinheiten im Koordinatensystem. Ein Mindestabstand muss angegeben werden und darf nicht kleiner als Null sein. Ein mit mindestens Null Längeneinheiten angegebener Mindestabstand sorgt gleichzeitig für unterschiedliche Positionen der Knotenbahnhöfe im Koordinatensystem. So werden Überlagerungen der Knoten ausgeschlossen. An dem folgenden Beispiel werden die Bedingungen veranschaulicht.

3 Aus ökonomischer Sicht sollen durch den gemeinsamen Streckenabschnitt Infrastrukturkosten eingespart werden. Aus ökologischem Blickwinkel soll ein Zerschneiden der Landschaft verhindert und Lärmemissionen verringert werden.

Beispiel 3.2: In einem Feld mit einer Breite und Höhe von jeweils acht Längeneinheiten sollen zehn Knoten platziert werden. Dazu soll die Bedingung erfüllt sein, dass jeder Knoten mindestens eine Längeneinheit entfernt zu den restlichen Knoten liegt. Abbildung 3.3 zeigt eine mögliche Lösung zu diesem Problem. Die Knoten werden durch ausgefüllte Kreise mit schwarzen Rahmen dargestellt. Der unausgefüllte schwarze Kreis zeigt den Bereich an, in dem sich, bis auf den Knoten an der Position (2;4), kein weiterer Knoten befinden darf. Die drei schwarzen Pfeile symbolisieren den geforderten einzuhaltenden Mindestabstand zu den drei nächst liegenden Knoten. Die helleren Kreise verdeutlichen die Abstandsbereiche der anderen Knoten.

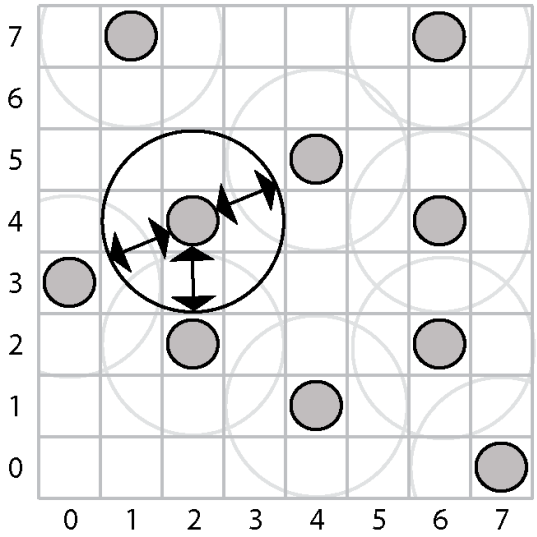

Abbildung 3.3: 10 Knoten mit einem Mindestabstand von einer LE

3.1.2 Knoten in Randbereiche

Weitere Bedingungen können Randzonen sein, in denen sich eine bestimmte Anzahl von Knoten befinden müssen. Diese Randzonen sind gemäß ihrer Lage in einen nördlichen, östlichen, südlichen und westlichen Bereich aufgeteilt. Durch diese Zonen kann eine Ausdehnung des Eisenbahnnetzes erzwungen bzw. verhindert werden. Beispielsweise ist die Modellierung von Ost-West- bzw. Süd-Nord-Korridoren möglich. Sie beinhalten alle Positionen des Feldes, die zwischen dem jeweiligen Feldrand und einer spezifizierten Grenzlinie liegen. Für jede dieser vier Randzonen wird vom Anwender eine Mindestanzahl und eine Höchstanzahl von Knoten angegeben, die sich in dem Bereich befinden sollen. Zu dem wird zu jeder Zone eine Grenzlinie definiert. Diese Linien

bestimmen das Ausmaß ihrer Randzonen. Zweckmäßig erscheint es, die Parameter als relative Maße anzugeben. Die Mindest- und Höchstanzahlen der Knoten werden in relativen Häufigkeiten angegeben. Ein Beispiel wäre die Angabe, dass mindestens 5% und maximal 10% der Knoten im nördlichen Bereich liegen sollen. Somit muss bei der Eingabe nicht auf die Gesamtzahl der Knoten Rücksicht genommen werden. Die Grenzlinien werden als rationale Werte von Null bis Eins übergeben und deuten bei Null auf den kleinsten Wert und bei Eins auf den größten Wert der horizontalen bzw. vertikalen Ausdehnung des Feldes. Beispielsweise würde der südliche Bereich in dem vorher verwendeten 8 mal 8 Feld bei einer Grenzlinie von 25% auf der vertikalen Achse alle Positionen in den Zeilen 0, 1 und 2 enthalten. Zur Veranschaulichung zeigt Abbildung 3.4 das oben abgebildete Beispiel 3.2 mit den zusätzlichen Bedingungen, dass von den zehn Knoten genau ein Knoten im nördlichen, östlichen, südlichen und westlichen Bereich sein soll. Die Randzonen sind so schmal definiert, dass sie nur die äußeren Zeilen bzw. Spalten des Feldes, in der Abbildung grau unterlegt, abdecken. Weil der nördlichere Bereich, dargestellt durch Zeile 7, genau einen Knoten beinhalten darf, stellt die Knotenbelegung aus Abbildung 3.3 mit zwei Knoten in dieser Zeile keine Lösung mehr dar. Die Abbildung 3.4 zeigt eine Lösung, in der sich nur ein Knoten in der siebten Zeile befindet und die Mindestabstandsbedingung weiterhin erfüllt ist. Der Knoten unten rechts auf der Position (7;0) belegt sowohl den südlichen als auch den östlichen Bereich. Ein Knoten darf also zu zwei Randzonen gehören.

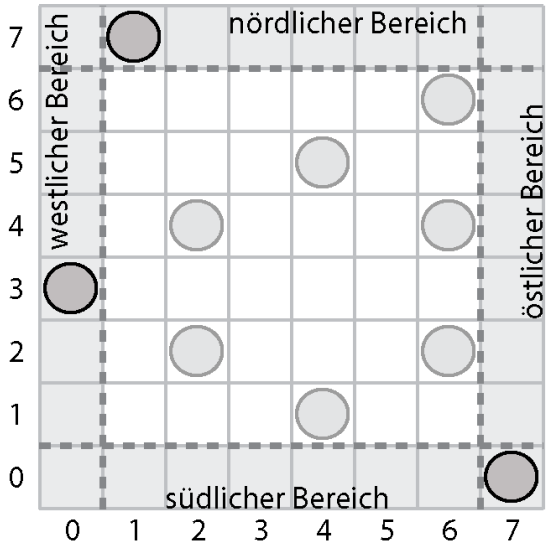

Abbildung 3.4: (Beispiel) Jede Randzone besitzt genau einen Knoten.

3.1.3 Die Ballung von Knoten

Eine weitere topologische Besonderheit in einem Schienennetz ist die lokale Konzentration von Knotenbahnhöfen. Sie zeichnet sich durch eine besonders hohe Dichte von Knoten in einem abgegrenzten Bereich aus. So sind beispielsweise im deutschen Raum in Großstädten wie Berlin und Hannover bei relativ geringer Ausdehnung verhältnismäßig viele Bahnhöfe vorzufinden. In [Wei02] werden diese Gebiete auch Großknotenbereiche genannt. Diese Modellierung erlaubt es, eine beliebige Anzahl von Ballungsgebieten zu definieren. Dabei wird eine Ballung mit der geforderten Anzahl enthaltener Knoten, seiner horizontalen und vertikalen Ausdehnung angegeben. Dadurch ist es möglich, die in der Realität unterschiedlich groß auftretenden Knotenhäufungen abzubilden. Jede Ballung muss mindestens einen Knoten enthalten und eine Breite und Höhe von mindestens einer Längeneinheit aufweisen. Folglich ist somit die Anzahl der möglichen Ballungsgebiete auf die Anzahl der Knoten begrenzt. Außerdem dürfen sich Ballungsgebiete nicht gegenseitig überlappen, um die eindeutige Zuweisung eines Knoten zu einem Ballungsgebiet zu gewährleisten. Im Folgenden wird das vorher verwendete Beispiel genutzt, um eine Forderung nach einer Häufung von Knoten zu demonstrieren. Beispielsweise sollen in einem Gebiet mit einer Breite und Höhe von je 4 Längeneinheiten genau 4 Knoten vorkommen. Dieses Kriterium erfüllt die Lösung aus Abbildung 3.4 nicht. Abbildung 3.5 zeigt eine mögliche Lösung, die alle vorhergehenden Bedingungen und die Häufungsbedingung erfüllt. Der Ballungsraum wird durch ein schwarzes Rechteck dargestellt. Im Unterschied zu der vorhergehenden Lösung liegt der Knoten von Position (4;5) jetzt auf der Position (4;3). Es ist hier anzumerken, dass eine konkrete Lage für den Ballungsraum nicht spezifiziert wurde, aber möglich ist.

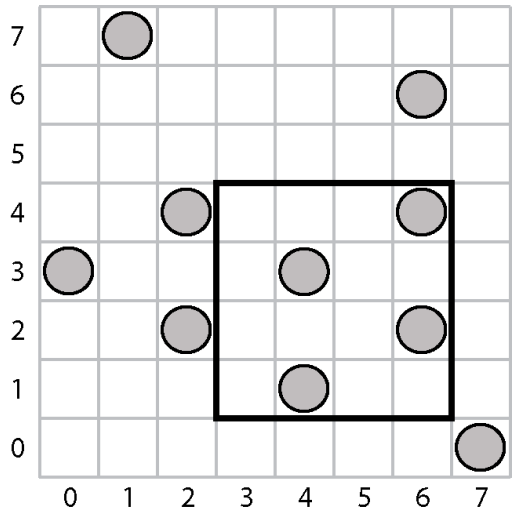

Abbildung 3.5: Eine erzwungene räumliche Konzentration von 4 Knoten.

3.2 Verbinden der Knoten

Nach dem Finden einer geeigneten Positionierung der Knoten stellt sich die Frage, wie die Knoten mit Kanten verbunden werden sollen. Der resultierende Graph soll topologisch einem in der Realität existierenden Eisenbahnnetz ähneln. So ist es unrealistisch einen Knoten mit jeden anderem Knoten potentiell verbinden lassen zu können. Beispielsweise wäre eine direkte Verbindung eines Knotenbahnhofs zu einem anderen Knotenbahnhof wenig realitätsbezogen, wenn an dieser Strecke ein weiterer dritter Knotenbahnhof relativ nah liegen würde, der nicht über diese Strecke zugänglich wäre. Dies würde zum einen den Ausbau und Unterhalt einer zweiten Strecke verlangen und zum anderen könnte dies aufgrund von Umwegen zu einer erheblich verlängerten Reisezeit für den Fahrgast bedeuten. Aus diesem Grund wurde für die Verbindung von Knoten ein Algorithmus entwickelt, der Knoten des Eisenbahnnetzes nach einem räumlichen Kriterium als potentielle Nachbarn klassifiziert.

Definition: Als *Nachbar* eines Knoten n_i wird ein anderer Knoten n_j bezeichnet, der mit n_i durch eine Kante verbunden sein darf.

3.2.1 räumliche Kriterien zur Bestimmung eines Nachbarknoten

Aufgrund dem Anstreben einer optimalen Betriebsweise im Eisenbahnwesen, kann angenommen werden, dass bei der Planung der Infrastruktur versucht wird, ein größtmögliches Dienstleistungsspektrum bei möglichst geringen Kosten zu erzielen. Dies bedeutet, dass versucht wird, möglichst viele Zielbahnhöfe mit so wenig wie möglich Strecken zu bedienen. Dem steht der Ausbau von Umwegen auf Strecken zu Knotenbahnhöfe, die zu weit von der ursprünglichen Route entfernt liegen, gegenüber. Die Konsequenz wäre, dass sich die Fahrtzeit des Zuges vom Anfangsbahnhof zum Endbahnhof aufgrund des Umweges erheblich erhöhen und somit zu einem kundenunfreundlicheren Betriebsablauf führen kann. Aus diesem Grund müssen Kriterien verwendet werden, die determinieren, welcher Knoten mit einem anderen Knoten verbunden werden kann und welcher besser nicht. Um die Modellierung in dieser Phase abstrakt zu halten, wurde ein Bewertungskriterium eingeführt, welches unabhängig von technischen Details des Eisenbahnwesens angewendet werden kann. Das Kriterium und der damit verbundene Algorithmus beziehen sich nur auf die räumliche Lage der Knoten des verwendeten Graphen.

Definition: Seien zwei Knoten n_a und n_b abhängig von ihrer Lage genau dann benachbart, wenn kein weiterer Knoten n_c existiert, der in einem definierten Bereich zwischen n_a und n_b liegt.

Dieser Bereich wird durch einen heuristischen Faktor f und dem euklidischen Abstand zwischen n_a und n_b bestimmt. Der euklidische Abstand zwischen zwei Knoten n_i und n_j wird hier mit $d(n_i, n_j)$ bezeichnet. Der Faktor f soll ein Kriterium darstellen, nach dem zwei Knoten n_a und n_b nicht miteinander verbunden werden sollen, wenn über einen gewissen Umweg ein dritter Knoten n_c auf dem Weg dazwischen liegt. Der Umweg wird als ein prozentualer Anteil des direkten Weges $d(n_a, n_b)$ bewertet. Unterschreitet die Verbindung $n_a - n_c - n_b$ diesen Faktor f,könnte der Knoten n_a mit n_c und n_c mit n_b verbunden werden, anstatt n_a und n_b direkt miteinander zu verbinden. Beispielsweise würde $f = 1{,}25$ eine direkte Verbindung zwischen n_a und n_b verbieten, wenn es einen Knoten n_c gibt, der zwischen n_a und n_b liegt und die Länge der Verbindung $n_a - n_c - n_b$ die Länge der direkten Verbindung $n_a - n_b$ um 25% nicht überschreitet. Ein Knoten n_c liegt zwischen zwei Knoten n_a und n_b, wenn die Abstände $d(n_a, n_c)$ und $d(n_c, n_b)$ kleiner als der Abstand $d(n_a, n_b)$ sind. Diese Bedingung ist notwendig, da sonst keine eindeutige Zuordnung erfolgen kann, welcher Knoten zwischen zwei anderen Knoten liegt. Bei gleichen Abständen $d(n_a, n_c) = d(n_a, n_b)$ könnten somit n_c die Verbindung $n_a - n_b$ und n_b die Verbindung $n_a - n_c$ gleichzeitig verhindern. Dadurch würde möglicherweise zu dem Knoten n_b keine Verbindung hergestellt werden können. Abbildung 3.6 zeigt den grau gefüllten Bereich, in dem kein Knoten liegen darf, damit zwei Knoten n_a und n_b direkt verbunden werden können. Knoten, die in diesem Bereich liegen und mit denen n_a und n_b verbunden wären, bilden einen Umweg, der kleiner gleich der 1,25 fachen Länge des direkten Weges von n_a nach n_b entsprechen würde. Der Knoten n_e würde einen Umweg von mehr als 125% des Direktweges betragen und verhindert aufgrund dessen die Möglichkeit einer Nachbarschaft von n_a und n_b nicht. Ein Umweg über den Knoten n_d ist zwar kleiner als $1{,}25 * d(n_a, n_b)$, weil aber $d(n_a, n_c) = d(n_a, n_b)$ gilt, stellt dieser Punkt auch keinen Grund für das Verbot für der Direktverbindung $n_a - n_b$ dar. Der Knoten n_c in dieser Abbildung liegt am Rand des Bereiches und der daraus ergebene Umweg besitzt eine Länge von $d(n_a, n_c) + d(n_c, n_b) < 1{,}25 * d(n_a, n_b)$. Zusätzlich gilt $d(n_a, n_c) < d(n_a, n_b)$ und $d(n_c, n_b) < d(n_a, n_b)$. Deswegen soll keine Direktverbindung zwischen n_a und n_b hergestellt werden können, da eine geeignete Verbindung über

einen zusätzlichen Knoten, hier n_c, mit relativ geringem Mehraufwand bewerkstelligt werden könnte.

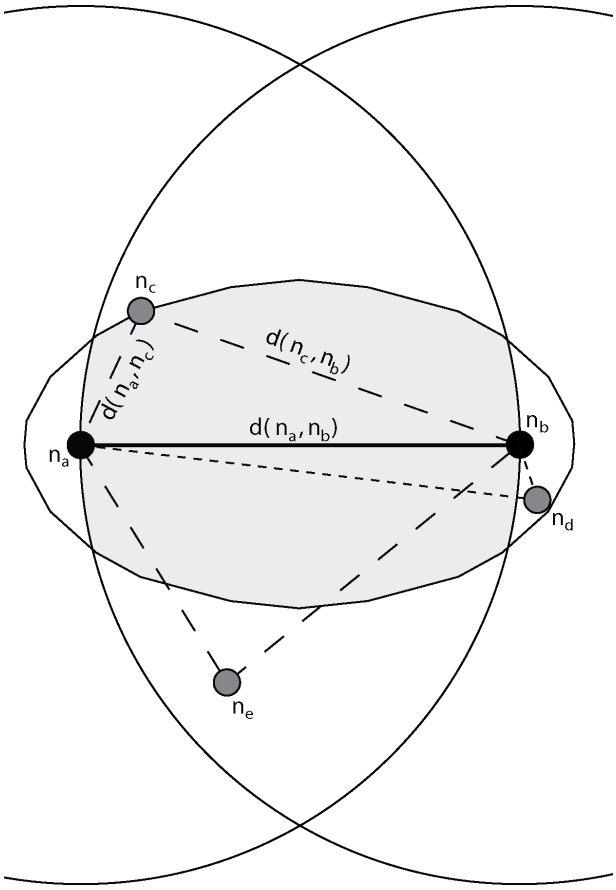

Abbildung 3.6: Darstellung des Bereiches für Umwege bis 125% des Direktweges

Sei $\{n_1, \ldots, n_m\}$ die Menge der Knoten und f der an verwendete Faktor. Dann kann das vorher beschriebene Kriterium mathematisch folgendermaßen angegeben werden.

$$a \in \{1,\ldots,m\}, b \in \{1,\ldots,m\}, n_a \text{ ist mit } n_b \text{ nicht benachbart} \Leftrightarrow \exists c \in \{1,\ldots,m\}, c \neq a, c \neq b:$$
$$d(n_a,n_b) > d(n_a,n_c) \ \wedge \ d(n_a,n_b) > d(n_c,n_b) \ \wedge \ f * d(n_a,n_b) > d(n_a,n_c) + d(n_c,n_b)$$

Das Kriterium ist nur für einen Faktor $1 < f < 2$ sinnvoll, weil $2 * d(n_a,n_b) > d(n_a,n_c) + d(n_c,n_b)$ wegen $d(n_a,n_b) > d(n_a,n_c)$ *und* $d(n_a,n_b) > d(n_c,n_b)$ immer gelten würde und $1 * d(n_a,n_b) \leq d(n_a,n_c) + d(n_c,n_b)$ aufgrund der Dreiecksungleichung gilt. Ein Faktor $f \leq 1$ würde das Kriterium immer fehlschlagen lassen und somit für jedes Knotenpaar eine Kante erlauben. Die Implementierung zu diesem Kriterium befindet sich in Kapitel 4.4.1.

3.2.2 Klassifizierung der Knoten des Graphen

Um ein Eisenbahnnetz genauer spezifizieren zu können bedarf es weiterer Kriterien. So wird die

Infrastruktur maßgeblich von der Komplexität ihrer Bahnhöfe bestimmt. Es bietet sich eine Klassifizierung der Knoten an, die dem Anwender weitere Parameter für die Beschreibung seines gewünschten Eisenbahnnetzes liefern. Ein Problem für die Modellierung stellt die Wahl dar, nach welchem Attributen die Knoten klassifiziert werden sollen. So wird in der Literatur beispielsweise von kleinen, mittleren und großen Bahnhöfen gesprochen ohne genau zu definieren nach welchen Kriterien diese Unterteilung erfolgt. In [Wei02] werden für die Klassifizierung von Knoten einige Kriterien vorgeschlagen. So wird u.a.

- die Anzahl zulaufender Strecken
- die Anzahl von Weichen und Gleisen
- die Summe der Zugfahrten im Bahnhof
- die Bedeutung für den Schienenpersonennahverkehr, Schienenpersonenfernverkehr und Schienengüterverkehr

als Klassifizierungsparameter für die Knoten angegeben. Da sich einige dieser Kriterien nur durch sehr umfangreiche Eingabedaten zur Unterteilung nutzen lassen und dies weitere Modelle nach sich ziehen würde, welche den Umfang dieser Arbeit übersteigen würden, wurde die Anzahl zulaufender Strecken verwendet. Beispielsweise würde die Klassifizierung der Knoten nach der Anzahl ihrer Zugfahrten die Ermittlung und Spezifizierung eines Betriebsprogramms fordern. Nach diesem Betriebsprogramm könnten dann die geplanten Zugfahrten durch die Knoten als Eingabedaten für die Klassifizierung genutzt werden. Die Klassifizierung nach solchen zusätzlichen Kriterien könnte ein Ansatzpunkt für weitere Arbeiten sein.

Die Anzahl zulaufender Strecken eines Knoten stellt ein wichtiges Kriterium für die Komplexität der Infrastruktur dar. So bestimmt diese Anzahl die Komplexität des Knotenbahnhofs selbst und beeinflusst die Dichte des Eisenbahnnetzes. Dies wirkt sich wiederum auf die Leistungsfähigkeit der Infrastruktur aus, da Knoten mit geringer Anzahl an Strecken weniger Alternativrouten für Verkehrsströme anbieten und deswegen eventuell Umwege in Kauf genommen werden müssen. Zudem erhöht sich die Anzahl der Züge auf einer Strecke, was durch Verspätungen zu weiteren Leistungseinbußen führen kann. Aufgrund dessen soll es dem Nutzer möglich sein, das zu generierende Schienennetz durch Angabe der Vorkommenshäufigkeit von Knoten mit unterschiedlicher Anzahl anliegender Strecken zu beschreiben. Die Vorkommenshäufigkeiten werden durch relative Häufigkeiten angegeben. So kann abhängig von der Knotenanzahl, ein Anteil von Knoten mit einer bestimmten Kardinalität verlangt werden. Für eine geforderte Kardinalität c

wird ein geforderter prozentualer Mindestanteil min_c und ein Maximalanteil max_c der Knoten übergeben. Beispielsweise könnte mit der Angabe von $min_6 = 10\,\%$ und $max_6 = 20\,\%$ gefordert werden, dass jeder zehnte bis fünfte Knoten mit sechs zulaufenden Strecken einen Bahnhof einer größeren Stadt repräsentiert. Die Angabe von Knoten mit hohen Kardinalitäten sorgt zugleich für eine Verdichtung des Graphen. Eine Ausdünnung eines Graphen durch die Angabe von Knotenkardinalitäten wird an das auf Seite 40 eingeführte Beispiel 3.2 demonstriert. Nachdem in diesem Beispiel die Knoten positioniert worden sind und berechnet worden ist, welche Knoten miteinander verbunden werden können, sollen unter den möglichen Lösungen die Knoten zu verbinden, nur diejenigen gelten, welche folgende Constraints erfüllen. Von den zehn Knoten sollen zwei Knoten genau eine Kante haben. Beliebig viele Knoten sollen jeweils zwei Kanten aufweisen und mindestens ein und maximal zwei Knoten sind jeweils inzident mit drei Kanten. Die Beschreibung für zehn Knoten ist $min_1 = 20\,\%$ und $max_1 = 20\,\%$, $min_2 = 0\,\%$ und $max_2 = 100\,\%$ sowie $min_3 = 10\,\%$ und $max_3 = 20\,\%$. Abbildung 3.7 zeigt eine mögliche Lösung, die diese Anforderungen erfüllt. Die Kanten des Graphen werden durch schwarze Linien dargestellt. Die Kardinalitäten stehen in den Knoten. Die gestrichelten grauen Linien veranschaulichen weitere potentielle Kanten, die das im vorherigen Abschnitt erläuterte Kriterium bei einem Faktor $f = 1{,}25$ erfüllen. Die gestrichelten grauen und durchgängig schwarzen Linien bilden alle möglichen Kanten, aus denen für weitere Lösungen Kanten zum Verbinden ausgewählt werden. Für die weitere Verarbeitung werden nur die Kanten des Graphen berücksichtigt.

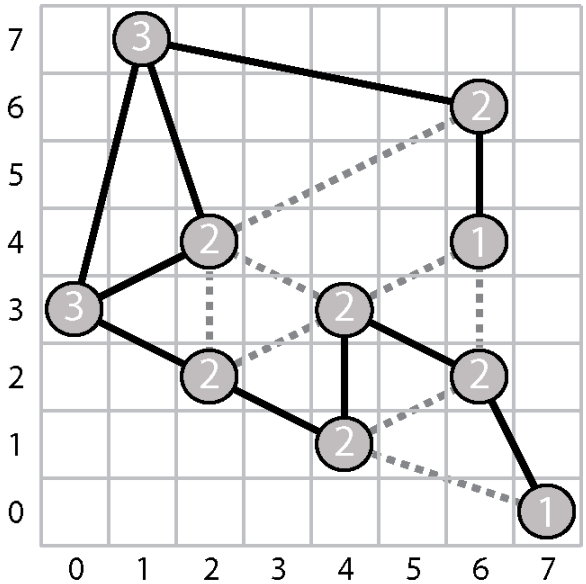

Abbildung 3.7: Ein Graph, der die geforderten Kardinalitäten einhält

3.3 Klassifizierung der Kanten des Graphen

Nach der Festlegung der Topologie des Eisenbahnnetzes, wird eine genauere Beschreibung der Infrastrutukurmerkmale der Strecken benötigt. Aufbauend auf dieser Beschreibung können die Daten der für diese Strecken notwendigen Infrastrukturelemente erzeugt werden. Da sich die Infrastruktur der Strecken aufgrund von Streckenparameter herleiten lässt, muss dem Anwender eine geeignete Angabe der im Netz auftretenden Strecken und seinen Parametern möglich sein. Weil die Anzahl der im Netz verwendeten Strecken sich aus der Generierung des Graphen ergibt, sollen die Parameter nicht für jede Strecke einzeln angegeben werden. Eine komfortable Lösung ist die Klassifizierung von Strecken, welche gleiche Parameter besitzen. Die Parameter einer Klasse schaffen zu dem die Möglichkeit weitere Regeln für die Generierung der Infrastruktur auf Basis dieser Parameter einzusetzen. Zwei wesentliche Parameter einer Strecke, die sich unmittelbar auf die Gestaltung der Infrastruktur auswirken, sind die zulässige Höchstgeschwindigkeit und die Typen der Verkehrsströme einer Strecke. Die zulässige Geschwindigkeit bestimmt unter anderen die Form der Signalisierung sowie das Vorhandensein und die Position spezieller Sicherungseinrichtungen bzw. das Nichtvorhandensein bestimmter Bahnanlagen und die Wahl der Fahrwegelemente auf der Strecke. Beispielsweise dürfen sich in Deutschland ab einer zulässigen Geschwindigkeit von über 160 km/h keine Bahnübergänge an den Strecken befinden. Auf Strecken bis zu einer Geschwindigkeit von 160 km/h werden die Züge durch ortsfeste Signale gesichert. Die Züge fahren auf diesen Strecken im festen Raumabstand (siehe Abschnitt 2.2.2). Für Geschwindigkeiten mit mehr als 160 km/h wird aufgrund zu langer Bremswege ein anderes Sicherungsverfahren angewandt. In Deutschland wird dieses Verfahren als so genannte *Linienzugbeeinflussung* (LZB) bezeichnet. Dieses Verfahren entspricht dem Fahren im absoluten Bremswegabstand (siehe Abschnitt 2.2.1) und bedarf daher keiner ortsfesten Signalisierung. Die Fahrtinformationen werden bei diesen Zügen, i.d.R. schnelle Fernverkehrszüge, durch entsprechende Fahrwegelemente kontinuierlich in den Führerraum des Zuges übertragen. In der Praxis kommen Strecken vor, die sowohl für das Fahren mit ortsfester Signalisierung als auch mit LZB ausgerüstet sind. Weiterhin bestimmt die angestrebte Geschwindigkeit die Dimensionierung der Kurven- und Weichenradien einer Strecke (siehe Tabelle 1, Seite 21). Die Art der Verkehrsströme beeinflusst aufgrund typspezifischer Eigenschaften u.a. die Anzahl und Länge der Gleise einer Strecke, das Vorhandensein spezieller Gleisanordnungen und Sicherungseinrichtungen. So sind Güterzüge in der Regel wesentlich langsamer als Fernverkehrszüge, wodurch Güterzüge im Mischbetrieb bei Überholungen auf ein entsprechend langes Gleis ausweichen müssen. In der EBO werden auf

Strecken mit Reisezügen strengere Flankenschutzmaßnahmen und die damit verbundenen Flankenschutzeinrichtungen gefordert als auf Strecken, wo ausschließlich Güterzüge fahren. Weiterhin bestimmt der Grad der Priorisierung bestimmter Zugarten die geforderte Leistungsfähigkeit ihrer Strecken, welche sich u.a. durch die Anzahl ihrer vorhanden Gleise widerspiegelt. In dieser Arbeit werden die Strecken in verschiedenen Klassen eingeteilt. Dabei sollen einzelne Eigenschaften dieser Klassen möglichst unabhängig von anderen Parametern beschreibbar sein. Aus diesem Grund werden die Klassen mit Nummern versehen auf die sich die Angabe von Eigenschaften bestimmter Strecken bezieht. Somit ist eine getrennte Spezifizierung verschiedener Eigenschaften einer Klasse möglich. Die Realisierung dieses Modells wird in Abschnitt 4.5 diskutiert.

3.3.1 Häufigkeiten der Streckentypen

Die Verteilung von Streckenklassen charakterisiert das resultierende Eisenbahnnetz maßgeblich, weil spezielle Streckeneigenschaften häufig auftreten können. Aus diesem Grund wird die Angabe der Vorkommen der Streckenklassen durch den Anwender geregelt. Dies geschieht analog zu der Beschreibung von Häufigkeiten der Kardinalitäten der Knoten. Demnach wird zu einer Streckenklassse c die relative Häufigkeit seiner repräsentierenden Kanten durch eine Mindestgrenze min_c und einer Höchstgrenze max_c angegeben. Abbildung 3.8 zeigt einen Graphen mit 20 Knoten und 27 Kanten. Die Kanten gehören zu jeweils einem der drei Streckentypen $c \in \{1,2,3\}$. Für die Streckentypen sollen $min_1 = 60\%$, $max_1 = 70\%$, $min_2 = 25\%$, $max_2 = 30\%$ sowie $min_3 = 5\%$ und $max_3 = 10\%$ gelten. Aus der Abbildung ist zu erkennen, dass die dünnen Kanten von Typ 1 die Mehrheit der Kanten darstellen.

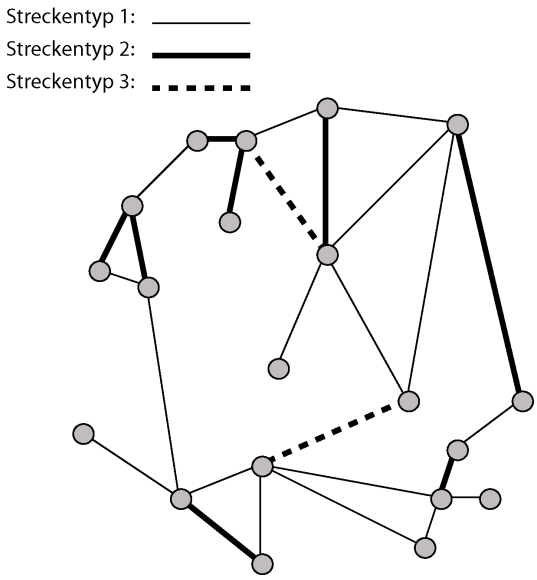

Abbildung 3.8: Eine Graph mit drei Kantentypen

3.3.2 Kantenzüge eines Streckentyps

Die bisherige Modellierung beschreibt die Einteilung der Strecken in Klassen und ihre Verteilung im gesamten Netz. Bei der Lösungsfindung können deswegen vereinzelt Kanten eines Streckentyps auftreten, die an keiner weiteren Kante diesen Typs angrenzen. Dies tritt auch in der Praxis auf, ist bei der Generierung nicht immer erwünscht. In Abbildung 3.8 sind beispielsweise die Kanten des Streckentyps 3 voneinander getrennt. Strecken werden in der Praxis aufgrund geplanter Zuglinien gebaut und entsprechend dem gewünschten Fahrweg entlang verlegt und ausgerüstet. Aufgrund der historischen Entwicklung eines Eisenbahnnetzes, welche durch zeitgemäße Anforderungen an die Infrastruktur und Änderungen in den Fahraufträgen beeinflusst wird, entstehen durch Neu- und Ausbauten sowie der Entfernung von Strecken Infrastrukturgebilde, an denen nicht mehr jeder ursprünglich gebaute Streckenzug erkennbar ist. Da sich Änderungen in der Infrastruktur nach aktuellen Betriebsprogrammen der Bahnunternehmen richten, werden vereinzelt Strecken nach den aktuellen Linien ausgerichtet. Deswegen treten in Eisenbahnnetzen zum einen Strecken gleichen Typs auf, die von Zügen in ihren geplanten Fahrwegen nacheinander befahren werden und zum anderen existieren einzelne Streckenabschnitte, die sich in ihrer infrastukturellen Ausstattung von anliegenden Strecken unterscheiden. Abhängig von dem gewünschten Eisenbahnnetz soll es dem Nutzer möglich sein, den Grad der Kombination von Streckenzügen und vereinzelt auftretenden Strecken eines Typs selbst zu wählen.

Aus diesem Grund kann der Anwender gewünschte Kantenzüge spezifizieren. Ein Kantenzug wird durch den gewünschten Streckentyp, seiner Länge und die mit ihm verbundenen Eigenschaften bestimmt. Die Länge wird durch die Anzahl aufeinander folgender Kanten angegeben. Die Eigenschaften, die mit einem Kantenzug gefordert werden, könnten komplex sein und können in weiteren Arbeiten modifiziert werden. Im Rahmen dieser Arbeit sollen bei der Spezifizierung eines Streckenzuges optional Verkehrstypen und weitere Streckentypen angegeben werden, die mit dem angegebenen Streckentyp kompatibel sind. Mit Spezifizierung der Verkehrstypen wird hier die Festlegung gemeint, ob Schienenpersonennahverkehr, Schienenpersonenfernverkehr oder Schienengüterverkehr auf diesen Strecken explizit erlaubt oder verboten werden soll. Bei der Angabe von kompatiblen Streckentypen, wird ein Toleranzspielraum geschaffen, in dem ein Streckenzug als richtig platziert gilt, obwohl Teile des Kantenzuges nicht dem spezifizierten Streckentyp entsprechen, sondern den der kompatiblen Streckentypen. Diese Kompatibilität ist z.B. dann erwünscht, wenn sich ähnliche Streckentypen in unterschiedlichen Klassen befinden, aber in der Realität aus beispielsweise ökonomischen Gründen trotzdem auf einer Linie verwendet werden. Es könnten z.B. Streckenzüge spezifiziert werden, die aus mehreren Hochgeschwindigkeitsstrecken bestehen, in denen aber Streckenabschnitte zugelassen sein sollen, die in ihrer zulässigen Geschwindigkeit variieren. Dieser Toleranzspielraum sorgt außerdem für eine leichtere Suche nach einer Lösung, da ein zu platzierender Kantenzug im Graphen nicht nur auf einen Streckentyp beschränkt ist.

Neben der Angabe geforderter Kantenzüge soll auch noch deren Ausrichtung im Graphen kontrolliert werden. Grundsätzlich ist es möglich, dass sich der Anfangsknoten und Endknoten eines Kantenzuges räumlich nicht voneinander entfernen. Dies wäre dann der Fall, wenn der Kantenzug einen Kreis bilden würde. Eine in der Praxis eingesetzte Zuglinie besitzt aber für gewöhnlich eine gewisse Distanz zwischen ihrem Start- und Zielbahnhof. Deswegen wird ein Faktor f mit $f \in \mathbb{R}, f > 0$ eingeführt, der bestimmt wie weit ein Knoten im Kantenzug zum Anfangsknoten mindestens entfernt sein soll. Seien $\{n_1, \ldots, n_k\}$ Knoten eines Kantenzuges und n_1 der Knoten, an dem der Kantenzug beginnt. Der Faktor f bestimmt für einen Knoten n_i, dass sein Abstand zum Anfangsknoten n_1 mindestens das f-fache des Abstandes des Vorgängerknotens n_{i-1} zum Anfangsknoten sein muss. Mit Abstand ist hier der euklidische Abstand gemeint. Demzufolge würde der Faktor $f = 1$ wegen $d(n_1, n_i) \geq d(n_1, n_{i-1}) * 1$ einen Kantenzug fordern, in dem jeder Knoten mindestens so weit entfernt vom Anfangsknoten ist wie sein Vorgängerknoten. Ein Faktor $f < 1$ würde es zulassen, dass Knoten für den Kantenzug gewählt werden könnten, die zu einem

gewissen Grad näher an dem Anfangsknoten liegen als ihr Vorgängerknoten. $f > 1$ fordert hingegen eine stärkere Ausdehnung des Kantenzuges. Abbildung 3.9 veranschaulicht dies für den Faktor $f = 1$. Durch den Abstand $d(n_1, n_{i-1})$ vom Anfangsknoten n_1 zum Knoten n_{i-1} können für den weiteren Kantenzug nur noch die Kanten ausgewählt werden, deren Endknoten sich im dargestellten grauen Bereich befinden. Die breiten schwarzen Linien deuten den Kantenzug an.

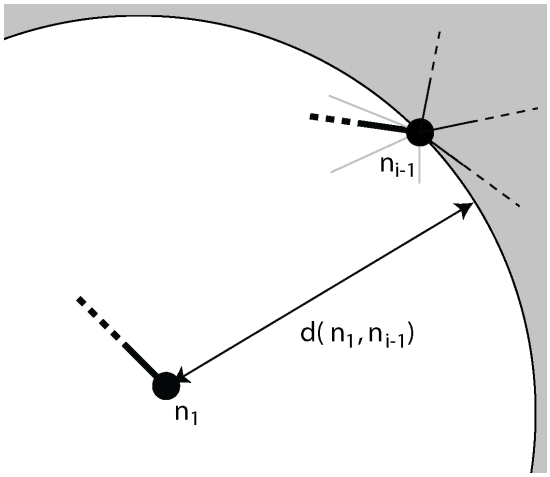

Abbildung 3.9: Ausdehnung des Kantenzuges durch einen Faktor

Der realisierte Algorithmus wird in Abschnitt 4.5.1 diskutiert.

3.4 Konkretisierung der Streckenparameter

In diesem Abschnitt des Generierungsprozesses werden unterschiedliche Parameter für eine Strecke festgelegt. Die Parameter sollen als Grundlage für die Erweiterung der generierten Daten dienen. Einige Parameter sind vom Streckentyp abhängig und werden somit durch den vorher festgelegten Streckentypen beeinflusst. Auf die in dieser Arbeit modellierten Streckeneigenschaften wird im Folgenden eingegangen. Die Beschreibung der Generierung der Streckenparameter wurde in einem Abschnitt zusammengefasst, weil sich ihre Erzeugung teilweise gegenseitig bedingen. Für die Implementierung dieses Modells ist keine zusätzliche Beschreibung nötig und daher wird auf einen Abschnitt im Implementierungskapitel verzichtet.

Die Anzahl der parallel verlaufenden Gleise

Zuerst stellt sich die Frage, wie viele Gleise einer Strecke zugeordnet werden. Dafür existieren keine klaren Aussagen. Gleise die parallel verlaufen, können auch als mehrere parallel verlaufende Strecken betrachtet werden. Bei der deutschen Bahn wird beispielsweise zwischen ein- und zweigleisigen Strecken unterschieden. Bei mehr als zwei Gleisen werden diese als getrennte

Strecken angesehen. Trotzdem wird im Eisenbahnwesen oft auch von drei- oder viergleisigen Strecken gesprochen. Beispielsweise wird in [Pa98] speziell auf dreigleisige Strecken eingegangen. Um das in dieser Arbeit verwendete Modell flexibel zu halten, können einer Strecke prinzipiell beliebig viele Gleise zugeordnet werden. Da in der Realität aber nur sehr selten mehr als vier Gleise auf einem Streckenabschnitt gebaut worden sind, wird der Fokus auf Strecken mit bis zu vier Gleisen gelegt. Ein Gleis wird auch als Spur bezeichnet. Für die Angabe der Anzahl parallel verlaufender Gleise einer Strecke wird für jede Kante im Graph eine Variable verwendet über die Constraints formuliert werden können. Ein typischer Anwendungsfall ist die Dimensionierung der Gleisanzahl einer Strecke abhängig von dem Streckentyp. Ist eine Strecke beispielsweise von geringer Bedeutung für den Verkehr, werden ihre Kosten durch wenig Gleise gering gehalten. Besteht auf einer Strecke hoch priorisierter Verkehr, wird versucht, den Verkehr durch mehrere Spuren flüssig zu betreiben. Aus diesem Grund soll es möglich sein, die Anzahl der Gleise eines Streckentyps nach gewünschten Häufigkeiten formulieren zu können. Die Häufigkeiten sollen jeweils für den Anteil von ein-, zwei-, drei- und viergleisigen Strecken eines Typs spezifiziert werden können. Nach [Fie05] sind beispielsweise die Strecken der Nebenbahnen in der Regel nur eingleisig. Sei eine Strecke vom Streckentyp s, dann werden für sie die Werte $min_{s,t}$ und $max_{s,t}$ mit $t \in \{1,2,3,4\}$ angegeben. $min_{s,t}$ bzw. $max_{s,t}$ bezeichnen die relative Häufigkeit, mit der t-gleisige Strecken des Typs s mindestens bzw. höchstens vorkommen dürfen. Wichtig ist, dass $min_{s,1}+min_{s,2}+min_{s,3}+min_{s,4} \leq 100\,\%$ und $max_{s,1}+max_{s,2}+max_{s,3}+max_{s,4} \geq 100\,\%$ gilt, damit eine mögliche Verteilung der Streckentypen gefunden werden kann. Abbildung 3.10 zeigt einen Graphen, der ein Eisenbahnnetz mit genau einem Streckentyp repräsentiert. Die Strecken dieses Streckentyps sollen mindestens 60% und höchstens 70% eingleisig, 20% bis 40% zweigleisig, 0% bis 10% dreigleisig und 0% bis 10% viergleisig sein. Dies ist erfüllt, weil von den zehn Strecken im Netz sieben eingleisig, zwei zweigleisig und eine viergleisig ist.

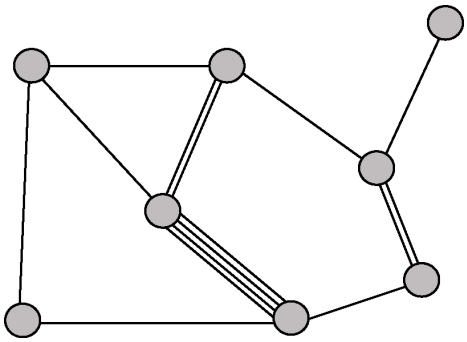

Abbildung 3.10: Ein Eisenbahnnetz mit ein-, zwei- und viergleisigen Strecken

Verkehrstypen einer Strecke

Bereits im vorhergehenden Abschnitt wurde die Bedeutung der auf einer Strecke betriebenen Verkehrstypen für die Infrastruktur beschrieben. In dem Modell wird dazu zwischen Schienenpersonenfernverkehr (SPFV), Schienenpersonennahverkehr (SPNV) und Schienengüterverkehr (SGV) unterschieden. Abhängig von den auf einer Strecke betriebenen Verkehrstyp können weitere Constraints verknüpft werden. So sollen in dieser Arbeit alle Strecken, auf denen Fernverkehr betrieben wird, mindestens zweigleisig sein. Dies ist damit begründet, weil der SPFV aufgrund einzuhaltender Anschlussmöglichkeiten in der Regel eine höhere Priorität besitzt als SPNV oder SGV und deshalb auf besser ausgestatteten Strecken fährt. Ein weiterer Grund sind die relativ langen Wartezeiten, die aufgrund von Folgeverspätungen und großen Entfernungen im SPFV auftreten können, die ein Fernzug bei eingleisiger Strecke warten müsste, wenn er an einem vorgegebenen Ausweichgleis mit dem verspäteten Fernzug kreuzen muss.

Weiterhin sollen Strecken, die mit nur einem Verkehrstyp betrieben werden, aufgrund der anzunehmenden geschwindigkeitshomogenen[4] Betriebsführung und der daraus resultierenden hohen Leistungsfähigkeit über maximal zwei Gleise verfügen. Im Allgemeinen genügt es jeweils ein Gleis für die Hin- und Rückrichtung bereitzustellen, um den auftretenden Verkehr solcher Strecken zu bewältigen.

Ob eine Strecke mit SPFV, SPNV oder SGV betrieben wird, wird durch die Angabe der in Kapitel 3.3.2 erläuterten Streckenzüge formuliert. Für die verbleibenden Strecken wird angenommen, dass sie für Schienenpersonennahverkehr ausgelegt sind. Die Annahme beruht auf der Tatsache, dass SPNV in real existierenden Eisenbahnnetzen quantitativ überwiegt. Durch die Festlegung ob SPFV, SPNV oder SGV auf den Strecken durchgeführt wird, werden die Knoten, die mit diesen Strecken verbunden sind, implizit durch diese Verkehrstypen klassifiziert. Die Form der Klassifizierung von Knoten wurde bereits auf Seite 46 erwähnt. Da in der Praxis in einem Teil der Nahverkehrsknoten auch Fernverkehrshalte durchgeführt werden, genaue Kriterien dazu ohne weitere komplexe Modellierung aber nicht berechenbar sind, sollen Knoten mit Fernverkehrshalt hier ausschließlich die Knoten sein, die zulaufende Strecken mit SPFV verbinden und gewisse Constraints erfüllen. Unter diesen Knoten wird ein Fernverkehrshalt nur bei denjenigen durchgeführt, die mindestens zwei Streckenzüge mit SPFV verknüpfen, eine Kardinalität von mindestens vier Strecken besitzen oder zu einem spezifizierten Anteil dieser Knoten gehören.

4 Geschwindigkeitshomogener Verkehr ist ein markantes Merkmal für leistungsfähige Strecken. In diesen Strecken behindern langsamere Züge die schnellen Züge nicht bzw. die langsameren Züge müssen nicht ausweichen um überholt zu werden.

Zusätzlich soll durch die Angabe, ob Mischungen bzw. Entmischungen der Verkehrstypen auf den Strecken bestehen die Möglichkeit gegeben sein, in der Realität vorkommende Netze zu charakterisieren. Dabei wird zwischen zwei Formen der Mischung von Verkehrstypen unterschieden. Zum einen kann eine Mischung unterschiedlich schneller Verkehre existieren und zum anderen ein Mischung von Personen- und Güterverkehr. Beispielsweise sind in Frankreich die Strecken mit Güterverkehr von den Strecken mit Personenverkehr getrennt. Um die Leistungsfähigkeit des Gesamtnetzes zu steigern, wird in Deutschland durch das Projekt *„Netz 21“* versucht, eine geschwindigkeitsorientierte Entmischung der Verkehre vorzunehmen.

Unterwegsbahnhöfe einer Strecke

Die Darstellung des Eisenbahnnetzes auf Grundlage eines Graphen betrachtete bisher nur die Knotenbahnhöfe. Um die restlichen Bahnhöfe auf den Strecken zu generieren, muss angegeben werden wie viele Unterwegsbahnhöfe erzeugt werden sollen und wo sich diese auf den Strecken befinden. Die gewünschte Anzahl aller Unterwegsbahnhöfe wird durch den Anwender spezifiziert. Die Positionen der Unterwegsbahnhöfe wird dagegen nicht spezifisch angegeben. Sie werden durch Constraints auf den Stecken verteilt. Dies hat den Vorteil, dass dieses Problem durch weitere Constraints spezieller modellierbar ist. Eine Bedingung, die gefordert werden muss, ist der Mindestabstand von Unterwegsbahnhöfen zu anderen Bahnhöfen. Im Gegensatz zu den euklidischen Abständen der Knotenbahnhöfe untereinander, wird der Abstand der Unterwegsbahnhöfe über die Kilometrierung entlang der Strecke angegeben. Die Anzahl möglicher zu positionierender Unterwegsbahnhöfe wird daher durch den geforderten Mindestabstand und der tatsächlichen Länge einer Strecke eingeschränkt. Bei einem geforderten Mindestabstand $min_d \in \mathbb{R}$; $min_d \geq 0$ und einer Streckenlänge $l \in \mathbb{R}$; $l \geq 0$, können $\lfloor (l / min_d) - 1 \rfloor$ Unterwegsbahnhöfe positioniert werden. Beispielsweise können auf einer 12,0 LE langen Strecke und einem Mindestabstand von 3,5 LE maximal $\lfloor (12{,}0 / 3{,}5) - 1 \rfloor = 2$ Unterwegsbahnhöfe liegen. Abbildung 3.11 veranschaulicht diesen Fall.

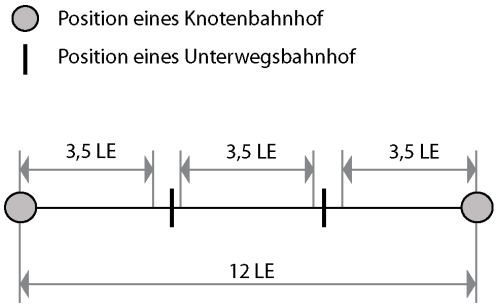

Abbildung 3.11: Ein Beispiel für die Positionierung zweier Unterwegsbahnhöfe

Zuordnung der Verkehrstypen zu den Streckengleisen

Verkehre werden für gewöhnlich auf einem Gleis im Zweirichtungsbetrieb oder auf zwei Gleisen im Ein- bzw. Zweirichtungsbetrieb durchgeführt. Zwei Gleise im Einrichtungsbetrieb sind im allgemeinen leistungsfähiger als ein Gleis, welches in beiden Richtungen befahren wird. Zugfolgezeiten unter zehn Minuten sind nach [Fie05] auf eingleisigen Strecken kaum realisierbar. Dagegen können Zugfolgezeiten auf zweigleisigen Strecken weit unter fünf Minuten erreicht werden. Aus diesen Gründen wird eine Strecke in *Einzelgleise* und *Gleispaare* unterteilt. Eine Strecke wird nur dann ein Einzelgleis zugeordnet, wenn aufgrund der Gleisanzahl kein weiteres Gleispaar gebildet werden kann. Verkehrsströme werden also nur einzelnen Gleisen in drei- oder eingleisigen Strecken zugeordnet. Zwei- und viergleisigen Strecken sind in ein bzw. zwei Gleispaare unterteilt. Demzufolge ist die Entmischung von unterschiedlichen Verkehrstypen auf einer Strecke nur mit drei oder vier Gleisen möglich. Aus diesem Grund werden geforderte Entmischungen von Verkehren durch Constraints überwacht. Beispielsweise sind Strecken mit Güterverkehr und Personenverkehr bei geforderter Entmischung nur drei- oder viergleisig. Wenn SPFV, SPNV und SGV auf einer Strecke auftreten, dann ist entweder die Entmischung bzgl. der Geschwindigkeiten oder die Entmischung von Güter- und Personenverkehr möglich. Zwei von den drei Verkehrstypen werden in diesem Fall auf ein Einzelgleis bzw. Gleispaar zusammen gelegt. SPFV ist i.d.R. schneller als SPNV und SGV. Aus diesem Grund würde eine Zusammenlegung mit SPFV die geforderte Entmischung von Geschwindigkeiten nicht einhalten. Andererseits würde eine Zusammenlegung des SPNV mit dem SGV die geforderte Entmischung von Güter- und Personenverkehr nicht erfüllen.

Wie bereits angesprochen, werden Gleispaare sowohl im Ein- als auch im Zweirichtungsbetrieb befahren. Da es keine eindeutige Festlegung gibt, wann welche Betriebsweise angeordnet wird, soll für jedes Einzelgleis bzw. Gleispaar die jeweilige Betriebsweise notiert werden können. Dabei wird im Rahmen dieser Arbeit der Zweirichtungsbetrieb für Einzelgleise mit Hilfe von Constraints

erzwungen. Weiterhin soll es dem Anwender möglich sein, auf bestimmten Streckentypen *Gleiswechselbetrieb*[5] (GWB) zu erzwingen (Abbildung 3.12). Dadurch ist ein mächtigere Beschreibung zur Spezifizierung von Strecken möglich. Beispielsweise können somit Neubaustrecken charakterisiert werden. Neubaustrecken werden in Deutschland immer für GWB ausgerüstet.

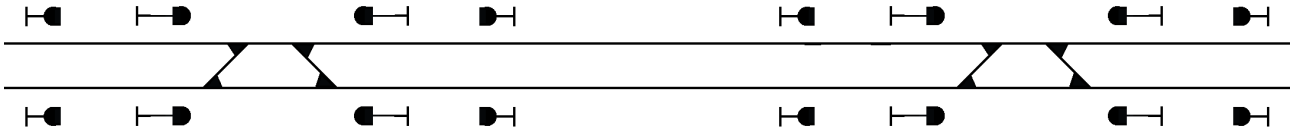

Abbildung 3.12: Eine für GWB im Zweirichtungsbetrieb befahrbare und mit Überleitstellen ausgestattete Strecke

3.5 Generierung der Bahnhöfe

Knotenbahnhöfe sind nach [Pa99] Bestandteile des Eisenbahnnetzes, die mehrere Strecken miteinander verbinden. Sie bestehen im wesentlichen aus zwei Elementen.

- den Bahnhofsköpfen, in denen die in den Bahnhof einmündenden Strecken physisch miteinander verbunden sind. Sie werden in Form von Fahrstraßenknoten dargestellt.
- der Gleisgruppe, die zwischen den Bahnhofsköpfen liegt. Sie verbindet die über den Bahnhof verlaufenden Linien verkehrstechnisch und betrieblich miteinander.

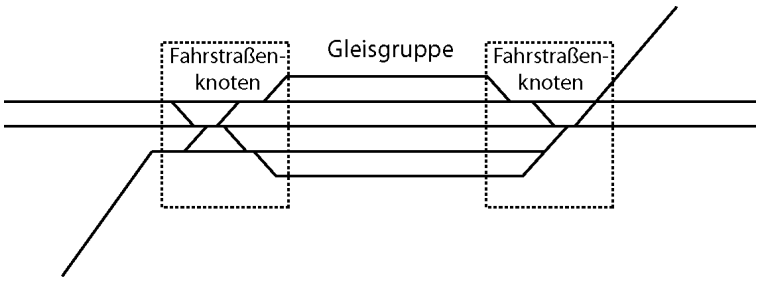

Abbildung 3.13: Struktur eines kleinen Knotens, Quelle: [Pa99]

Unterwegsbahnhöfe bestehen ebenso aus zwei Bahnhofsköpfen und einer Gleisgruppe, wobei deren Bahnhofsköpfe für gewöhnlich eine einfachere Struktur aufweisen als die der Knotenbahnhöfe. Aus der Struktur dieser Bahnhöfe ergeben sich für deren Leistungsanforderungen zwei wesentliche Aufgaben. Zum einen muss der Umfang der Gleisgruppe bemessen und zum anderen müssen die

5 Gleiswechselbetrieb ist die Betriebsweise, bei der ein Gleis einer zweigleisigen Strecke signaltechnisch entgegen der gewöhnlichen Fahrtrichtung, also im Zweirichtungsbetrieb befahren werden kann. Außerdem muss eine ausreichende Anzahl an Überleitstellen für den Gleiswechsel vorhanden sein.

Fahrstraßenknoten gemäß den Anforderungen gestaltet werden. In der Eisenbahnbetriebswissenschaft hat sich eine getrennte Betrachtung dieser beiden Aufgaben durchgesetzt.

3.5.1 Bemessung der Gleisgruppe

Aufgrund einer geforderten Betriebsqualität und Leistungsfähigkeit eines Knotenbahnhofs sollten Durchfahrten nicht durch haltende Züge anderer einmündender Strecken in gleicher Fahrtrichtung blockiert werden. Aus diesem Grund wird für die Infrastrukturplanung empfohlen, dass im gleichen Bahnhofskopf einmündende Strecken erst im Ausfahrbereich vereinigt werden. (vgl. [Pa99]) In dieser Arbeit wird für die Bemessung der Gleisgruppe deswegen eine Mindestanzahl an Gleisen in der Gleisgruppe gefordert. Zu modellieren sei ein Bahnhof mit $n_{Gleise,1}$ einmündende Streckengleise am ersten Bahnhofskopf und $n_{Gleise,2}$ Streckengleise am zweiten Bahnhofskopf. Demzufolge sollen mindestens $max(n_{Gleise,1}, n_{Gleise,2})$ Gleise in der Gleisgruppe vorhanden sein. Somit ist es möglich, dass jedes einmündende Gleis eines Bahnhofskopfes einem Gleis in der Gleisgruppe zugewiesen werden kann. Weiterhin soll gelten, dass jedem einmündenden Streckengleis genau ein Gleis in der Gleisgruppe zugewiesen werden darf. Somit ist jedes Gleis in der Gleisgruppe mit höchstens zwei Streckengleise von jeweils einem Bahnhofskopf verbunden.

Definition: Im folgenden Text wird ein Gleis der Gleisgruppe, dem mindestens ein Streckengleis zugewiesen ist und es quasi weiterführt, als *weiterführendes Hauptgleis* bezeichnet.

Es existieren somit maximal $n_{Gleise,1}+n_{Gleise,2}$ weiterführende Hauptgleise pro Bahnhof. Nach dieser Definition ist ein durchgehendes Hauptgleis (siehe Abschnitt 2.2, S. 20) stets ein weiterführendes Hauptgleis. Die Leistungsfähigkeit eines Bahnhofs wird durch die Anzahl seiner weiterführenden Hauptgleise charakterisiert. Daher soll diese Anzahl in dieser Arbeit durch Constraints beschrieben werden können.

Anforderungen von Integralen Taktfahrplänen an die Infrastruktur von Knoten

Der *Integrale Taktfahrplan* (ITF) besteht aus einem Netz von Zuglinien, die in Knoten miteinander verknüpft werden. Die Linien sind so auf einen Takt abgestimmt, dass zwischen diesen Zuglinien zu einer bestimmten Systemzeit gleichzeitig untereinander umgestiegen werden kann. Dafür kommen

die Züge der Linien kurz vor der ITF-Systemzeit in den Knoten an, halten dort solange über die Systemzeit hinaus bis genügend Zeit vergangen ist, um zwischen diesen Linien umzusteigen. Anschließend verlassen sie den Knoten wieder. Aus der Zahl miteinander zu verknüpfender Zuglinien ergibt sich unmittelbar die geforderte Gleisanzahl eines ITF-Knotens. Ein Problem bei der Auslegung der Knoten für den Integralen Taktfahrplan, sind die Einfahrten der einzelnen Züge kurz vor der ITF-Systemzeit. Da sie in einem relative kleinen Zeitfenster eintreffen, sollten sich die Fahrstraßen der Einfahrten nicht gegenseitig ausschließen. Zudem können durch sich ausschließende Ausfahrstraßen anfallende Haltezeiten während des Verkehrshaltes im Knoten als zusätzliche Umstiegszeiten genutzt werden. Die Sperrzeit einer Einfahrstraße ist im ITF aufgrund der Annäherungsfahrzeit[6] zwischen Einfahrvorsignal und Einfahrsignal wesentlich größer als die Sperrzeit einer Ausfahrstraße. Denn durch das Halten der Züge im Knoten, entfällt die Annäherungsstrecke der Ausfahrstraße. Aus diesen Gründen werden Kreuzungen und Einmündungen von Fahrstraßen bei ITF-Knoten in den Ausfahrbereich verlegt. Um ITF-Knotenbahnhöfe generieren zu können, die diese Anforderungen erfüllen, soll für jedes Streckengleis spezifiziert werden können, ob eine Linie dieser Strecke eine ungehinderte Einfahrt in die Bahnhöfe verlangt. Aus der Angabe von Strecken, dessen Linien eine ungehinderte Einfahrt verlangen, wird durch Constraints die benötigte Anzahl weiterführender Hauptgleise eines Bahnhofs abgeleitet. Demnach müssen mindestens so viele weiterführende Hauptgleise existieren, wie ungehinderte Einfahrten ermöglicht werden sollen. Zudem darf keinem weiterführenden Hauptgleis zwei Strecken mit Linien ungehinderter Einfahrt zugewiesen werden.

Beispiel 3.2: In einem Knotenbahnhof münden fünf Strecken A, B, C, D und E ein (Abbildung 3.14). Strecke A und E sind zweigleisig und bestehen jeweils aus einem Gleispaar. Diese beiden Strecken werden im Einrichtungsbetrieb befahren. Die Strecken B, C und D sind eingleisig und werden im Zweirichtungsbetrieb befahren. Dem Verkehr auf den Strecken A, C, D und E sollen gleichzeitige Einfahrten in den Knoten ermöglicht werden. Diese werden in der Abbildung durch gestrichelte Pfeile dargestellt. Aufgrund der vier gleichzeitig geforderten Einfahrten, muss der Knoten mindestens vier weiterführende Hauptgleise besitzen. Die Einfahrt von Strecke B kann nur dann erfolgen, wenn die Einfahrstraße nicht von den anderen Einfahrstraßen ausgeschlossen wird. Entweder kann von der Strecke B in ein freies weiterführendes Hauptgleis oder wenn die

6 Die Annäherungsfahrzeit ist ein Teil der Gesamtsperrzeit für Fahrstraßen, die ohne vorherigen Halt befahren werden sollen. Sie wird durch den Vorsignalabstand und der Fahrtgeschwindigkeit maßgeblich bestimmt. Aufgrund der Signalabhängigkeit von Fahrstraßen, muss die zu befahrene Fahrstraße bereits festgelegt sein, bevor das Hauptsignal und das zugehörige Vorsignal auf Fahrt gestellt werden können.

erreichbaren weiterführenden Hauptgleise belegt sind in das Überholungsgleis eingefahren werden.

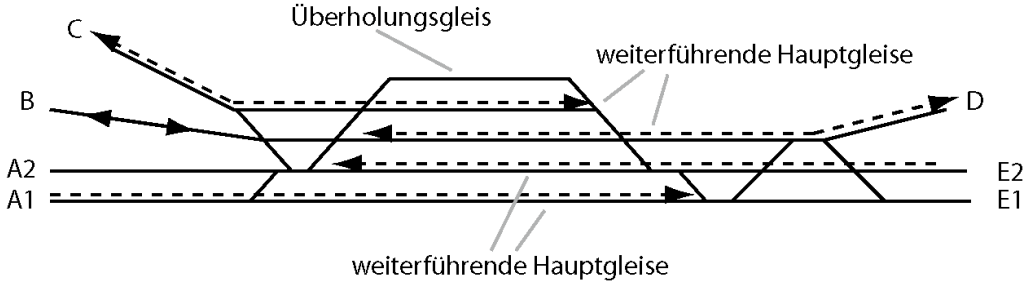

Abbildung 3.14: Ein Knoten, der vier Einfahrten gleichzeitig erlaubt

Zusätzlich müssen die Gleise der Gleisgruppe bestimmte Mindestlängen einhalten, um die im Bahnhof haltenden Züge aufnehmen zu können. Die Gleislänge wird besonders von den Einfahrgeschwindigkeiten und der Art der Züge bestimmt. Die Gleise müssen entsprechend lang sein, um bis zu 700 m lange Güterzüge oder bis zu 400 m lange Fernverkehrszüge behandeln zu können. Zu dem wird noch der Durchrutschweg mit eingeplant, dessen Länge abhängig von den zulässigen Einfahrgeschwindigkeiten von unter 50 m bis 200 m variieren kann. Überdies wird noch eine Streckschutzlänge von 10 m hinzu addiert (siehe Seite 34). Diese Werte orientieren sich an den in der Praxis eingesetzten Maßen und sollen bei der Generierung durch Constraints überwacht werden.

Überholungsgleise

Zu den weiterführenden Hauptgleisen sollen zusätzlich Überholungsgleise durch Constraints spezifizierbar sein. Somit besteht die Möglichkeit in weiteren Arbeiten Überholungsgleise aufgrund spezieller Modelle näher beschreiben zu können. In dieser Arbeit beschränkt sich die Modellierung der Überholungsgleise auf die Angabe, wie viele Überholungsgleise wie oft im Eisenbahnnetz auftreten dürfen. Dabei wird sich der Angabe von relativen Häufigkeiten, der genauen Anzahl der Überholungsgleise, der Angabe auf welcher Seite des Bahnhofs und ob sich diese Gleise auf nur einer Seite oder zwei Seiten des Bahnhofs befinden sollen, bedient. Mit der Seite eines Bahnhofs sei hier gemeint, ob sich die Überholungsgleise in der zeichnerischen Darstellung eines Bahnhofs oberhalb oder unterhalb der weiterführenden Hauptgleise befinden. In der Abbildung 3.14 liegt das Überholungsgleis oberhalb der weiterführenden Hauptgleise. Um eine mächtigere Charakterisierung des gewünschten Eisenbahnnetzes zu ermöglichen, werden diese Angaben zusätzlich für Knoten- und Unterwegsbahnhöfe unterschieden. Dies ist sinnvoll, da in den Unterwegsbahnhöfen auf einer Strecke weniger Verkehr auftritt als in den mehrere Strecken aufnehmenden Knotenbahnhöfen. Beispielsweise können die Forderungen gestellt werden, dass

40% bis 50% der Unterwegsbahnhöfe genau ein Überholungsgleis an der unteren Seite besitzen und die Überholungsgleise bei 25% aller Unterwegsbahnhöfe nur auf einer Seite liegen.

Stumpfgleise

Die Modellierung von Stumpfgleisen wird hier nur ansatzweise eingeführt und soll das Grundgerüst für die Weiterentwicklung von Stumpfgleisen bilden. Stumpfgleise sind Bahnhofsgleise, die aufgrund einer verzweigenden Weiche nicht in ein anderes Bahnhofsgleis münden, sondern an einem Prellbock enden. Der Bahnhof in Abbildung 2.20 auf Seite 33 besitzt ein Überholungsgleis, welches rechts in einem Stumpfgleis endet. Oftmals werden in der Praxis Überholungsgleise mit Stumpfgleisen versehen, um den Durchrutschweg der Einfahrstraße auf das Stumpfgleis abzulenken. Die somit zusätzlich verbaute Schutzweiche ermöglicht durch ihren Flankenschutz weniger Fahrstraßenausschlüsse. Im Programm soll für jedes Überholungsgleis eines Bahnhofs angegeben werden können, an welchem Ende es in einem Stumpfgleis mündet oder nicht.

3.5.2 Gestaltung der Fahrstraßenknoten

Die Gestaltung der Fahrstraßenknoten eines Bahnhofs folgt in dieser Arbeit dem in [Pa99] vorgeschlagenen Grundsatz, von jedem Ausfahrgleis eine Ausfahrmöglichkeit auf jede Strecke zu bieten. Dies bedeutet, dass ein Bahnhofskopf dermaßen mit Gleisverbindungen ausgestattet sein muss, um von einem beliebigen Bahnhofsgleis eine Ausfahrstraße zu einem beliebigen Streckengleis, das in Ausfahrrichtung betrieben wird, bilden zu können. Somit besitzen die Bahnhöfe des generierten Eisenbahnnetzes die Eigenschaft, Züge den Wechsel von einer anliegenden Strecke zu jeder anderen anliegenden Strecke zu ermöglichen. Zur weiteren Verfeinerung könnten der Modellierung in späteren Arbeiten beispielsweise konkrete Betriebsprogramme unterlegt werden, auf dessen Grundlage verzichtbare Streckenverknüpfungen durch Constraints ausgeschlossen werden. Dies würde sich in dem Wegfall entsprechender Gleisverbindungen äußern und somit den wirtschaftlichen Aspekt bei der Generierung mehr berücksichtigen.

Aufbauend auf diesen Grundsätzen und den vorher beschriebenen Parametern wurde die Implementierung durchgeführt. Genauere Informationen werden dazu in Kapitel 4.6 ab Seite 80 gegeben.

4 Implementierung

Für die Realisierung der Algorithmen wird das Programmiersystem *CHIP* verwendet. *CHIP* enthält verschiedene Constraint-Löser (Solver). Diese unterscheiden sich im Typ der verwendeten Variablen. Die Solver können entweder rationale Zahlen, boolesche Werte oder natürliche Zahlen bearbeiten. Der Constraint-Löser, welcher natürliche Zahlen verarbeitet, wurde im praktischen Teil eingesetzt. In *CHIP* müssen diese Variablen zudem mit endlichen Domänen versehen werden. Aus diesem Grund werden die mit diesem Constraint-Löser bearbeiteten Constraints als *Finite-Domain-Constraints* bezeichnet. Da die Variablen mit endlichen Wertebereichen von den in der logischen Programmierung bekannten logischen Variablen unterschieden werden müssen, werden sie zur besseren Differenzierung als *Domänenvariablen* bezeichnet. Zusätzlich unterstützt CHIP das objektorientierte Paradigma, welches das Deklarieren von Klassen und die Erzeugung von Instanzen dieser Klassen unterstützt. In diesem Kapitel werden die wesentlichen Algorithmen dieser Arbeit erläutert und auf ihre Implementierungen näher eingegangen.

4.1 Einschränkungen im CHIP-System

In diesem Abschnitt werden Einschränkungen des CHIP-Systems diskutiert, die die Problemlösungen während der Implementierung beeinflusst haben bzw. die Anwendung der Software einschränken.

Im Finite-Domain-System werden *arithmetische Constraints* zur Verfügung gestellt. Sie gestatten es arithmetische Gleichungen und Ungleichungen abzusetzen. So sind Bedingungen wie $A\#=B+C$ oder $A\#>B+C$ formulierbar. A, B und C müssen Domänenvariablen oder natürliche Zahlen sein. Soll in einer Gleichung bzw. Ungleichung multipliziert werden, kann dies in *CHIP* aber nur eingeschränkt genutzt werden. So sind Ausdrücke wie $A\#=B*C$ und $A\#>B*C$ nur dann absetzbar, wenn mindestens einer der beiden Faktoren eine natürliche Zahl ist. Ist B eine Domänenvariable sind Bedingungen wie $A\#=B*3$ und $A\#>4*2$ möglich, $A\#=B*B$ hingegen nicht.

Eine weitere Einschränkung im CHIP-System ist die Belegung der Domänenvariablen mit Werten bis maximal 100000. Größere Werte werden von diesem Datentyp nicht unterstützt.

4.2 Gliederung der Teilprobleme

Gemäß der Aufteilung der Modellierung des Generierungsprozesses in mehrere Teilprobleme wird die Implementierung der Teilprobleme auch voneinander getrennt. Die Abarbeitung der Teilprobleme verläuft sequentiell. Dies ist in einigen Fällen notwendig, da bestimmte Teilprobleme die Ergebnisdaten vorhergehender Teilprobleme als Ausgangsdaten verwenden. Die Einzelschritte zur Bearbeitung eines Teilproblems können als ein zusammenhängender Abschnitt betrachtet werden. Ein Abschnitt zur Lösung eines Teilproblems gliedert sich in

- eine Initialisierungsphase
- eine Phase für das Absetzen von Constraints
- eine Phase zur Lösungssuche

In der Initialisierungsphase werden Methoden ausgeführt, um die Bearbeitung des Teilproblems durch Constraints vorzubereiten. In der Regel sind dies die Erzeugung der benötigten Datenstrukturen und ihre Initialisierungen. Es können hier auch komplexere Berechnungen durchgeführt werden, die basierend auf den vorher gelösten Teilproblemen beruhen. Werden für ein Teilproblem keine vorausgehenden Berechnungen benötigt, kann dieser Teilabschnitt entfallen. Die Initialisierungsphase eines Teilproblems wird im Programm mit *init(Problemname)* aufgerufen. Im zweiten Teilabschnitt werden die für das Teilproblem geforderten Constraints formuliert und abgesetzt. Für weitere Modellierungen eines Teilproblems können in dieser Phase Constraints hinzugefügt, abgeändert oder entfernt werden. Dieser Teilabschnitt wird in er Implementierung als *stateConstraints(Problemname)* bezeichnet. Der dritte Teilabschnitt enthält die Lösungssuche für das Teilproblem. Hier werden die implementierten Suchverfahren aufgerufen. Dieser Abschnitt wird in der Applikation mit *label(Problemname)* aufgerufen. Um diese Software weiter zu entwickeln, könnten weitere Teilprobleme mit dieser Struktur in den Generierungsprozess eingefügt werden.

4.3 Positionierung der Knoten

Die Positionen von Objekten werden in der praktischen Arbeit durch zweidimensionale Koordinaten beschrieben. Die erste Koordinatenkomponente beschreibt die Abszisse und wird im Folgendem mit x bezeichnet. Die zweite Komponente beschreibt die Ordinate und wird durch y dargestellt. Erzeugte Knoteninstanzen haben somit zwei Domänenvariablen als Attribute. Diese Attribute besitzen entsprechend der spezifizierten Feldgröße einen Wertebereich von Null bis zum

in ihrer jeweiligen Dimension größtmöglichem Wert im Feld. Das Feld wird durch zwei spezifizierte Werte in seiner horizontalen und vertikalen Ausdehnung bestimmt. Bei einer Feldgröße von $m*n$ Längeneinheiten beginnen die Koordinaten der Positionen bei (0;0) und enden bei (m-1; n-1).

4.3.1 Die Mindestabstände der Knoten

Als erster Ansatz für die Implementierung der Forderung zur Einhaltung der Mindestdistanzen zwischen den Knoten wurde die Berechnung auf Grundlage des euklidischen Abstandes zwischen den Knoten verwendet. Abbildung 4.1 veranschaulicht den euklidischen Abstand zwischen zwei Knoten im zweidimensionalen Raum.

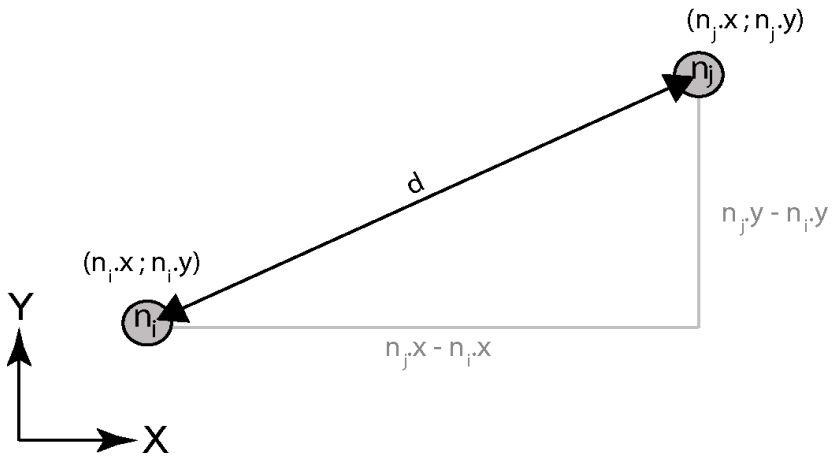

Abbildung 4.1: Der euklidische Abstand d zweier Knoten.

Bei zwei gegebenen Knoten n_i und n_j mit den Koordinaten $(n_j.x ; n_i.y)$ bzw. $(n_j.x ; n_j.y)$ würde sich ihr euklidischer Abstand nach der folgenden Formel berechnen lassen.

$$d = \sqrt{((n_j.x - n_i.x)^2 + (n_j.y - n_i.y)^2)} \quad (1)$$

Die Positionen der beiden Knoten werden durch Domänenvariablen repräsentiert. Bei einem geforderten Mindestabstand d_{min} muss demnach $d \geq d_{min}$ gelten. Durch Umstellung der Formel lässt sich die Wurzel eliminieren. Dies führt zu der Gleichung, welche dem Satz des Pythagoras gleicht. Die Gleichung lautet:

$$d^2 = (n_j.x - n_i.x)^2 + (n_j.y - n_i.y)^2 \quad (2)$$

Aus $d \geq d_{min}$ folgt $d^2 \geq d_{min}^2$. Letztendlich führt dies zu der folgenden Bedingung:

$$(n_j.x - n_i.x)^2 + (n_j.y - n_i.y)^2 \geq d_{min}^2 \quad (3)$$

Sind $a = n_j.x - n_i.x$ und $b = n_j.y - n_i.y$ zwei Domänenvariablen, dann würde die Gleichung 3 als

$a*a+b*b \geq d_{min}^2$ formuliert werden. In Abschnitt 4.1 wurde bereits beschrieben, dass sich Multiplikationen mit mehr als einer Domänenvariable, hier sind es $a*a$ und $b*b$, nicht durch Constraints ausdrücken lassen. Aus diesem Grund ist es nicht möglich, den euklidischen Abstand zwischen zwei Knoten in einem Constraint zu verwenden.

Ein alternativer Ansatz besteht in der Forderung, dass zwei Knoten als weit genug entfernt gelten, wenn der horizontale oder vertikale Abstand zwischen diesen beiden Knoten größer gleich dem spezifizierten Mindestabstand ist. Mit dieser Bedingung ist sichergestellt, dass der Abstand der beiden Knoten bereits in einer Dimension eingehalten wird. Wird für einen Knoten diese Bedingung eingehalten, kann sich der Bereich in dem kein anderer Knoten vorkommt als ein Quadrat vorgestellt werden. Dieses Quadrat besitzt bei einem gegebenen Mindestabstand d_{min} eine Seitenlänge von $2*d_{min}+1$ Längeneinheiten (Abbildung 4.2). Besitzen zwei Knoten einen horizontalen bzw. vertikalen Abstand von weniger als $2*d_{min}$ Längeneinheiten, würden sich ihre Quadrate überlappen.

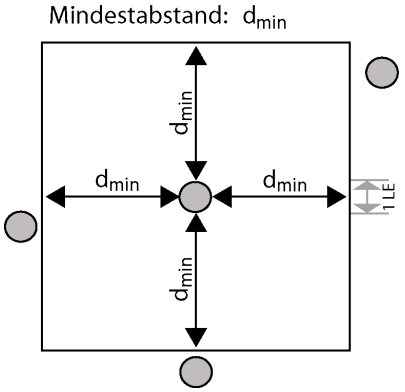

Abbildung 4.2: geforderter horizontaler oder vertikaler Mindestabstand zwischen Knoten.

Die daraus folgende Idee für die Implementierung dieser Bedingung ist, dass es genügt für jeden Knoten n_j zu prüfen, ob diejenigen Knoten, die vertikal oder horizontal weiter entfernt vom Ursprung liegen, den Mindestabstand in der jeweiligen Dimension zum Knoten n_j einhalten. Die verbleibenden Knoten, welche in beiden Dimensionen näher am Ursprung liegen, müssen nicht geprüft werden. Sei n_i ein Knoten der vertikal bzw. horizontal näher am Ursprung liegt als n_j. Dann wird für n_i bereits geprüft, ob der Knoten n_j in der jeweiligen Dimension den Mindestabstand einhält. Veranschaulicht bedeutet dies, dass für einen Knoten n_i mit den Koordinaten $(n_i.x\,;n_i.y)$ nur noch der Bereich $n_i.x$ bis $n_i.x+d_{max}$ für die horizontale Dimension und $n_i.y$ bis $n_i.y+d_{max}$ für die vertikale Dimension auf vorhandene Knoten geprüft werden muss. Abbildung 4.3

veranschaulicht diesen Sachverhalt. Da der Ursprung in der Abbildung links unten ist, erstreckt sich ein für einen Knoten freizuhaltender Bereich, entgegen dem Ursprung, nach rechts oben. Es ist zu erkennen, dass für jeden Knoten ein Bereich geprüft werden muss, der einem Quadrat mit der Seitenlänge $d_{max}+1$ LE entspricht.

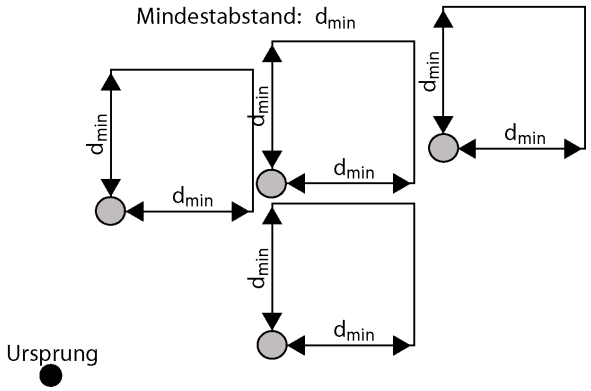

Abbildung 4.3: Verfahren zur Prüfung des Mindestabstands.

Die Knoten halten genau dann den Mindestabstand ein, wenn sich ihre quadratischen Bereiche nicht überlappen. Dies entspricht der Spezifikation des im Kapitel 2.1.6.1 beschriebenen *diffn*-Constraints im zweidimensionalen Raum. Um für die zehn Knoten $\{n_1,\ldots,n_{10}\}$ aus dem Beispiel auf Seite 40 einen Abstand von mindestens einer Längeneinheit zu sichern, wird das Constraint

$$\text{diffn}([n_1.x, n_1.y, 2, 2], [n_2.x, n_2.y, 2, 2], [n_3.x, n_3.y, 2, 2], [n_4.x, n_4.y, 2, 2], [n_5.x, n_5.y, 2, 2],$$
$$[n_6.x, n_6.y, 2, 2], [n_7.x, n_7.y, 2, 2], [n_8.x, n_8.y, 2, 2], [n_9.x, n_9.y, 2, 2], [n_{10}.x, n_{10}.y, 2, 2])$$

im Programm abgesetzt. In Abbildung 4.4 wird gezeigt, wie der Abstand durch die geforderte Überlappungsfreiheit der zehn Quadrate für dieses Beispiel eingehalten wird. Die Quadrate besitzen eine Kantenlänge von zwei Längeneinheiten. Die Forderung von getrennten Mindestabständen pro Dimension ist möglich, aber in der Praxis nicht sinnvoll.

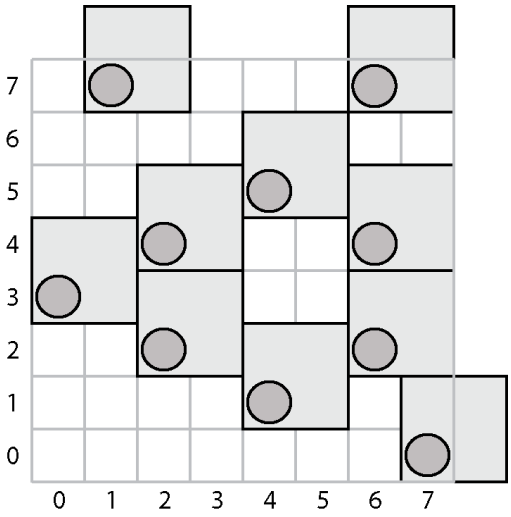

Abbildung 4.4: Realisierung von Mindestabständen mit 1LE durch das diffn-Constraint

4.3.2 Knoten in Randbereiche

Um die Forderung nach einer gewissen Anzahl von Knoten in einer der in Kapitel 3.1.2 beschriebenen Randzonen zu realisieren, existiert in der Spezifikation für jede der vier Randzonen ein Prädikat. Diese Prädikate heißen gemäß ihren Zonen *northzoneNodeProp*, *eastzoneNodeProp*, *southzoneNodeProp* und *westzoneNodeProp*. Die Bedingungen sind optional und können durch die Prädikate jeweils ein- und ausgeschaltet werden. Wie in Kapitel 3.1.2 angesprochen, sind ihre Parameter als relative Werte anzugeben. Das bedeutet, dass ihre Werte in einem Bereich von Null bis Eins liegen müssen. Dies gilt für die geforderte Mindest- und Höchstanzahl der Knoten in der Randzone, sowie für dessen Grenzlinie. Im Folgenden wird die Berechnung für die westliche und die östliche Randzone erläutert.

Sei $feld_{X,max}$ die Ausdehnung des Feldes auf der horizontalen Achse und k die Anzahl aller Knoten n_i. In der westlichen Randzone soll ein Anteil von mindestens min_{west} und höchstens max_{west} aller Knoten liegen. Die Grenzlinie der westlichen Zone wird durch $grenze_{west}$ bestimmt. Die absolute Mindest- und Höchstanzahl der Knoten, die in dem westlichen Bereich liegen müssen, wird demnach durch $Min_{west} = \lfloor k * min_{west} \rfloor$ und $Max_{west} = \lfloor k * max_{west} \rfloor$ berechnet. Der größte dem westlichen Randbereich zugehörige Wert auf der horizontalen Achse wird durch $spalte_{west} = (feld_{X,max} - 1) * grenze_{west}$ bestimmt. Zusätzlich sei der östliche Randbereich mit min_{ost}, max_{ost} und $grenze_{ost}$ spezifiziert. Da die beiden Zonen sich nicht überschneiden sollen, gilt zusätzlich $grenze_{ost} > grenze_{west}$. Die Werte Min_{ost}, Max_{ost} und $spalte_{ost}$ werden analog berechnet. Mit dem *among*-Constraint (siehe Kapitel 2.1.6.2) können die Bedingungen für die westliche und

östliche Randzone gleichzeitig abgesetzt werden. Der Aufruf dieses Constraints ist:

$$among([N_{west}, N_{ost}], [n_1.x, \ldots, n_k.x], [0,\ldots,0], [[0,\ldots, spalte_{west}], [spalte_{ost}, \ldots, feld_{X,max}-1]], all)$$
$$mit\ N_{west} \in [Min_{west}, Max_{west}]\ \ und\ \ N_{ost} \in [Min_{ost}, Max_{ost}]$$

Die beiden Randzonen können deshalb in einem *among*-Constraint beschrieben werden, weil sie sich auf denselben Domänenvariablen, den *x*-Attributen der Knoteninstanzen, beziehen. Die Formulierung der südlichen und nördlichen Randbereiche erfolgt analog über ein weiteres *among*-Constraint, welches sich aber auf die *y*-Attribute der Knoteninstanzen bezieht.

4.3.3 Knotenballungen

Um die in Kapitel 3.1.3 besprochene Modellierung von Ballungsräume zu realisieren, wurde eine neue Klasse deklariert. Sie enthält neben Attributen zur Speicherung der Position und Ausdehnung eines Ballungsraumes auch eine Liste auf die dem Ballungsraum zugewiesenen Knoten. Die notwendigen Knoten werden den Ballungsräumen im Initialisierungsteil der Positionierungsphase zugewiesen. Dem Anwender ist es freigestellt, beliebig viele Ballungsräume zu spezifizieren. Eine Ballung von Knoten wird durch die Angabe der horizontalen und vertikalen Ausdehnung der Ballung sowie der gewünschten Knotenanzahl definiert. Das Ballungsgebiet wird als eine rechteckige Fläche interpretiert. Die Positionen der Ballungsrauminstanzen wird durch Domänenvariablen realisiert und können daher bei weiterer Modellierung mit zusätzlichen Constraints beschrieben werden.

Um die in dieser Modellierung geforderte Überlappungsfreiheit von Ballungsräumen zu realisieren, bot sich aufgrund der rechteckigen Struktur der Ballungsräume die Implementierung durch das *diffn*-Constraint (siehe Kapitel 2.1.6.1) an. Abhängig von der Zugehörigkeit der Knoten zu den Ballungsräumen werden die möglichen Positionen der Knoten durch Constraints eingeschränkt. Gehört ein Knoten zu einem Ballungsraum, werden seine möglichen Positionen auf diesen Fläche eingeschränkt. Gehört ein Knoten zu keinem Ballungsraum, wird er durch Constraints von den Ballungsgebieten ausgeschlossen. Im Folgenden werden diese Constraints beschrieben.

Seien $b_1, \ldots, b_n$ n spezifizierte Ballungsrauminstanzen. Für eine Instanz $b_i\ mit\ 1 \le i \le n$ sei $(b_i.x\,;b_i.y)$ dessen Position, $b_i.width$ dessen horizontale Ausdehnung und $b_i.height$ dessen vertikale Ausdehnung. Sei eine Knoteninstanz n_j einem Ballungsgebiet b_i zugeordnet, dann werden auf den Knoten n_j die vier arithmetischen Constraints $n_j.x\# \ge b_i.x$, $n_j.x\# \le b_i.x + b_i.width$, $n_j.y\# \ge b_i.y$ und $n_j.y\# \le b_i.y + b_i.height$ angewendet. Gehört ein Knoten n_k zu keinem

Ballungsraum, so wird für jeden Ballungsraum b_i *mit* $1 \leq i \leq n$ ein Constraint $(n_k.x\#<b_i.x) \vee (n_k.x\#>b_i.x+b_i.width) \vee (n_k.y\#<b_i.y) \vee (n_k.y\#>b_i.y+b_i.height)$ abgesetzt. Abbildung 4.5 veranschaulicht diese Constraints für einen Ballungsraum b_i und zwei Knoten n_j und n_k, wobei n_j im Ballungsraum und n_k rechts neben dem Ballungsraum liegt. Somit erfüllt n_k das bedingte Constraint $n_k.x\#>b_i.x+b_i.width$ während der Knoten n_j die vier auf ihm abgesetzten arithmetischen Constraints erfüllt.

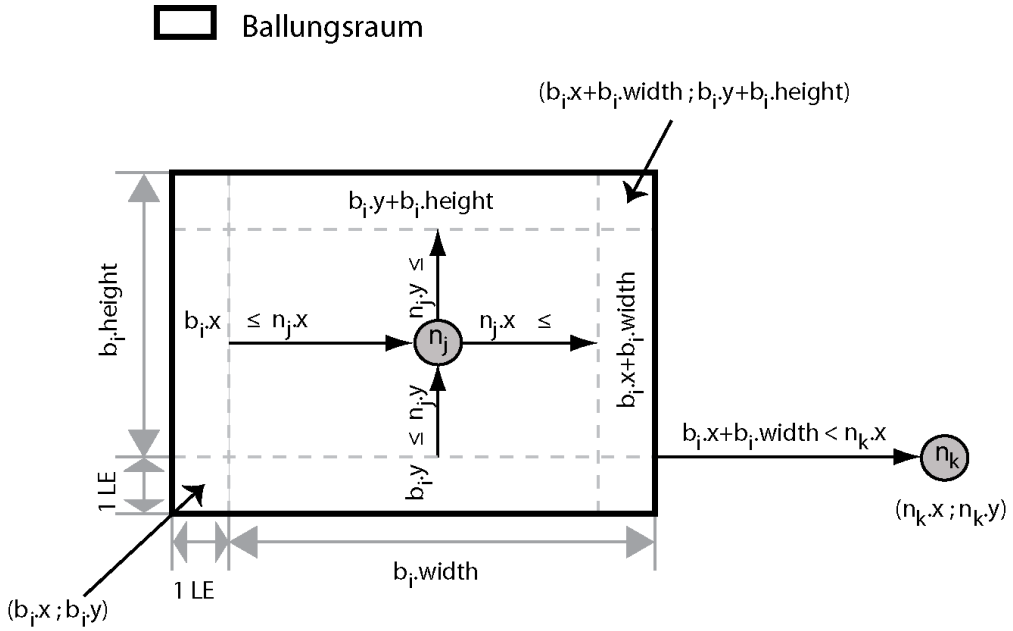

Abbildung 4.5: Geometrische Darstellung eines Ballungsraums

4.4 Knotenverbindungen

Diese Phase des Generierungsprozesses schließt nach der Positionierung der Knoten an und verbindet die Knoten des Graphen mit Kanten. Die Kanten des Graphen werden in der Implementierung durch eine eigene Klasse beschrieben. Dabei werden die Kantenobjekte erst nach der Positionierung der Knoten erzeugt. Dies hat den Vorteil, dass abhängig von den Knotenpositionen die genaue Anzahl an benötigten Kanten ermittelt werden kann. Aufgrund des in Kapitel 3.2.1 beschriebenen Kriteriums für die Zulassung einer Knotenverbindung wird nicht für jedes Paar von Knoten eine Kante benötigt. Das Kriterium sorgt in der Regel für einen Bedarf an Kanten zwischen zwei Knoten, die sich unmittelbar nebeneinander befinden. Daher entspricht die Anzahl wirklich benötigter Kanten nur einen Bruchteil der für jedes Knotenpaar möglichen Kanten.

Weil nicht jede mögliche Kante im Graphen eine im generierten Netz existierenden Kante sein muss, besitzt die Kantenklasse ein Attribut *linked,* welches angibt, ob eine Kante zwei Knoten tatsächlich verbindet oder nicht. Werden zwei Knoten durch eine Kante verbunden, speichert das Attribut den Wert 1, sonst 0.

4.4.1 Ermittlung von benötigten Kanten zwischen zwei Knoten

Um festzustellen zwischen welchen Knoten eine mögliche Kante existieren soll, wird der folgende Algorithmus angewendet. Dieser Algorithmus bestimmt, welche Knoten mit einer Kante verbunden werden dürfen. Das bedeutet, dass für diese Fälle Kantenobjekte erzeugt werden. Der Algorithmus gibt noch keine Auskunft darüber, ob im Graphen tatsächlich eine Kante zwischen zwei Knoten existieren wird. Die tatsächlichen Verbindungen werden anschließend durch Constraints bestimmt. Im folgenden Algorithmus wird der euklidische Abstand zweier Knoten n_i und n_j durch $d(n_i, n_j)$ ausgedrückt. Die Anwendung dieses Verfahrens ist möglich, weil in in diesem Algorithmus keine Verarbeitung von Constraints stattfindet. Der in Kapitel 3.2.1 beschriebene Faktor, welcher bestimmt, ob der Umweg zu einem dritten Knoten eine Verbindung zwischen zwei Knoten verhindert, wird als hier mit f bezeichnet.

Algorithmus 1:

$K := \{n_1, \ldots, n_k\}$ *%die Menge der Knoten*
$B := \emptyset$
for all $n_a \in K$ do
  $B := B \cup \{n_a\}$
  for all $n_b \in (K \setminus B)$ do
    nicht_verbinden := false
    for all $n_c \in (K \setminus \{n_a, n_b\})$ do
      if ($d(n_a, n_b) > d(n_a, n_c) \wedge d(n_a, n_b) > d(n_c, n_b) \wedge f * d(n_a, n_b) > d(n_a, n_c) + d(n_c, n_b)$)
      then
        nicht_verbinden := true
        verlasse diese Schleife
      fi
    od
    if (nicht_verbinden = false) then
      vermerke, dass die Knoten n_a und n_b benachbart sind

```
    fi
  od
od
```

Der Algorithmus enthält drei verschachtelte Schleifen. In der äußersten Schleife werden nacheinander Knoten $n_i mit 1 \leq i < k$ ausgewählt. In der zweiten Schleife werden Knoten $n_j mit\ j > i$ gewählt. Dies sind $(k^2 - k)/2$ mögliche (n_i, n_j) -Paare. Im schlimmsten Fall wird das Kriterium für jedes dieser Paare auf die verbleibenden $k-2$ Knoten angewendet. Werden die Bedingungen in der innersten Schleife durch einen dritten Knoten erfüllt, wird diese Schleife vorzeitig beendet. Daher müssen in diesem Algorithmus für k Knoten maximal

$$\frac{(k^2-k)}{2} * (k-2) \ = \ \frac{k^3 - 3k^2 + 2k}{2} \tag{4}$$

Bedingungen geprüft werden. Sollte für zwei Knoten n_i und n_j kein anderer Knoten die Bedingungen erfüllen, wird in den Knoteninstanzen notiert, dass eine Kante für dieses Paar erzeugt werden soll.

4.4.2 Knotenkardinalitäten

Um mit den Kardinalitäten der Knoten arbeiten zu können, wird jeder Knoten mit einem Attribut versehen, welches die Summe der mit ihm inzidenten Kanten speichert. Dieses Attribut wird *cardinality* bezeichnet und ist eine Domänenvariable. Daher wird für jeden Knoten ein arithmetisches Constraint über die Summe seiner möglichen Kanten abgesetzt. Seien $\{e_1, \ldots, e_m\}$ die Kanten, bei denen der Knoten n_i jeweils ein Endknoten ist, dann wird das Constraint $n_i.cardinality \ \#= \ e_1.linked + \ldots + e_m.linked$ aufgerufen. Die Summe kann auf diese Weise berechnet werden, weil für die tatsächlich verwendeten Kanten $linked = 1$ und sonst $linked = 0$ gilt.

Die geforderten Häufigkeiten der Grade von Knoten werden in der Spezifikation durch die Angabe von drei Parametern pro Grad festgelegt. Diese werden durch die Fakten *nodeCardProp* übergeben. Der erste Parameter von *nodeCardProp* beschreibt die Kardinalität, der zweite Parameter die minimale Häufigkeit von Knoten mit dieser Kardinalität und der dritte Parameter gibt die maximale Häufigkeit an. Die Parameter zur Beschreibung der Häufigkeiten geben diese relativ an und werden analog zu den vorher beschriebenen Spezifikationen durch rationale Werte von Null bis Eins

dargestellt. Auch hier kann das *among*-Constraint (siehe Abschnitt 2.1.6.2) für die Implementierung der Kardinalitäten genutzt werden. Wird für eine gewünschte Kardinalität c eine relative Mindesthäufigkeit min_c und eine Maximalhäufigkeit max_c angegeben, dann werden die absoluten Häufigkeiten für k Knoten durch $Min_c = \lfloor k * min_c \rfloor$ bzw. $Max_c = \lfloor k * max_c \rfloor$ berechnet. In der Implementierung werden mehrere spezifizierte Bedingungen über die Kardinalitäten $c \in \{1, \ldots, p\}$ durch das folgende *among*-Constraint ausgedrückt.

$$among([N_1, \ldots, N_p], [n_1.cardinality, \ldots, n_k.cardinality], [0, \ldots, 0], [[1], \ldots, [p]], all)$$
$$mit\ N_c \in [Min_c, Max_c]$$

4.4.3 Zusammenhang des Graphen

Bei den in der Lösungssuche auftretenden Zuweisungen, ob eine Kante zwei Knoten verbindet oder nicht, wird der Zusammenhang des Graphen nicht geprüft. Es können somit mehrere Zusammenhangskomponenten entstehen, die mehrere Netze ergeben. Dies ist in praktischen Anwendungen unwahrscheinlich, da meist Knoten mit Grad drei oder höher vorkommen. Es werden mindestens vier Knoten mit Grad drei benötigt um eine Zusammenhangskomponente zu bilden. Diese vier Knoten müssten untereinander verbunden sein. Tests haben in solchen Fällen fast immer zu einem zusammenhängenden Graphen geführt. Bei Knoten mit Kardinalität eins oder zwei treten hingegen häufiger kleinere Zusammenhangskomponenten auf. Der Grund ist, dass bereits zwei adjazente Knoten mit Grad eins sich vom Graphen isolieren. Das Gleiche gilt für drei untereinander verbundene Knoten mit Grad zwei. Aus diesem Grund wurde ein Constraint implementiert, welches verbietet zwei Knoten mit Grad eins zu verbinden. Seien n_i und n_j zwei Endknoten einer Kante, dann gilt $n_i.cardinality + n_j.cardinality \#> 2$. Eine Verhinderung von Zusammenhangskomponenten mit drei Knoten vom Grad zwei wurde nicht implementiert, da Knoten mit Grad zwei selten vorkommen. Ein Ansatzpunkt zur Erweiterung des Programms wäre die zusätzliche Formulierung solcher Constraints oder die Implementierung eines Algorithmus, der den gefundenen Graphen auf Zusammenhang prüft und ihn gegebenenfalls verwirft und die Suche nach einer anderen Lösung anstößt.

4.4.4 Netzgröße

Eine konkretere Formulierung der Topologie des generierten Schienennetzes ist durch weitere Constraints auch in dieser Phase möglich. So wurde ein Constraint zur Einschränkung der

Gesamtlänge der Kanten des Graphen implementiert. Die Gesamtlänge der Kanten wirkt sich wiederum auf die Gesamtlänge des Eisenbahnnetzes aus. Es werden dazu eine geforderte Mindest- und Höchstlänge in der Spezifikation angegeben. Sie werden hier als Min_L bzw. Max_L bezeichnet. Die Länge einer Kante $e_i \in \{e_1, \ldots, e_m\}$ wird durch $e_i.length$ bestimmt. Die Bedingung wird durch das Constraint $L\# = e_1.linked * e_1.length + \ldots + e_m.linked * e_m.length \quad mit \quad L \in [Min_L, Max_L]$ realisiert. Es werden nur die Kantenlängen der tatsächlich verwendeten Kanten $(linked = 1)$ in die Summe aufgenommen. Weil die Gesamtlänge durch das Constraint in einer Domänenvariable verarbeitet wird, ist aufgrund des in Abschnitt 4.1 beschriebenen restriktiven Maximalwertes einer Domänenvariable eine Einschränkung der Gesamtlänge bis zu 100000 LE zulässig. Vergleichsweise beträgt die Gesamtlänge des Netzes der DB AG ca. 38000 km. (vgl. [Fie05] S.14) Da eine zu strikte Eingrenzung der geforderten Gesamtlänge des Graphen unerfüllbar sein kann, sollte dieses Constraint sorgsam eingesetzt werden. Beispielsweise würde die Forderung einer sehr kurzen Gesamtlänge durch einen komplexen Graphen nicht erfüllt werden können.

4.5 Klassifizierung der Kanten des Graphen

Um die Strecken zu klassifizieren, mussten spezielle Parameter für die Klassen in der Spezifikation angegeben werden. Es stellte sich die Frage, nach welchen Kriterien Strecken zusammengefasst werden und welche Parameter verwendet werden. Diese Arbeit orientiert sich aufgrund der Praxisnähe an die bei der DB AG eingeführten Streckenkategorien [DB06a]. Die Streckenkategorien wurden von der DB Netz AG[7] erstellt, um anderen Eisenbahnunternehmen eine einfache und transparente Übersicht ihrer Infrastruktur, für deren Nutzung zu geben. Anhand dieser Kategorisierungen ist es möglich, gewisse Eigenschaften der Infrastrukturausstattungen abzuleiten. Es wird hier angemerkt, dass eine alternative Klassifizierung der Strecken nachträglich durch Änderungen in der Spezifikation möglich ist. Die DB Netz AG hat die Strecken ihres Netzes in 12 Kategorien unterteilt. Davon gehören sieben Kategorien zu den *Fernstrecken*, zwei zu den so genannten *Zulaufstrecken* und drei Kategorien beinhalten die Strecken des Stadtschnellverkehrs. Die Arbeit beschränkt sich auf die Strecken der klassischen Eisenbahn. Dies wären somit alle Zulauf- und Fernstrecken. Die Unterteilung ist für diese Arbeit nützlich, da die Zulaufstrecken den in der EBO spezifizierten Nebenbahnen und die Fernstrecken den Hauptbahnen entsprechen. Haupt- und Nebenbahnen unterscheiden sich aufgrund ihrer ökonomischen Bedeutsamkeit stark in ihren

7 Die DB Netz AG ist eine Tochtergesellschaft der Deutschen Bahn AG, die für die Schieneninfrastruktur verantwortlich ist.

Infrastrukturausstattungen. So sind Hauptbahnen vorrangig für schnelle Verkehre geeignet, während Nebenbahnen aufgrund teilweise stark reduzierter Sicherungstechnik und veralteter Infrastrukturelemente langsamere Verkehre bedienen. Diese Streckenkategorien werden nach den zulässigen Geschwindigkeiten der Strecken und den sich daraus ergebenen Infrastrukturausstattungen eingeteilt. So sind Zulaufstrecken bis maximal 100 km/h und Fernstrecken ab einer zulässigen Geschwindigkeit von 101 km/h zu befahren.

In der Implementierung wird ein Streckentyp durch eine natürliche Zahl größer Null repräsentiert. Die Verwendung von natürlichen Zahlen ermöglicht die Verarbeitung durch Finite-Domain-Constraints. Um die Streckenklassen zu beschreiben, muss in der Spezifikation die Gesamtzahl n aller möglichen Typen angegeben werden. Eine Streckenklasse entspricht einer Zahl $i \in \{1, \ldots, n\}$. Der Streckentyp einer Kanteninstanz wird in dem Attribut *linetype* abgelegt. Die bei der Modellierung besprochene Verteilung eines Streckentyps kann jeweils durch eines der n Prädikate *lineTypeProp* mit einer Mindest- und Höchstgrenze bestimmt werden. Die Verwirklichung dieser Bedingungen erfolgt analog zu den vorangegangenen Implementierungen von relativen Häufigkeiten durch das *among*-Constraint. Aus diesem Grund wird hier nicht näher darauf eingegangen. Es wurde bereits beschrieben, warum die zulässige Geschwindigkeit einer Strecke ausschlaggebend für ihre Infrastruktur ist. Daher sollen die zulässigen Geschwindigkeiten der Strecken für jede Streckenkategorie in Form eines Intervalls von der Spezifikation übergeben werden. Dieses Intervall bestimmt, in welchen Bereich die Höchstgeschwindigkeiten der Strecken einer Kategorie liegen müssen. Aufbauend auf diesen Angaben können zusätzliche Infrastrukturdaten, welche in dieser Arbeit nicht behandelt worden sind, durch zusätzliche Funktionen erzeugt werden.

Zusätzlich wurden für diese Generierungsphase zwei optionale Bedingungen implementiert, die sich aus in der Realität durch die Streckenkategorien ergeben. Zum einen sollen Strecken, die mit mehr als 160 km/h befahren werden immer als Strecken mit Fernverkehrsströmen gelten und zum anderen sollen Strecken, die zwei sich in Ballungsräume befindende Knoten verbindet keine Nebenbahnen sein.

4.5.1 Kantenzüge im Graphen verlegen

Neben dem Klassifizieren der Streckentypen muss die Forderung nach den auf Seite 50 beschrieben Kantenzügen über diese Streckentypen berücksichtigt werden. Der folgende Algorithmus zur

Verlegung dieser Kantenzüge kann als Suchalgorithmus verstanden werden, weil er versucht Kanten des Graphen Werte zuzuweisen. Wird in diesem Algorithmus versucht einem Kantenattribut ein Wert zuzuweisen, werden durch den Constraint-Löser alle mit diesem Attribut verbundenen Constraints geweckt und geprüft ob die Konsistenz dieses CSP durch diese Wertzuweisung erhalten bleibt. Durch die Verwendung einer logischen Programmiersprache bietet sich das Backtracking für die Implementierung des folgenden Suchalgorithmus an. Im Folgenden wird der auf Seite 77 angegebene Algorithmus näher erläutert.

Der Graph wird durch die Knoteninstanzen $N := \{n_1, \ldots, n_m\}$ und den Kanteninstanzen $E := \{e_1, \ldots, e_p\}$ repräsentiert. Zudem werden vom Anwender die gewünschten Streckenzüge $L := \{l_1, \ldots, l_k\}$ angegeben. Die Streckenzüge werden in der Applikation als *lines* bezeichnet. Zu jedem Streckenzug l_i wird sein Streckentyp, seine Länge und die Streckentypen zu denen dieser Streckenzug kompatibel ist, spezifiziert. Der l_i zugeordnete Streckentyp heißt hier typ_i. $länge_i$ bezeichnet die Länge von l_i und stellt die geforderte Anzahl der Kanten dar. Die Menge der kompatiblen Streckentypen ist $KompTyp_i$. Zusätzlich können zu jedem Streckenzug l_i optional weitere Attribute $Attr_i$ angegeben werden, die die Kanten des Streckenzugs erfüllen müssen. Diese Attribute sind die im Modellierungsabschnitt 3.3.2 angesprochenen Verkehrstypen. Der Faktor, mit dem die Ausdehnung der Kantenzüge reguliert wird, wird im Algorithmus mit f bezeichnet.

Um die Kantenzüge im Graphen zu platzieren wird die Prozedur `PLATZIERE_ALLE_KANTENZUEGE` aufgerufen. Zuerst wird in Zeile 1 ein Kantenzug l_i mit $i \in \{1, \ldots, k\}$ ausgewählt, der durch die Prozedur `PLATZIERE_KANTENZUG` im Graphen verlegt werden soll. Gelingt dies nicht, ist die Suche beendet und das Problem kann nicht gelöst werden. Kann der Kantenzug hingegen doch verlegt werden, wird l_i als bereits verlegt notiert (Zeile 6) und `PLATZIERE_ALLE_KANTENZUEGE` rekursiv aufgerufen. Durch diesen rekursiven Aufruf wird ein weiterer noch nicht verlegter Kantenzug l_j versucht im Graphen zu platzieren. Um l_i zu verlegen wird in der Prozedur `PLATZIERE_KANTENZUG` in Zeile 11 von den Knoten des Graphen ein Startknoten ausgewählt, von dem aus der Graph traversiert wird. Anschließend wird durch die Prozedur `SUCHE_PASSENDEN_KANTENZUG` versucht ausgehend vom gewählten Startknoten einen Kantenzug zu finden (Zeile 14). Kann kein Kantenzug gefunden werden, wird ein anderer Startknoten gewählt bis die Suche erfolgreich ist oder kein Startknoten mehr ausgewählt werden kann. Für die Suche eines Kantenzuges wird in Zeile 18 in `SUCHE_PASSENDEN_KANTENZUG`

zunächst ein weiterer Knoten $n_{i,\,nächster}$ bestimmt, der mit dem Startknoten adjazent ist. Der Startknoten wird zu diesem Zeitpunkt als aktuell betrachteter Knoten $n_{i,\,aktuell}$ angesehen. Die Kante, die diese beiden Knoten verbindet, wird in den Zeilen 23 bis 30 mit dem Streckentyp typ_i und den geforderten Attributen $Attr_i$ belegt, bzw. beim Streckentyp auf Kompatibilität geprüft. Ist dies nicht möglich, wird ein anderer adjazenter Knoten gewählt bei dem dies gelingt. Existiert kein weiterer Knoten, der adjazent ist und bei dem dies möglich ist, wird die Prozedur verlassen und das Scheitern durch *false* symbolisiert. Konnte ein Knoten $n_{i,\,nächster}$ gefunden werden, dessen inzidente Kante zum Knoten $n_{i,\,aktuell}$ die Anforderungen erfüllt, wird in Zeile 32 geprüft, ob die Anzahl der bisher verlegten Kanten von l_i der Länge des gewünschten Kantenzuges entsprechen. In diesem Fall würde der verlegte Kantenzug l_i in Form aller seiner bis dahin besuchten Knoten vermerkt werden. In Zeile 36 wird die Prozedur `SUCHE_PASSENDEN_KANTENZUG` durch einen rekursiven Aufruf wieder ausgeführt. Dieses mal ist der vorher bestimmte Knoten $n_{i,\,nächster}$ der aktuell betrachtete Knoten. Ausgehend von diesem Knoten wird jetzt ein adjazenter Knoten gewählt, der vorher noch nicht betrachtet worden ist und die vorher beschriebenen Anforderungen erfüllt. Dies wird solange wiederholt, bis alle Kanten des Kantenzugs platziert worden sind. Nachdem ein Kantenzug l_i verlegt worden ist, und die Suche für einen weiteren Kantenzug l_j ausgeführt wird, kann es aufgrund der durch l_i belegten Werte im Graphen dazu führen, dass l_j ausgehend von seinem Startknoten nicht verlegt werden kann. In diesem Fall wird die Suche für l_j mit anderen Startknoten durchgeführt bis die Suche erfolgreich war oder festgestellt wurde, dass l_j nicht platzierbar ist. Wenn der Kantenzug l_j nicht platzierbar ist, wird durch Backtracking die Suche von l_i wieder aufgenommen und versucht eine alternative Platzierung für l_i zu finden. Ist dies der Fall, kann die komplette Suche für l_j erneut beginnen. Ist eine weitere Platzierung für den Kantenzug l_i nicht möglich, werden analog die davor verlegten Kantenzüge versucht anders zu platzieren. Ist es nicht möglich eine alternative Platzierung für diese Kantenzüge zu erreichen, ist die Suche gescheitert. Ist es möglich durch die wieder aufgenommene Suche von l_j eine Lösung zu finden, werden danach die restlichen Kantenzüge versucht zu platzieren. Die Suche ist erfolgreich, wenn alle Kantenzüge $L := \{l_1, \ldots, l_k\}$ im Graphen verlegt worden sind.

Algorithmus 2:

$B := \emptyset$ *%bereits platzierte Kantenzüge*

```
PLATZIERE_ALLE_KANTENZUEGE
 1:  wähle l_i ∈ L\B  %wähle eine Linie aus den noch nicht verlegten Linien
 2:  platziert := PLATZIERE_KANTENZUG( l_i )
 3:  if ( platziert = false) then
 4:    return false
 5:  fi
 6:  B := B ∪ {l_i}          %füge l_i zu den bereits platzierten Linien hinzu
 7:  platziert := PLATZIERE_ALLE_KANTENZUEGE    %rekursiver Aufruf
 8:  if ( platziert = false) then
 9:   GOTO 2    %Backtracking, versuche l_i alternativ zu platzieren
10:  fi
END
```

```
PLATZIERE_KANTENZUG( l_i )
11:  wähle einen Startknoten n_{i,start} ∈ N , der noch nicht gewählt wurde, sonst
                                                    return false
12:  n_{i,aktuell} := n_{i,start}      %Der betrachtete Knoten ist der Startknoten
13:  Besucht_i := ∅                    %Die Menge der besuchten Knoten der Linie l_i
14:  gefunden := SUCHE_PASSENDEN_KANTENZUG( l_i )
15:  if ( gefunden = false) then
16:   GOTO 11          %bei Misserfolg, wähle neuen Startknoten n_{i,start}
17:  fi
END
```

```
SUCHE_PASSENDEN_KANTENZUG( l_i )
18: wähle einen Knoten n_{i,nächster} ∈ (N \ Besucht_i) , der mit n_{i,aktuell} adjazent ist
                                  und noch nicht gewählt wurde, sonst
                                  return false
19: if ( d(n_{i,start}, n_{i,nächster}) < d(n_{i,start}, n_{i,aktuell}) * f ) then
20:   GOTO 18   %Kriterium nicht erfüllt, wähle neuen Knoten n_{i,nächster}
21: fi
22: wähle die Kante e_j ∈ E , die n_{i,aktuell} und n_{i,nächster} verbindet
23: gueltige_kante := (e_j.lineType # = typ_i) ∨ (e_j.lineType ∈ KompTyp_i)
```

```
24: if (gueltige_kante = false) then
25:   GOTO 18   %Streckentyp der Kante e_j ist nicht kompatibel, wähle
                %neuen Knoten n_{i, nächster}
26: fi
27: zusatz := die Attribute attr_i der Kante e_j zuweisen bzw. vergleichen
28: if (zusatz = false) then
29:   GOTO 18         %Attribute der Kante e_j passen nicht zu attr_i , wähle
                      %neuen Knoten n_{i, nächster}
30: fi
31: Besucht_i := Besucht_i ∪ {n_{i, aktuell}}
32: if ( |Besucht_i ∪ {n_{i, nächster}}| = länge_i + 1 ) then
33:   return Besucht_i ∪ {n_{i, nächster}}    %Linie platziert, beende Prozedur
                                              %bei Backtracking wird hier eingesetzt
                                              %und GOTO 18 ausgeführt
34: fi
35: n_{i, aktuell} := n_{i, nächster}    %rufe diese Prozedur für den nächsten Knoten auf
36: gefunden := SUCHE_PASSENDEN_KANTENZUG( l_i )
37: if (gefunden = false) then
38:   GOTO 18 %Restkantenzug nicht platzierbar,wähle neuen Knoten n_{i, nächster}
39: fi
END
```

Es sei hier noch anzumerken, dass ein Kantenzug im Programm nur dann verlegt wird, wenn er sich in mindestens einer Kante von anderen Kantenzügen unterscheidet. Dies hat den Grund, dass sich mehrere spezifizierte kompatible Kantenzüge nicht komplett überlagern sollen. Zur Vereinfachung wurde dies im Algorithmus nicht dargestellt. Es wäre auch die Realisierung denkbar, dass sich zwei spezifizierte Kantenzüge in keiner Kante überlagern dürfen.

Ein Problem stellt die Tatsache dar, dass spezifizierte Kantenzüge in ihrer Länge, die am Anfang dieses Kapitel besprochene, geforderte Höchstanzahl an Kanten ihres Streckentyps im Graphen übersteigen können. Somit kann in solch einem Fall keine Lösung gefunden werden. Dies wurde allerdings nicht vor dem Verlegen der Kantenzüge erkannt, was zu unnötigen Rechenaufwand führt. Da die Höchstanzahl der Kanten eines Streckentyps, aufgrund des Constraints über das Vorkommen von Streckentypen, relativ von der Anzahl aller im Graph existierenden Kanten ist, wurde ein

Constraint implementiert, welches anhand der spezifizierten Kantenzüge ermittelt, wie viele Kanten mindestens im Graphen vorhanden sein müssen, um die Kantenzüge verlegen zu können. Dieses Constraint wird bereits vor dem Verbinden der Knoten (siehe Kapitel 4.4) abgesetzt. Somit muss die Auswahl der tatsächlich verwendeten Kanten die geforderte Mindestanzahl erfüllen. Beispielsweise sollen zwei Kantenzüge verlegt werden. Ein Kantenzug soll sieben Kanten vom Typ 1 besitzen und ein Kantenzug soll zwei Kanten vom Typ 2 besitzen. Kanten vom Typ 1 dürfen höchstens 75% des Graphen belegen. Dagegen dürfen Kanten vom Typ 2 höchstens 25% in Anspruch nehmen. Weil für den Kantenzug mit den sieben Kanten $7/0{,}75 = 9{,}\overline{3}$ und für den Kantenzug mit den zwei Kanten $2/0{,}25 = 8$ gilt, müssen mindestens zehn Kanten im Graphen vorhanden sein, damit eine Belegung von sieben Kanten des Typs 1 möglich ist. In diesem Fall würde der Kantenzug mit seinen sieben Kanten 70% der zehn Kanten belegen. Unter den verbleibenden drei Kanten muss der andere Kantenzug verlegt werden. Dieses Constraint berücksichtigt nicht die Möglichkeit, einen Kantenzug auf kompatible Streckentypen zu verlegen. Dies wäre ein Ansatzpunkt zur Erweiterung des Verfahrens.

Weil die spezifizierten Kantenzüge in der Regel nur einen Teil der Kanten des Graphen belegen, wird eine gültige Belegung für die Streckentypen der verbleibenden Kanten zum Schluss dieser Generierungsphase gesucht. Abbildung 4.6 zeigt den auf Seite 50 verwendeten Beispielgraphen mit drei verlegten Kantenzügen. Ein Kantenzug besitzt eine Länge von fünf Kanten und ist vom Streckentyp 2. Auf diesen Strecken soll Fernverkehr gelten. Ein Kantenzug besitzt drei Kanten vom Streckentyp 2 und soll für Güterverkehr ausgerüstet sein. Der dritte Kantenzug verfügt über zwei Kanten vom Streckentyp 3 und wird für den Nahverkehr verwendet. Den verbleibenden Kanten wurde durch Labeling Streckentyp 1 aber kein Verkehrstyp zugewiesen.

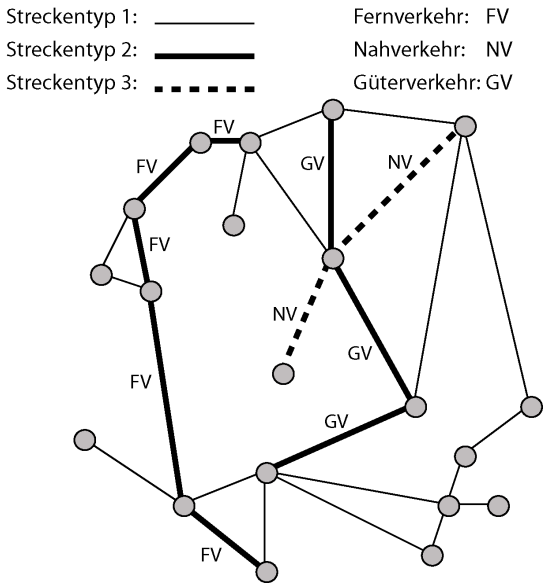

Abbildung 4.6: Drei Kantenzüge mit entweder Fern-, Nah- oder Güterverkehr

4.6 Generierung der Bahnhöfe

In diesem Abschnitt werden die implementierten Methoden erklärt, die zur Berechnung der konkreten Daten der Bahnhöfe führen. Für die Speicherung und Weiterverarbeitung dieser Daten wurde eine neue Klasse eingeführt. Die Instanzen dieser Bahnhofsklasse repräsentieren im Gegensatz zu der Knotenklasse nicht nur Knotenbahnhöfe, sondern auch die Unterwegsbahnhöfe des Eisenbahnnetzes. Die Modellierung für diese Implementierung wurden im Abschnitt 3.5 beschrieben.

4.6.1 Ausrichtung der Bahnhöfe

Bevor die Generierung der Gleistopologie eines Bahnhofs begonnen werden kann, muss zunächst geklärt werden, in welche Richtung der Bahnhof ausgerichtet wird. Das ist dadurch begründet, dass aufbauend auf dieser Ausrichtung erst festgestellt werden kann, welche Streckengleise an welchen der beiden Bahnhofsköpfe anliegen sollen. In dieser Arbeit wurde aufgrund des Aufwands auf die Modellierung von Überwerfungen von Gleisen verzichtet. Damit Streckengleise sich an den Bahnhofsköpfen nicht unnötig überkreuzen, werden sie gemäß der Ausrichtung des Bahnhofs nach ihrer Lage mit den jeweiligen Bahnhofsköpfen verbunden. Es stellt sich die Frage, wie ein Bahnhof ausgerichtet werden kann. Ausgehend von der Forderung die Strecken direkt ohne Umwege an einen Bahnhofskopf anzuschließen, ergibt sich Zahl der möglichen Ausrichtungen eines Bahnhofs

aus der Anzahl seiner zulaufenden Strecken. Die Anzahl der möglichen Kombinationen die Strecken an den beiden Bahnhofsköpfen zu verknüpfen ist höchstens der Anzahl der Strecken. Dies soll am folgenden Beispiel veranschaulicht werden.

Beispiel: Der in Abbildung 4.7 dargestellte Knoten soll ausgerichtet werden. An diesem Knoten liegen die vier Strecken A, B, C und D an. Die vier grundlegenden Möglichkeiten den Knoten nach den Verbindungskombinationen seiner Strecken auszurichten, wird durch seine Querachsen[8] w, x, y und z dargestellt. Würde der Knotenbahnhof nach der Querachse w ausgerichtet werden, würde ein Bahnhofskopf mit den drei Strecken B, C und D und der andere Bahnhofskopf mit Strecke A verbunden sein. Bei einer Ausrichtung nach der Querachse x, würden die beiden Bahnhofsköpfe jeweils mit zwei Strecken (entweder Strecke B und C oder D und A) verknüpft werden. Für die Querachse y wären wiederum jeweils zwei Strecken (C und D oder A und B) mit einem Bahnhofskopf verbunden. Die letzte Möglichkeit, dargestellt durch die Querachse z, würde einen Bahnhofskopf mit der Strecke B und den anderen Bahnhofskopf mit den Strecken C, D und A verbinden. Für jede dieser Querachsen besteht ein gewisser Spielraum, den die Achse im Anstieg verändert werden kann, ohne die Zuordnungen der Strecken zu den Bahnhofsköpfen zu verändern.

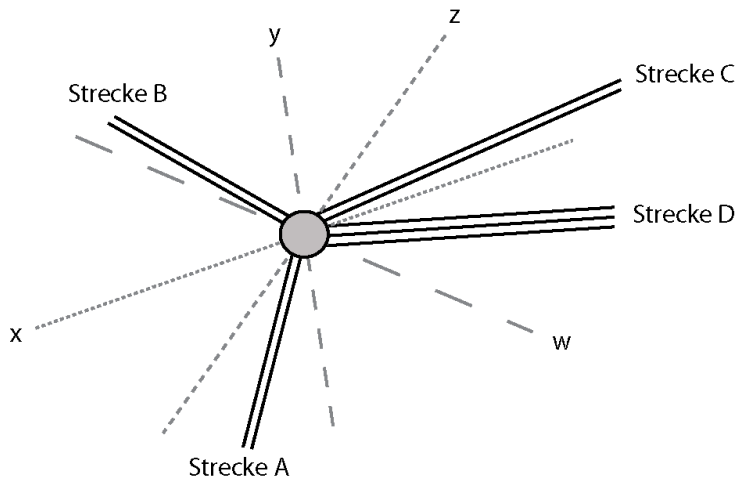

Abbildung 4.7: Ausrichten eines Knoten mit vier zulaufenden Strecken

Die Ausrichtung der Bahnhöfe wurde durch einen Algorithmus implementiert, der zusätzlich noch Angaben in der Spezifikation zu den gewünschten Streckenverteilungen an den Bahnhofsköpfen berücksichtigt. Die Verteilungen der Strecken kann durch zwei Werte $min_{vert} \in \mathbb{R}$ und $max_{vert} \in \mathbb{R}$ mit $0 \leq min_{vert} \leq max_{vert} < 0{,}5$ bestimmt werden. Die Werte grenzen die Möglichkeiten ein, in dem die Verteilung der Anzahl der Strecken pro Bahnhofkopf liegen soll. Beispielsweise würde der

8 Die Querachse eines Bahnhofs ist die Achse, die orthogonal zu den Gleisen der Gleisgruppe liegt.

Algorithmus für $min_{vert} = 0{,}4$ und $max_{vert} = 0{,}499$ versuchen eine Kombination zu finden, die die Bahnhofsköpfe mit ungefähr der gleichen Anzahl von Strecken verbindet.

4.6.2 Berechnung der Gleisgruppe

Wie bereits im Abschnitt 3.5.1 beschrieben, ist die Dimensionierung der Gleisgruppe eines Bahnhofs abhängig von seinen einmündenden Streckengleisen. Aus diesem Grund werden die notwendigen Gleisdaten aus den Streckendaten extrahiert und zur Weiterverarbeitung aufbereitet. Diese Gleisdaten werden für die beiden Bahnhofsköpfe getrennt berechnet. Für jedes anliegende Gleis wird die Richtung gespeichert, in der das Gleis signaltechnisch befahren werden darf. Weiterhin wird geprüft, ob für den Verkehr auf diesem Gleis eine ungehinderte Einfahrt gefordert wird. Ausgehend von diesen Streckengleisdaten kann die Anzahl der benötigten weiterführenden Hauptgleise und deren Zuordnung zu den jeweiligen Streckengleisen durch Constraints bestimmt werden. Die Anzahl der weiterführenden Hauptgleise unterliegt den im Abschnitt 3.5.1 erläuterten Bedingungen. Die Zuweisung der weiterführenden Hauptgleise zu den Streckengleisen stellt ein eigenständiges CSP dar und wird anhand des oben eingeführten Beispielbahnhofs erläutert. Es wird der Fall betrachtet, in dem der Knotenbahnhof an der Querachse y ausgerichtet worden ist. Daher liegen die zweigleisigen Strecken A und B an einem Bahnhofskopf an, wohingegen die zweigleisige Strecke C und die dreigleisige Strecke D am anderen Bahnhofskopf anliegen (Abbildung 4.8). Durch die Forderung, dass mindestens so viele weiterführende Hauptgleise existieren, wie Streckengleise an einem Bahnhofskopf einmünden, werden wegen den Gleisen der Strecken C und D mindestens fünf weiterführende Hauptgleise in der Gleisgruppe benötigt. Weiterhin sollen ungehinderte Einfahrten für alle Streckengleise außer dem Streckengleis C1 gelten. Das Streckengleis C1 wird daher grau dargestellt. In der Abbildung werden die restlichen Streckengleise mit geforderter ungehinderter Einfahrt schwarz gekennzeichnet. Die Fahrtrichtungen in denen auf den Streckengleisen gefahren werden darf, werden durch Pfeile symbolisiert.

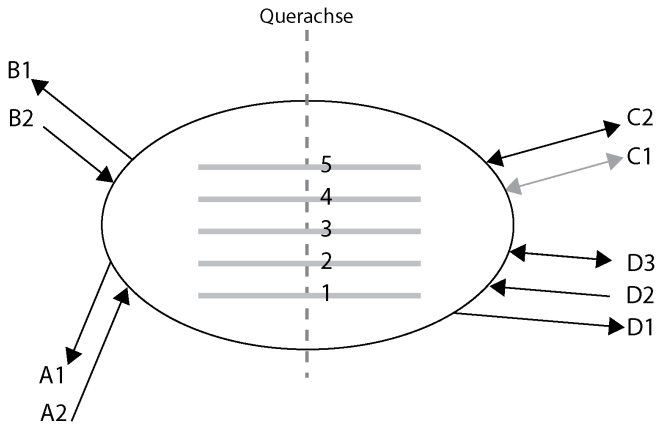

Abbildung 4.8: visualisierte Streckengleisdaten am Beispielknoten

Bei der Suche wird versucht, eine Lösung mit so wenig wie möglich weiterführenden Gleisen zu finden. Die Lösung des CSP stellt somit eine gewisse optimale Zuordnung dar. Jedes weiterführende Hauptgleis wird, wie in der Abbildung dargestellt, durch eine Nummer repräsentiert. Im Folgenden wird auf die in diesem CSP auftretenden Besonderheiten anhand des Beispiels näher eingegangen. Zuerst wird versucht eine Zuordnung mit fünf weiterführenden Hauptgleisen zu finden. Aufgrund dessen, dass sich die Streckengleise nicht überkreuzen dürfen, steht die Zuordnung für die Streckengleise am rechten Bahnhofskopf bereits fest. Streckengleis D1 wird dem Hauptgleis 1, D2 dem Hauptgleis 2, D3 dem dem Hauptgleis 3, C1 dem Hauptgleis 4 und C2 dem Hauptgleis 5 zugewiesen. Am linken Bahnhofskopf darf dem Streckengleis A2 nur das Bahnhofsgleis 1 oder 2 zugeordnet werden, weil den drei anderen Streckengleise A1, B2 und B1 jeweils ein Bahnhofsgleis mit höherer Nummer zugeordnet werden muss. Durch die feste Zuweisung von Bahnhofsgleis 2 zu Streckengleis D2, kann dem Streckengleis A2 nicht das Bahnhofsgleis 2 zugewiesen werden. Sowohl für Streckengleis A2 als auch für D2 wird eine ungehinderte Einfahrt in ihren zugewiesenen Bahnhofsgleisen beansprucht. Dieses Constraint verbietet daher für A2 und D2 die Zuweisung des selben Bahnhofgleises. Dem Streckengleis A2 muss daher das Bahnhofsgleis 1 zugeordnet werden. Diese Zuordnung ist ist gültig. Das dem Bahnhofsgleis 1 ebenfalls zugeordnete Streckengleis D1 hat auch den Anspruch auf ungehinderte Einfahrt, schränkt aber die Bedingungen der Gleiszuordnungen dieses Knotens nicht ein, weil dieses Streckengleis nur für die Fahrt aus dem Knoten genutzt wird. Das Streckengleis B2 kann dem Bahnhofgleis 3 oder 4 zugeordnet werden, weil das Streckengleis A1 unterhalb und B1 oberhalb von B2 zugeordnet sein müssen. Aufgrund nicht einzuhaltender ungehinderter Einfahrten durch Streckengleis D3 auf dem Bahnhofsgleis 3, wird dem Streckengleis B2 das Bahnhofsgleis 4 zugewiesen. Dies ist zulässig, weil das an der anderen Seite des Bahnhofsgleises zugeordnete

Streckengleis C1 keinen Anspruch auf ungehinderte Einfahrt besitzt. Durch diese Zuordnung muss dem Streckengleis B1 das Bahnhofsgleis 5 zugewiesen werden. Diese Zuordnung verletzt kein Constraint, da B1 nur für die Ausfahrt vorgesehen ist und somit die ungehinderte Einfahrt vom Streckengleis C2 nicht beeinflusst wird. Folglich bleiben für das Streckengleis A1 nur noch die Bahnhofsgleise 2 und 3 übrig. Das Streckengleis A1 wird ebenfalls nur für die Ausfahrt aus dem Knoten genutzt und kann daher einem der beiden Bahnhofsgleise zugeordnet werden. Für weitere Erläuterungen wird der Fall betrachtet, dass A1 dem Bahnhofsgleis 2 zugeordnet wird (Abbildung 4.9). Wie in diesem Beispiel zu sehen ist, können für die Zuweisungen der Streckengleise zu den weiterführenden Hauptgleisen eines Bahnhofs mehrere Lösungen existieren. Dies bietet die Möglichkeit, in zukünftigen Arbeiten hierfür zusätzliche Constraints zu modellieren und zu implementieren. Aus den vorgesehenen Fahrtrichtungen, der mit einem weiterführenden Hauptgleis verbundenen Streckengleise lassen sich unmittelbar die Fahrtrichtungen ermitteln, für die das weiterführende Hauptgleis signaltechnisch ausgestattet sein muss. In der Abbildung 4.9 werden die Fahrtrichtungen durch graue Pfeile dargestellt.

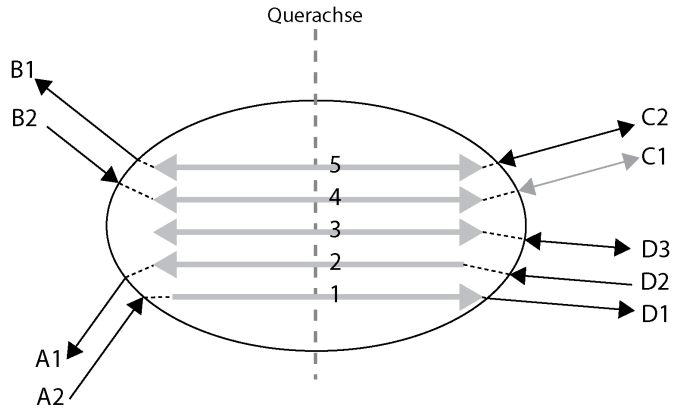

Abbildung 4.9: den Streckengleisen zugewiesene Bahnhofsgleise

Im Anschluss werden der Gleisgruppe zu den weiterführenden Hauptgleisen noch Überholungsgleise hinzugefügt. Gemäß der Modellierung auf Seite 60 werden Daten generiert, die die implementierten Constraints erfüllen.

4.6.3 Berechnung der Fahrstraßenknoten

Nach dem Erzeugen der Gleisgruppe müssen die beiden Bahnhofsköpfe in Form von Fahrstraßenknoten berechnet werden. Ein Fahrstraßenknoten besteht aus einer Menge von Gleisen und Gleisverbindungen. Um aussagekräftige Daten zu generieren, muss festgelegt werden in welcher Form und welchen Umfang die Gleise und Gleisverbindungen dargestellt werden. Die im

folgenden beschriebene Datenstrukturen orientieren sich an RailML, welches als zukünftiges Standardformat für den Austausch von Daten zwischen Anwendungen im Bahnwesen entwickelt wird. Begründend für diese Herangehensweise ist, dass eine Implementierung einer Exportfunktion der Datenstrukturen in das RailML-Format, bei zukünftiger Weiterentwicklung vorliegenden Programms, erleichtert wird.

Datenstruktur eines Gleises

Für die Repräsentation eines Gleises werden folgende Daten erzeugt:

- der Anfang des Gleises
- das Ende des Gleises
- der Gleistyp
- die Gleislänge
- die Richtungen, der auf dem Gleis im Regelbetrieb durchgeführten Fahrten
- eine Liste der dem Gleis zugeordneten Gleisverbindungen

Diese Attribute können im RailML-Format zur Beschreibung eines Gleises ebenfalls angegeben werden. Der Gleisanfang und das Gleisende werden durch Identifikatoren auf das Gleis begrenzende Infrastrukturelemente angegeben. Diese Infrastrukturelemente sind andere Gleise, Gleisverbindungen oder Prellböcke[9]. Ein Gleis wird eindeutig durch Gleisanfang und Gleisende identifiziert. Für die Verarbeitung eines Gleises mit RailML wird nur die Angabe des Gleisanfangs, des Gleisendes und eine eindeutige Bezeichnung gefordert. Diese Anforderung ist somit bereits erfüllt. Die restlichen Attribute sind optional in RailML anzugeben und dienen der konkreteren Beschreibung.

Beim Gleistyp wird in Fahrstraßenknoten zwischen Bahnhofsgleis und Verbindungsgleis unterschieden. Entsprechend dem RailML-Schema Infrastruktur [Rail04b] werden die Typen durch die Bezeichner *stationTrack* und *connectingTrack* angegeben. Einzige Ausnahme bilden die äußersten Bahnhofsgleise des Fahrstraßenknotens. Sie werden zur Vereinfachung einer verarbeitungstechnischen Sonderbehandlung als *outerStationTrack* gekennzeichnet.

Die Gleislänge wird als Kilometrierungswert in Form einer Dezimalzahl dargestellt. Die Kilometrierung wird vom Gleisanfang bis zum Gleisende durchgeführt.

9 engl. buffer stop

Die Fahrtrichtungen in den Fahrstraßenknoten werden durch die Bezeichner *in*, *out* und *both* angegeben. Wird ein Gleis im Fahrstraßenknoten ausschließlich für die Einfahrt in den Bahnhof verwendet wird der Bezeichner *in* verwendet. Analog wird für ausschließlich vorgesehene Ausfahrten der Bezeichner *out* und für das Befahren des Gleises in beiden Richtungen *both* verwendet.

Die dem Gleis zugeordneten Gleisverbindungen werden in einer Liste, nach ihren Positionen auf dem Gleis geordnet, gespeichert. Jede Gleisverbindung wird wie in RailML genau einem Gleis zugeordnet.

Datenstruktur einer Gleisverbindung

Für jede Gleisverbindung werden folgende Attribute erzeugt:

- der Name der Gleisverbindung, der als Identifikator dient
- der Typ der Gleisverbindung
- die Position auf dem Gleis, zu dem die Gleisverbindung zugeordnet wird
- die Ausrichtung der Gleisverbindung auf dem Gleis
- die Richtung der Verzweigung des Gleises, zu dem die Gleisverbindung zugeordnet wird

Diese Arbeit beschränkt sich bei der Verwendung von Gleisverbindungen ausschließlich auf die in auf Seite 20 beschriebenen einfachen Weichen. Die Gründe dafür sind zum einen die in der Praxis vorkommende quantitative Überzahl dieser Art von Weichen gegenüber anderen Weichenarten und zum anderen die fehlenden Randbedingungen zum ersatzweisen Verwenden von anderen Weichenarten.

Der Name einer Gleisverbindung kann eine beliebige Zeichenkette sein, darf aber aufgrund der Identifizierungsfunktion nur einmal verwendet werden.

Der Typ der Gleisverbindung wird durch einen Bezeichner gekennzeichnet und ist hier aufgrund der ausschließlichen Verwendung von einfachen Weichen bei allen Verbindungen gleich. Analog zu der Bezeichnung in RailML wurde hier der Begriff *ordinarySwitch* verwendet.

Die Position der Weiche wird durch den Kilometrierungswert auf dem Gleis in Form einer Dezimalzahl dargestellt.

Die Ausrichtung der Gleisverbindung auf dem Gleis gibt die Richtung der Einmündung in das Gleis

an. Dazu wurden die eingeführten Begriffe von RailML verwendet. Erfolgt die Einmündung der Weiche in Richtung der Kilometrierung des Gleises wird der Bezeichner *incoming* verwendet. Mündet die Weiche entgegen der Kilometrierungsrichtung ein, wird die Weiche mit *outgoing* gekennzeichnet (Abbildung 4.10).

Das Gleis dem eine einfache Weiche zugeordnet wird, ist das Stammgleis der Weiche. Aus der Richtung einer stumpf befahrenen einfachen Weiche betrachtet, liegt das Stammgleis entweder links oder rechts. Dementsprechend wird die Ausrichtung mit *left* bzw. *right* angegeben (Abbildung 4.10).

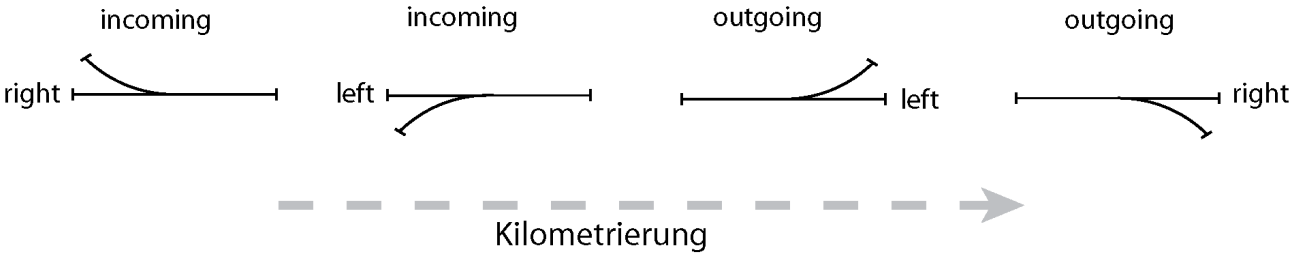

Abbildung 4.10: Beschreibung der Lage einfacher Weichen

Der Algorithmus zur Erstellung eines Fahrstraßenknotens

Die Erstellung der Bahnhofsköpfe eines Bahnhofs erfolgt getrennt voneinander. Sie werden durch den selben Algorithmus berechnet, der ihre seitenverkehrte Ausrichtung an den Bahnhofsenden berücksichtigt. Die Grundidee des Algorithmus ist es, jedes Bahnhofsgleis, von dem der Bahnhof über den betrachteten Fahrstraßenknoten verlassen werden darf, mit jedem hinaus führenden Streckengleis zu verbinden. Damit soll die im Modell gestellte Bedingung erfüllt werden, dass die Ausfahrt über einen Bahnhofskopf zu jeder anliegenden Strecke möglich ist. Der Algorithmus baut auf die im vorherigen Abschnitt erzeugten Gleisgruppendaten auf. Durch die Angaben über die weiterführenden Hauptgleise und die Überholungsgleise wird eine Beschreibung des inneren Bahnhofs geliefert. Diese Daten beschreiben die Zuordnung der weiterführenden Hauptgleise auf die am Bahnhofskopf anliegenden Streckengleise. Wurde ein weiterführendes Hauptgleis an einer Bahnhofsseite keinem Streckengleis zugewiesen, endet dieses weiterführende Hauptgleis in einer Gleisverbindung. Überholungsgleise enden im Fahrstraßenknoten entweder in einer Gleisverbindung oder über einem Stumpfgleis an einem Prellbock. Die Erzeugung des Fahrstraßenknotens durch den Algorithmus erfolgt sukzessiv von der Gleisgruppe zu den anliegenden Streckengleisen. Veranschaulicht betrachtet, wird der Fahrstraßenknoten vom Inneren

des Bahnhofs nach außen hin aufgebaut.

Definition: Aus der Gleisgruppe $G=\{g_1,\ldots,g_n\}$ werden folgende Informationen als Eingabedaten für die Generierung durch den Algorithmus genutzt. Für jedes Gleis g_i aus der Gleisgruppe, sei $g_{i,Ende}$ der Name des Streckengleises mit dem g_i verknüpft ist. Wenn g_i in einer Gleisverbindung endet, sei $g_{i,Ende}=connection$. Ist g_i ein Überholungsgleis und endet an einem Prellbock ist $g_{i,Ende}=buffer\ stop$. Die Fahrtrichtungen mit der ein Gleis g_i befahren werden darf, werden durch $g_{i,Richtung}$ angegeben. Entsprechend den Abbildungen, sei für $1\leq i\leq n$, $g_i\in G$ das i-te Gleis der Gleisgruppe von unten nach oben betrachtet. g_1 ist das unterste und g_n das oberste Gleis der Gleisgruppe.

Der dargestellte Algorithmus wird anschließend an dem oben eingeführten Beispiel demonstriert.

Algorithmus 3:

```
BERECHNE_FAHRSTRAßENKNOTEN
 1:  pos_unten,aktuell := 0
 2:  pos_oben,aktuell := 0
 3: wähle das am weitesten unten liegende Gleis g_unterste Ausfahrt ∈ G, von dem
    aus auf ein Streckengleis gefahren werden darf
 4: wähle das am weitesten oben liegende Gleis g_oberste Ausfahrt ∈ G, von dem
    aus auf ein Streckengleis gefahren werden darf
 5: i := 1
 6: k := n
 7: if ( g_1,Ende = connection ) then
 8:   i := SONDERFALL_UNTEN
 9: fi
10: if ( g_n,Ende = connection ) then
11:   k := SONDERFALL_OBEN
12: fi
13: while ( (g_i ≠ g_oberste Ausfahrt) ∨ (g_k ≠ g_unterste Ausfahrt) ) do
14:    if ( g_i ≠ g_oberste Ausfahrt ) then
15:       GZ := Ø
16:      j := i + 1
```

```
17:     while ( g_{j, Ende} = connection ) do
18:       GZ := GZ ∪ g_j        %Gleise die auf einer Weiche enden, sammeln
19:       j := j + 1
21:     od
22:     pos_neu := ERZEUGE_VERBINDUNG( pos_{unten, aktuell} , g_i , g_j , GZ )
23:     pos_{unten, aktuell} := pos_neu
24:     i := j
25:   fi
26:   if ( g_k ≠ g_{unterste Ausfahrt} ) then
27:     GZ := ∅
28:     l := k - 1
29:     while ( g_{l, Ende} = connection ) do
30:       GZ := GZ ∪ g_l        %Gleise die auf einer Weiche enden, sammeln
31:       l := l - 1
32:     od
33:     pos_neu := ERZEUGE_VERBINDUNG( pos_{oben, aktuell} , g_k , g_l , GZ )
34:     pos_{oben, aktuell} := pos_neu
35:     k := l
36:   fi
37: od
END
```

```
%erzeugt ein Verbindungsgleis von gleis_1 nach gleis_2 und verbindet alle
%Gleise zwischen gleis_1 und gleis_2 , gesammelt in GZ an dem Verbindungsgleis
ERZEUGE_VERBINDUNG( pos_aktuell , gleis_1 , gleis_2 , GZ )
38:  erzeuge ein Verbindungsgleis vg
39:  erzeuge eine hinaus führende Weiche w_1 und speichere sie in gleis_1
     mit der Position pos_aktuell
40:  pos_neu := pos_aktuell + Ausdehnung von vg entlang gleis_2
41:  erzeuge eine einmündende Weiche w_2 und speichere sie in gleis_2 mit
     der Position pos_neu
42:  verknüpfe den Anfang von vg mit w_1 und das Ende von vg mit w_2
43:  pos_vg := 0
44:  while ( (GZ ≠ ∅) ∧ (∀ g ∈ GZ : g noch nicht an Weiche angeschlossen) ) do
```

45: wähle mit $g \in GZ$ das an $gleis_1$ nächstliegendste Gleis
46: $pos_{vg} := pos_{vg} + Regelabstand$
47: erzeuge eine einmündende Weiche w_{vg} und speichere sie in vg mit der Position pos_{vg}
48: verknüpfe das Ende von g mit w_{vg}
49: $GZ := GZ \setminus \{g\}$
50: od
51: return pos_{neu}
END

SONDERFALL_UNTEN
52: j := 2
53: $pos_{gl} := 0$
54: while ($g_{j,\,Ende} = connection$) do
55: $pos_{gl} := pos_{gl} + Regelabstand$
56: erzeuge eine einmündende Weiche w_{gl} und speichere sie in g_1 mit
57: der Position pos_{gl}
58: verknüpfe das Ende von g_j mit w_{gl}
59: j := j + 1
60: od
61: erzeuge eine einmündende Weiche w und speichere sie in g_j mit der Position entsprechend der Ausdehnung von g_1
62: return j
END

SONDERFALL_OBEN
63: l := n-1
64: $pos_{gn} := 0$
65: while ($g_{l,\,Ende} = connection$) do
66: $pos_{gn} := pos_{gn} + Regelabstand$
67: erzeuge eine einmündende Weiche w_{gn} und speichere sie in g_n mit
68: der Position pos_{gn}
69: verknüpfe das Ende von g_l mit w_{gn}
70: l := l - 1

```
71: od
72: erzeuge eine einmündende Weiche w und speichere sie in g_l mit der
    Position entsprechend der Ausdehnung von g_n
73: return l
END
```

Im Folgenden wird die Generierung eines Fahrstraßenknotens exemplarisch an dem in Abbildung 4.11 dargestellten Knotenbahnhof aus dem auf Seite 81 eingeführten Beispiel erläutert. Die Ausführungen beschreiben den Fahrstraßenknoten, der in der Abbildung auf der linken Seite benötigt wird. Dieser Bahnhofskopf verbindet die vier Streckengleise A2, A1, B2 und B1 mit den Gleisen der Gleisgruppe. Für den Bahnhof wurden in der Gleisgruppe zu den fünf weiterführenden Hauptgleisen zusätzlich zwei Überholungsgleise festgelegt. Das Überholungsgleis u_1 liegt unterhalb der weiterführenden Hauptgleise und das Überholungsgleis o_1 befindet sich oberhalb der Hauptgleise. Die weiterführenden Hauptgleise werden zur besseren Differenzierung mit w_1, w_2, w_3, w_4 und w_5 bezeichnet. Die folgenden Erläuterungen beziehen sich auf den linken Bahnhofskopf.

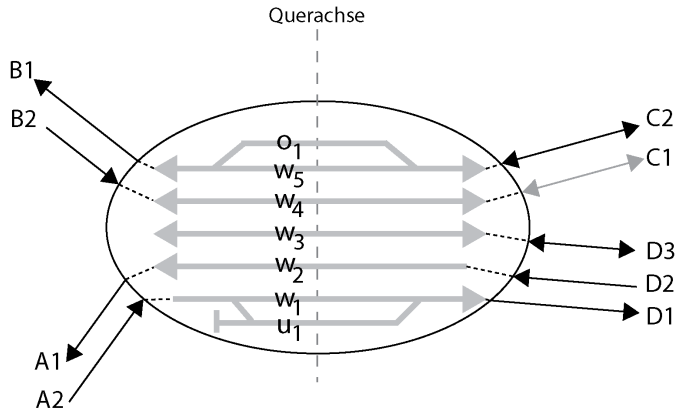

Abbildung 4.11: Ausgangspunkt vor der Erstellung der Bahnhofsköpfe

Das Überholungsgleis u_1 soll auf der linken Bahnhofsseite über ein Stumpfgleis an einem Prellbock enden (dargestellt in der Abbildung). Darum ist $u_{1,Ende} = buffer\ stop$. Das Überholungsgleis o_1 endet mit $o_{1,Ende} = connection$ in einer Gleisverbindung. Durch Zeile 3 und 4 des Algorithmus werden oberstes und unterstes Bahnhofsgleis bestimmt, die für Ausfahrten genutzt werden können. Demnach ist $g_{unterste\ Ausfahrt} = w_2$ und $g_{oberste\ Ausfahrt} = w_5$. w_1 ist für Ausfahrten über den linken Bahnhofskopf nicht ausgerüstet. Anschließend wird geprüft, ob eines der beiden äußersten Gleise der Gleisgruppe in einer Gleisverbindung endet. Dies gilt nur für das oberste Gleis o_1, wodurch die

Prozedur SONDERFALL_OBEN in Zeile 11 aufgerufen wird. Die Prozedur verbindet das Überholungsgleis o_1 durch eine einfache Weiche *sw1* an das Gleis w_5. Diese Weiche wird dem Gleis w_5 zugeordnet (Zeile 72). Die Position dieser Weiche ist von der horizontalen Ausdehnung von o_1 abhängig. Für das Maß der Ausdehnung wurden im Algorithmus Regelwerte verwendet. Gemäß der Position der Weiche auf Gleis w_5 verlängert sich dessen Länge bis zu dieser Position. (Abbildung 4.12a) Nach der Behandlung der Sonderfälle müssen noch die verbleibenden Gleise der Gleisgruppe verbunden werden. Dazu wird alternierend von unten nach oben und von oben nach unten zwischen zwei Gleisen der Gleisgruppe ein Verbindungsgleis verlegt (Zeile 13 bis 37). Ein Verbindungsgleis wird genau dann zwischen zwei Gleisen der Gleisgruppe verlegt, wenn zwischen ihnen kein Gleis oder nur Gleise, die in Weichen enden, liegen. Durch den Algorithmus wird zunächst zwischen dem Überholungsgleis u_1 und w_1 ein Verbindungsgleis in Richtung der Streckengleise verlegt. Dies bedeutet, dass das Gleisende des Verbindungsgleises näher an den Streckengleisen liegen wird als der Gleisanfang. Die Weichen *sw2* und *sw3*, die das Verbindungsgleis mit u_1 und w_1 verbinden, werden u_1 und w_1 zugeordnet. u_1 und w_1 sind für ihre zugeordneten Weichen jeweils das Stammgleis und das Verbindungsgleis bildet für beide Weichen die Abzweigung. Anschließend wird ein Verbindungsgleis von oben nach unten, also von Gleis w_5 nach w_4 verlegt (Abbildung 4.12b). Die Weiche *sw4* auf Gleis w_5, welche mit dem Gleisanfang des Verbindungsgleises verbunden wird, muss auf w_5 weiter in Richtung der Streckengleise positioniert werden als die vorher positionierte Weiche *sw1* auf w_5. Nur so ist es möglich, dass in einer Ausfahrtbewegung von dem Überholungsgleis o_1 über die Weiche *sw1* weiter über *sw4* und *sw5* auf das Gleis w_4 gewechselt werden kann. Aus diesem Grund werden neu positionierte Weichen auf einem Gleis weiter in Richtung der Streckengleise als die bisherigen Weichen auf demselben Gleis verlegt. Veranschaulicht betrachtet, verlängern neu positionierte Weichen die Gleise der Gleisgruppe nach außen. In den Abbildungen richtet sich die Verlängerung nach links. Danach werden wieder zwei Gleise der Gleisgruppe von unten nach oben mit einem Verbindungsgleis verbunden. Dieses Mal wird eine Verbindung von w_1 nach w_2 hergestellt. Die Verbindung wird durch die Weichen *sw6* und *sw7* realisiert. Die nächste Verbindung wird wieder von oben nach unten betrachtet. Das Gleis w_4 wird mit dem nächstliegenden Gleis verbunden, welches nicht in einer Weiche endet. Weil w_3 keinem Streckengleis auf der linken Seite zugeordnet wurde, ist $w_{3,Ende} = connection$. Aus diesem Grund wird w_4 nicht mit w_3 verbunden. Das nächstliegende Gleis, welches die Bedingung erfüllt, ist w_2. Deswegen wird ein Verbindungsgleis

zwischen w_4 und w_2 verlegt (Abbildung 4.12c). Die zugehörigen Weichen *sw8* und *sw9* werden in w_4 bzw. w_2 vermerkt. Weil w_3 zwischen w_4 und w_2 liegt, wird w_3 an das Verbindungsgleis mit der Weiche *sw10* angeschlossen. Dies wird im Algorithmus durch die Zeilen 44 bis 50 realisiert. In diesem Fall ist das Verbindungsgleis das Stammgleis der Weiche *sw10* und w_3 repräsentiert das Abzweiggleis. Durch diese Verbindung ist es möglich über den Fahrstraßenknoten in das Gleis w_3 einzufahren. Darauf folgend wird wieder eine Verbindung von unten nach oben hergestellt. Im Gegensatz zu der vorherigen Verbindung wird das Verbindungsgleis umgekehrt von w_2 nach w_4 verlegt. Das Verbindungsgleis wird durch die Weichen *sw11* und *sw12* begrenzt. In diesem Fall wird das zwischen w_2 und w_4 liegende Gleis w_3 nicht mehr an das Verbindungsgleis angeschlossen, weil w_3 bereits mit der Weiche *sw10* verbunden ist. Diese Bedingung wird in Zeile 44 formuliert. Alternierend wird danach wieder versucht, Gleise von oben nach unten zu verbinden. Dies ist jedoch aufgrund der Bedingung in Zeile 26 nicht mehr notwendig, weil das unterste Gleis aus dem der Bahnhof verlassen werden kann $g_{unterste\ Ausfahrt} = w_2$ ist. w_2 wurde bereits von w_4 aus verbunden und somit wird die Möglichkeit gegeben, von jedem Gleis oberhalb von w_2, eine Ausfahrt über w_2 durchzuführen. Daher wird wieder versucht eine Verbindung von unten nach oben herzustellen. Dies ist möglich, weil das oberste Gleis w_5 aus dem hinaus gefahren werden darf, noch nicht von jedem darunter liegenden Gleis erreicht werden kann. In dieser Iteration wird eine Verbindung von w_4 nach w_5, durch die Weichen *sw13* und *sw14* aufgebaut (Abbildung 4.12d). Durch diese Verbindung ist es möglich, dass von jedem Ausfahrgleis auf das Gleis w_5 gewechselt werden kann. Damit wird die Bedingung in Zeile 13 nicht mehr eingehalten und der Algorithmus wird erfolgreich beendet. Im Anschluss werden die Gleise im Fahrstraßenknoten, die ein weiterführendes Hauptgleis mit einem Streckengleis verbinden, auf eine gemeinsame Länge gesetzt.

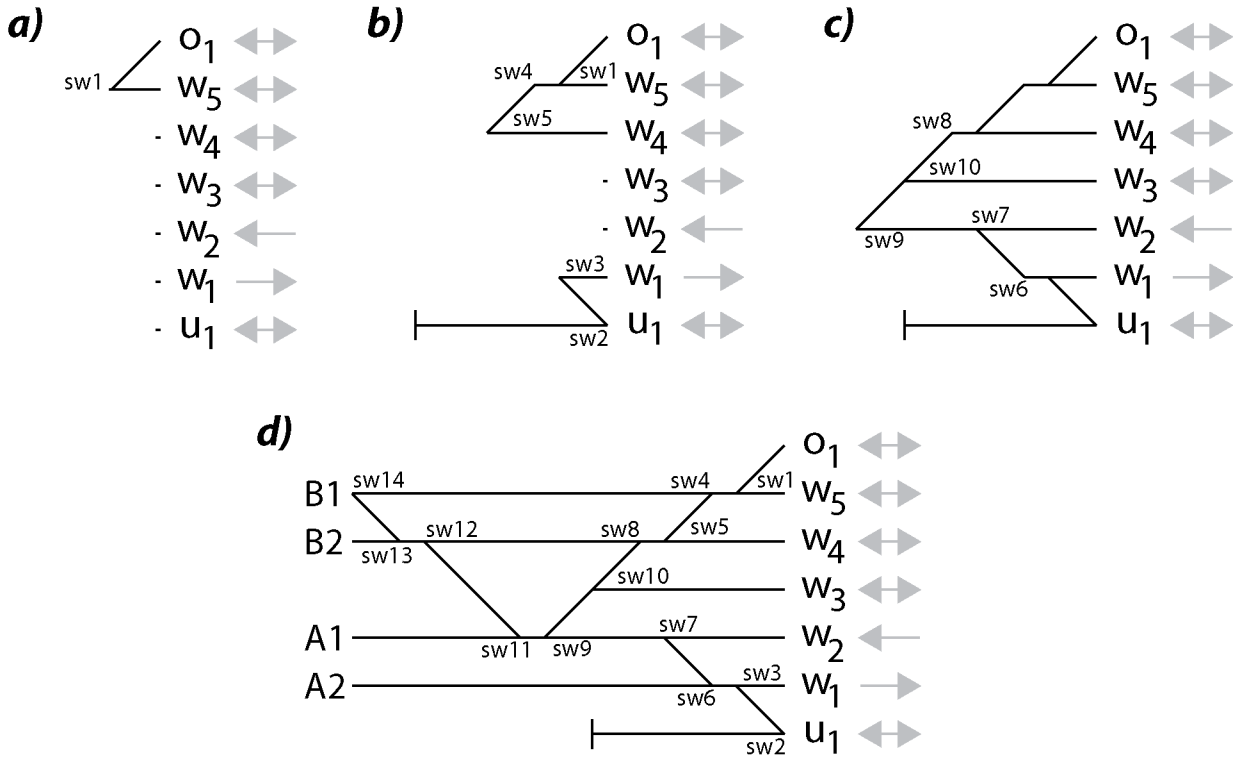

Abbildung 4.12: Generierungsschritte eines Fahrstraßenknotens

Auf Seite 96 sind die generierten Daten zu diesem Fahrstraßenknoten aufgelistet. Die Werte für die in der Abbildung horizontalen Ausdehnung und Längsausdehnung eines Verbindungsgleises orientieren sich an die im [Rail04c] verwendeten Maße. Die horizontale Ausdehnung eines Verbindungsgleises, welches zwei direkt nebeneinander liegende Gleise verbindet wird mit 0,063 berechnet. Die Längsausdehnung für dieses Verbindungsgleis beträgt 0,065. Der Abstand zwischen zwei Weichen wird mit 0,006 angegeben. Aus diesen drei Werten lassen sich die Positionen und Längen ableiten. Die generierten Daten von einem Fahrstraßenknoten werden in zwei Gruppen unterteilt. Eine Gruppe sind die Gleisdaten. Die Daten eines Gleises werden entsprechen ihrer auf Seite 85 erläuterten Datenstruktur in folgender Form gespeichert.

track(Gleistyp, Gleisanfang, Gleisende, Gleislänge, Fahrrichtungen,

Liste der dem Gleis zugeordneten Gleisverbindungen)

Die Gleisverbindungen repräsentieren die zweite Gruppe. Die Daten einer Gleisverbindung werden gemäß ihrer auf Seite 86 definierten Datenstruktur in dieser Form gespeichert.

connection(Name, Typ, Position auf dem Gleis, Ausrichtung, Verzweigungsrichtung)

4.6.4 standardisierte Fahrstraßenknoten

Alternativ zu der Berechnung der Fahrstraßenknoten ist es möglich vordefinierte Fahrstraßenknoten in der Applikation zu verwenden. Diese Option beruht auf dem in [Pa99] vorgeschlagenem Konzept, aus einem vorgegebenen Typenprogramm passende standardisierte Bahnhofsköpfe auswählen zu können. Die Fahrstraßenknoten müssen zu den Gleisen in der Gleisgruppe und den anliegenden Strecken kompatibel sein. In der Implementierung wurden exemplarisch einige Fahrstraßenknoten für kleine Bahnhöfe angegeben. Die Option, vorgegebene Bahnhofsköpfe in die Lösung zu integrieren, ermöglicht die Verwendung von Fahrstraßenknoten, die verschieden zu den durch den Algorithmus generierten Fahrstraßenknoten sind. Die Verwendung von standardisierten Fahrstraßenknoten kann in der Applikation ein- und abgestellt werden. Ist diese Option eingestellt, wird bei passenden Möglichkeiten ein standardisierter Fahrstraßenknoten ausgewählt. In diesem Fall wird die Generierung des Algorithmus übergangen. Wird kein passender vorgegebener Fahrstraßenknoten im Typenprogramm gefunden, wird ein Fahrstraßenknoten durch den Algorithmus generiert..

Die erzeugten Daten des Fahrstraßenknoten aus dem Beispiel:

Gleisdaten

Bahnhofsgleise

```
track(outerStationTrack, o1, sw1, 0.065, both, [])
track(stationTrack, u1, buffer stop, 0.200, both, [SW2])
track(stationTrack, w1, A2, 0.465, in, [SW6, SW3])
track(stationTrack, w2, A1, 0.465, out, [SW11, SW9, SW7])
track(stationTrack, w3, sw10, 0.201, both, [])
track(stationTrack, w4, B2, 0.465, both, [SW13, SW12, SW8, SW5])
track(stationTrack, w5, B1, 0.465, both, [SW14, SW4, SW1])
```

Verbindungsgleise

```
track(connectingTrack, sw2,  sw3,  0.065, both, [])
track(connectingTrack, sw4,  sw5,  0.065, both, [])
track(connectingTrack, sw6,  sw7,  0.065, both, [])
track(connectingTrack, sw8,  sw9,  0.130, both, [SW10])
track(connectingTrack, sw11, sw12, 0.130, both, [])
track(connectingTrack, sw13, sw14, 0.065, both, [])
```

Gleisverbindungen

```
SW1  = connection(sw1, ordinarySwitch, 0.063, incoming, left)
SW2  = connection(sw2, ordinarySwitch, 0, outgoing, right)
SW3  = connection(sw3, ordinarySwitch, 0.063, incoming, right)
SW4  = connection(sw4, ordinarySwitch, 0.069, outgoing, left)
SW5  = connection(sw5, ordinarySwitch, 0.132, incoming, left)
SW6  = connection(sw6, ordinarySwitch, 0.069, outgoing, right)
SW7  = connection(sw7, ordinarySwitch, 0.132, incoming, right)
SW8  = connection(sw8, ordinarySwitch, 0.138, outgoing, left)
SW9  = connection(sw9, ordinarySwitch, 0.264, incoming, left)
SW10 = connection(sw10, ordinarySwitch, 0.065, incoming, right)
SW11 = connection(sw11, ordinarySwitch, 0.270, outgoing, right)
SW12 = connection(sw12, ordinarySwitch, 0.396, incoming, right)
SW13 = connection(sw13, ordinarySwitch, 0.402, outgoing, right)
SW14 = connection(sw14, ordinarySwitch, 0.465, incoming, right)
```

5 Testergebnisse

Die Tests wurden auf einem AMD Athlon 64 3000+ mit 1,8 GHz und 1 GB Arbeitsspeicher durchgeführt. Das Betriebssystem ist Windows XP Professional Version 2002 mit Service Pack 2. Außerdem wurde das *CHIP* System in der Version 5.7.0.0 verwendet.

5.1 Das Testszenario

In diesem Testszenario sollen die Daten für ein Eisenbahnnetz in der Komplexität eines Bundeslandes generiert werden. Zu diesem Zweck wurden die Parameter an dem Eisenbahnnetz von Sachsen-Anhalt orientiert. Mit Hilfe dieser Parameter soll geprüft werden, inwieweit die Implementierung mit Problemen dieser Größenordnung zurecht kommt. Es soll nicht versucht werden, ein Eisenbahnnetz mit genau den selben Parametern wie das Eisenbahnnetz in Sachsen-Anhalt zu generieren. Weiterhin sollen mögliche Schwachstellen in der Implementierung diagnostiziert und Verbesserungsvorschläge erarbeitet werden. Im Folgenden werden die Parameter erläutert, die charakteristisch für dieses Szenario sind. Anschließend werden die in diesen Tests verwendeten Parametereinstellungen aufgelistet. Nach der in dieser Arbeit eingeführten Definition eines Knoten im Eisenbahnnetz, entsprechen 65 Bahnhöfe in Sachsen-Anhalt diesem Kriterium. Insgesamt besitzt Sachsen-Anhalt ungefähr 330 Bahnhöfe (vgl. [DB06b]). Diese Zahl variiert leicht mit der Zeit, da beispielsweise Strecken und die darüber bedienten Bahnhöfe verkauft oder umgebaut werden. Sachsen-Anhalt besitzt nach [Wi07] eine Fläche von 20446,31 km². Dies entspricht einer quadratischen Fläche von ca. 143 km Seitenlänge. Durch die in dieser Arbeit angewandten Interpretation von einer Längeneinheit als einen Kilometer, wird eine quadratische Fläche von 143 mal 143 LE verwendet. Sachsen-Anhalt besitzt keine quadratische Ausdehnung. Dieser Unterschied wird hier vernachlässigt, weil aufgrund der gleichgroßen Fläche die selbe durchschnittliche Ausdehnung des Eisenbahnnetzes erreicht wird. Aus dem Grund der geforderten Ausdehnung, wurde verlangt, dass mindestens ein Knoten in der Nähe der Flächenränder liegen soll. Im Test wurde dies durch die Bedingung formuliert, dass sich mindestens ein Knoten höchstens 14,3 LE von einer Flächenkante entfernt befinden darf. 14,3 LE sind 10 % der Flächenausdehnung in beiden Dimensionen. Dieses Testszenario verfügt über keinen Ballungsraum. Die Kardinalitäten der Knotenbahnhöfe in Sachsen-Anhalt, wurden als grobe Richtwerte für die Forderung der Grade der Knoten im zu erzeugenden Eisenbahnnetz verwendet. Dazu wurden

relative Häufigkeiten angegeben, die das Auftreten von Knoten mit gleichen Grad bestimmen. Diese Angaben dürfen in diesem Testfall zwei bis sechs Prozent vom ermittelten Wert abweichen. Dadurch wird ein kleiner Spielraum für die Lösungsfindung geboten. Es sei hier anzumerken, das in Sachsen-Anhalt so gut wie keine Bahnhöfe mit mehr als vier zulaufenden Strecken auftreten. Dies Spiegelt sich in der anschließenden Spezifikation wider.

Die Streckentypen im Testszenario richten sich nach den von der DB AG in [DB06a] veröffentlichten Streckenkategorien. Die Häufigkeiten der Streckenkategorien von Sachsen-Anhalt bilden die Richtwerte für die Forderung der Verteilung der entsprechenden Streckentypen in diesem Testszenario. Zu der Angabe der Streckentypen soll die Verlegung von Kantenzügen im Eisenbahnnetz getestet werden. Dadurch soll die Funktionsweise des *Algorithmus 2* aus Kapitel 4.5.1 überprüft werden. In folgenden Tests werden dafür fünf Kantenzüge unterschiedlicher Länge und Streckentyps verlegt.

1. Kantenzug: 11 Kanten mit Streckentyp 3
2. Kantenzug: 5 Kanten mit Streckentyp 8
3. Kantenzug: 5 Kanten mit Streckentyp 6
4. Kantenzug: 5 Kanten mit Streckentyp 5
5. Kantenzug: 3 Kanten mit Streckentyp 3

Der in Kapitel 3.3.2 ab Seite 50 beschriebene Faktor für die Verlegung von Kantenzügen von Streckentypen beträgt in allen Tests 1,0. Dadurch wird gefordert, dass ein Kantenzug von Streckentypen sich dem Kantenzuganfang nicht mehr nähern darf. In der Regel dehnt sich der Kantenzug über den Graphen aus.

Die restlichen Parameter in der Spezifikation beziehen sich nicht speziell auf dieses Testszenario. Sie sind allgemeine Annahmen über die Eisenbahninfrastruktur.

5.2 Die Spezifikation für das Testszenario

Die maximale Ausdehnung des Eisenbahnnetzes in beiden Dimensionen ist 143 Längeneinheiten.
mapSizeX(143).
mapSizeY(143).
Das Eisenbahnnetz besitzt 65 Knotenbahnhöfe.
nodes(65).

Insgesamt 330 Bahnhöfe sollen im Eisenbahnnetz sein. (Knoten- und Unterwegsbahnhöfe)

amountOfStations(330).

Die horizontale und vertikale Mindestdistanz zwischen den Knoten soll 3 LE sein.

nodesMinimumDistance(3).

Null bis maximal 20 Unterwegsbahnhöfe kommen auf einer Strecke vor.

Der Abstand zwischen den Unterwegsbahnhöfe beträgt mindestens 2 LE.

intermediateStationsOnALine([0,20],2.0).

Mindestens 2 % bis maximal 100 % der Knoten liegen im nördlichen, südlichen, östlichen und westlichen Bereich.

westzoneNodeProp(0.1,[0.02,1]) :- true.

eastzoneNodeProp(0.9,[0.02,1]) :- true.

southzoneNodeProp(0.1,[0.02,1]) :- true.

northzoneNodeProp(0.9,[0.02,1]) :- true.

Die Gesamtlänge des Eisenbahnnetzes umfasst mindestens 1 LE und maximal 100000 LE.

netLength([1,100000]) :- true.

Es kommen Knotenbahnhöfe mit maximal 8 zulaufenden Strecken vor.

maxCardinalityOfOneNode(8).

Die Angabe der Verteilung der Knotenkardinalitäten.

nodeCardProp(1,[0.15,0.17]). 15 % bis 17 % der Knoten haben eine zulaufende Strecke.

nodeCardProp(2,[0.25,0.3]). 25 % bis 30 % der Knoten haben zwei zulaufende Strecke.

nodeCardProp(3,[0.3 ,0.4]). 30 % bis 40 % der Knoten haben drei zulaufende Strecke.

nodeCardProp(4,[0.12 ,0.18]). 12 % bis 18 % der Knoten haben vier zulaufende Strecke.

nodeCardProp(5,[0.01 ,0.02]). 1 % bis 2 % der Knoten haben fünf zulaufende Strecke.

nodeCardProp(6,[0.01 ,0.02]). 1 % bis 2 % der Knoten haben sechs zulaufende Strecke.

nodeCardProp(7,[0.01 ,0.02]). 1 % bis 2 % der Knoten haben sieben zulaufende Strecke.

nodeCardProp(8,[0.0 ,0.0]). Kein Knoten hat acht zulaufende Strecke.

Es wird aufgrund der leeren Liste kein Ballungsraum spezifiziert.

clusters([]) :- true.

Es werden neun Streckentypen spezifiziert.

maxLineTypes(9).

Die Parametrisierung der Streckentypen, richtet sich grob an den Streckenkategorien der DB AG.

lineTypeProp(1,[0.00 ,0.04],maxV(201,280)). Streckenkategorie F1 (201 bis 280km/h)

lineTypeProp(2,[0.00 ,0.05],maxV(161,200)). Streckenkategorie F2 (161 bis 200km/h)

lineTypeProp(3,[0.2 ,0.3],maxV(101,160)). Streckenkategorie F3 (101 bis 160km/h)

lineTypeProp(4,[0.0 ,0.0],maxV(101,160)). Streckenkategorie F4 (101 bis 160km/h)
lineTypeProp(5,[0.1 ,0.2],maxV(1,120)). Streckenkategorie F5 (1 bis 120km/h)
lineTypeProp(6,[0.05 ,0.15],maxV(101,160)). Streckenkategorie F6 (101 bis 160km/h)
lineTypeProp(7,[0.0 ,0.0],maxV(281,350)). Streckenkategorie Fplus (größer 281km/h)
lineTypeProp(8,[0.3 ,0.45],maxV(1,100)). Streckenkategorie Z1 (1 bis 100km/h)
lineTypeProp(9,[0.05 ,0.15],maxV(1, 50)). Streckenkategorie Z2 (1 bis 50km/h)

Mit dieser Anweisung werden die fünf vorher beschriebenen Kantenzüge verlegt. Die Einsen in den Termen spezifizieren, dass diese Strecken für Nahverkehr ausgelegt sein sollen.

LineTypeLines([line(3,11,[] ,_,1,_),
line(3,3,[] ,_,1,_),
line(8,5,[] ,_,1,_),
line(6,5,[] ,_,1,_),
line(5,5,[] ,_,1,_)]).

Der Ausdehnungsfaktor der Kantenzüge liegt bei 1,0.

lineTypeLinesExpandingFactor(1.0).

Hier wird festgelegt, mit welcher Wahrscheinlichkeit eine Strecke eines Streckentyps ein, zwei, drei oder vier Gleise besitzt. Eine Strecke vom Streckentyp 1 (Streckenkategorie F1) ist eine Fernverkehrsstrecke, und soll nie aus nur einem Gleis bestehen. 50 % bis 100 % der Strecken vom Streckentyp 1 sind zweigleisig. Maximal 20 % sind dreigleisig und maximal 20 % sind viergleisig. Die Beschreibung für die restlichen Streckentypen erfolgt analog.

NumberOfTracksPerLineType(1,[0.0,0.0], [0.5,1.0], [0.0,0.2], [0.0,0.2]). Streckenkategorie F1
numberOfTracksPerLineType(2,[0.0,0.0], [0.5,1.0], [0.0,0.2], [0.0,0.2]). Streckenkategorie F2
numberOfTracksPerLineType(3,[0.0,0.2], [0.0,1.0], [0.0,0.6], [0.0,0.2]). Streckenkategorie F3
numberOfTracksPerLineType(4,[0.0,0.2], [0.0,1.0], [0.0,0.2], [0.0,0.2]). Streckenkategorie F4
numberOfTracksPerLineType(5,[0.0,0.2], [0.0,1.0], [0.0,0.2], [0.0,0.2]). Streckenkategorie F5
numberOfTracksPerLineType(6,[0.0,0.2], [0.0,1.0], [0.0,0.2], [0.0,0.2]). Streckenkategorie F6
numberOfTracksPerLineType(7,[0.0,0.0], [0.8,1.0], [0.0,0.1], [0.0,0.1]). Streckenkateg. Fplus
numberOfTracksPerLineType(8,[0.95,1.0], [0.0,0.1], [0.0,0.0], [0.0,0.0]). Streckenkategorie Z1
numberOfTracksPerLineType(9,[0.95,1.0], [0.0,0.1], [0.0,0.0], [0.0,0.0]). Streckenkategorie Z2

Hier wird spezifiziert, dass auf Strecken der Streckentypen 1, 2 und 7 immer schneller Fernverkehr besteht. Dies sind die Streckenkategorien F1, F2 und Fplus.

longDistanceLineTypes([1,2,7]).

Nebenbahnen sollen Knotenbahnhöfe in Ballungsräumen nicht verbinden.

noLocalRailwaysInClusters :- true.
Strecken mit Fernverkehr dürfen keine eingleisigen Strecken sein.

longDistanceTrafficRunsOnTwoTracksOrMore :- true.
Ein Drittel der Knotenbahnhöfe, durch die Fernverkehr läuft, die aber die Bedingungen nicht erfüllen, um als Fernverkehrshalt zu gelten, wird trotzdem als Fernverkehrshalt angenommen.

regularLongDistanceNodesProp(0.33).
Güter- und Personenzüge müssen nicht auf getrennten Streckengleisen verlaufen.
seperateGoodsAndPassengerTraffic :- fail.
Hiermit werden Streckentypen spezifiziert, deren Strecken immer für den Gleiswechselbetrieb ausgerüstet sein sollen. Dies sind die Fernverkehrsstrecken F1, F2 und Fplus.

twoWayWorkingLineTypes([1,2,7]).
Mindestens 10 % bis 49 % der zulaufenden Strecken sollen an einem der beiden Bahnhofsköpfe einmünden. Knotenbahnhöfe mit mindestens 5 Strecken haben deswegen an jeder Seite mindestens ein zulaufendes Streckengleis.

LinesPerNodePortProportion([0.1,0.49]).
Ein Bahnhof soll maximal vier Überholungsgleise (engl. passing loop) an einer Seite besitzen.
maxPassingLoopsPerStationSide(4).

Hier wird festgelegt, mit welcher Wahrscheinlichkeit Knoten- und Unterwegsbahnhöfe eine bestimmte Anzahl an Überholungsgleisen an den Bahnhofsseiten besitzen dürfen. Der erste Parameter einer Klausel drückt die Anzahl der Überholungsgleise aus, über die eine Aussage getroffen wird. Durch den zweiten Parameter der ersten Klausel wird beispielsweise festgelegt, dass keiner oder höchstens 30 % der Knotenbahnhöfe Null Überholungsgleise an der unteren Bahnhofseite besitzen dürfen. Der dritte Parameter bezieht sich auf die Überholungsgleise oberhalb des Knotenbahnhofs. Der vierte und fünfte Parameter spezifiziert die Verteilung der Überholungsgleise bei den Unterwegsbahnhöfen.

numberOfPassingLoopsProp(0,[0.0,0.3],[0.0,0.3],[0.5,1.0],[0.5,1.0]).
numberOfPassingLoopsProp(1,[0.3,1.0],[0.3,1.0],[0.2,1.0],[0.2,1.0]).
numberOfPassingLoopsProp(2,[0.1,0.3],[0.1,0.3],[0.0,0.1],[0.0,0.1]).
numberOfPassingLoopsProp(3,[0.0,0.1],[0.0,0.1],[0.0,0.0],[0.0,0.0]).
numberOfPassingLoopsProp(4,[0.0,0.1],[0.0,0.1],[0.0,0.0],[0.0,0.0]).

Jeder Bahnhof kann auf nur einer Seite Überholungsgleise besitzen.
oneSidedPassingLoopsStationsProp([0.0,1.0]).

5.3 Testergebnisse

Während der Testläufe wurde der Generierungsprozess in mehrere Abschnitte unterteilt, für die die Laufzeiten getrennt gemessen wurden. Die Zeiten werden in Millisekunden angegeben. Im ersten Abschnitt, für den die Laufzeit überprüft worden ist, wird versucht die Knoten anhand der abgesetzten Constraints zu positionieren (siehe Kapitel 4.3). Im zweiten getesteten Abschnitt wurde das Laufzeitverhalten des auf Seite 70 implementierten Algorithmus zur Bestimmung von Nachbarschaftsbeziehungen zwischen den Knoten untersucht. Anschließend wurde im dritten Testabschnitt das Laufzeitverhalten für das Verbinden der Knoten auf Basis der Nachbarschaftsbeziehungen und der spezifizierten Constraints geprüft. Der darauf folgende Algorithmus für das Verlegen der spezifizierten fünf Kantenzüge stellt den vierten Testabschnitt dar. Die Laufzeit für die verbleibende Generierung wird als der fünfte Abschnitt gewertet.

Positionierung der Knoten

Die Testzeit umfasst die Formulierung der Constraints zur Einhaltung der Mindestabstände, der Randzonenbedingungen und die Zeit für die Suche einer Lösung. Da kein Ballungsraum spezifiziert ist, geht der Verarbeitungsaufwand dieses Constraints nicht mit in die Tests ein. Für die im Testszenario zu positionierenden 65 Knoten werden im Durchschnitt 4041 Millisekunden verwendet. Die kürzeste Laufzeit zur Lösung dieses Problems liegt bei 3969 ms. Die längste Laufzeit beträgt hingegen 4125 ms. Die Differenz zwischen diesen Zeiten beträgt 156 ms. Die Laufzeiten sind fast konstant. Dies liegt zum Einen an der Netzdichte in diesem Testszenario und zum Anderen werden bei der Positionierung der Knoten globale Constraints verwendet. Daher stellt die Positionierung der Knoten in diesem Testszenario kein schwer zu lösendes Problem dar.

Ermittlung von Nachbarschaftsbeziehungen

Der Algorithmus zur Bestimmung von Nachbarschaftsbeziehungen muss für k Knoten im schlimmsten Fall $(k^3-3k^2+2k)/2$ Vergleiche durchführen (siehe Kapitel 4.4.1). Dies sind im Testszenario für die 65 Knoten 131040 Vergleiche. Die Tests zeigen, dass mindestens 29080 und höchstens 39413 Vergleiche durchgeführt werden. Der Durchschnitt liegt bei 33498. Damit werden in diesem Szenario ungefähr ein Viertel (25,56 %) der möglichen Knotenkombinationen für die Ermittlung der Nachbarschaftsbeziehungen herangezogen. Die Ersparnis von drei Viertel der

Vergleiche beruht auf der Tatsache, dass die meisten Knoten aufgrund ihrer Entfernung zueinander nicht benachbart sind und eine Prüfung bereits genügt, um festzustellen, dass zwei Knoten nicht benachbart sind. In diesem Fall werden die restlichen Vergleiche bzgl. zweier Knoten ausgelassen. Die Laufzeit dieses Algorithmus variiert zwischen 2813 und 3922 Millisekunden. Die Testergebnisse zeigen, dass die Laufzeit des Algorithmus proportional zu der Anzahl der Vergleiche steigt. Im Durchschnitt werden bei dieser Spezifikation 3233 ms für den Algorithmus benötigt.

Verbinden der Knoten

In diesem Abschnitt wird versucht, eine Lösung für die Verbindung der Knoten zu finden. Durch Constraints werden die spezifizierten Vorkommen von Knotenkardinalitäten gefordert. Die Netzlänge darf zwischen 1 bis 100000 LE variieren und stellt deswegen keine wesentliche Einschränkung dar. Die ermittelten Laufzeiten für das Absetzen der Constraints und das Verbinden der Knoten liegen bei ca. 16 ms und treten somit kaum in Erscheinung.

Verlegen der Kantenzüge

In den Tests zeigt der in Kapitel 4.5.1 beschriebene Algorithmus große Unterschiede in seiner Laufzeit. Der Algorithmus benötigt für dieses Testszenario in 75 % der Testläufe weniger als 10000 Millisekunden. In ungefähr 63 % der Tests werden weniger als 2000 ms benötigt. Dagegen entstehen relativ lange Laufzeiten in ungefähr 10 % der Durchläufe. In diesen Fällen dauert es mehr als 360000 ms um die Kantenzüge zu verlegen. In vereinzelten Fällen werden Lösungen erst nach mehr als 14 Minuten gefunden. Die große Varianz in den Laufzeiten ist auf die Anzahl benötigter Schritte im Backtracking zurückzuführen, da die Anzahl der Schritte sich proportional zu der Laufzeit des Algorithmus verhält. Ein Grund für diese Varianz sind die zufälligen Startpositionierungen der Kantenzüge. Durch ungünstige Wahl sind somit unverhältnismäßig viele Backtrackingschritte notwendig, um die Verlegung eines Kantenzugs abschließen zu können. Eine weitere Ursache besteht in dem Verhältnis der Anzahl der Kanten des Graphen zu der Anzahl zu verlegener Kanten in den Kantenzügen. Werden beispielsweise Kantenzüge spezifiziert, deren Länge nicht im Graphen untergebracht werden kann, wird dies nicht rechtzeitig erkannt und führt zu exponentiellen Laufzeiten.

Der restliche Generierungsprozess

Unter diesem Abschnitt fällt die Generierung der in Kapitel 3.4 modellierten Streckenparameter und die Generierung der in Kapitel 4.6 beschriebenen Bahnhofsdaten. Die Tests ergeben für diesen Abschnitt Laufzeiten von 656 bis 702 ms bei einer mittleren Laufzeit von 672 ms. Mit einer Varianz

von 46 ms verhält sich die Laufzeit dieses Generierungsabschnitts stabil.

Die folgende Tabelle fasst die ermittelten kürzesten, längsten und mittleren Laufzeiten der Generierungsabschnitte, die sich diesbezüglich stabil verhalten, noch einmal zusammen. Dies sind der erste, zweite, dritte und fünfte Generierungsabschnitt.

Teilabschnitte	*Kürzeste Laufzeit*	*Längste Laufzeit*	*Mittlere Laufzeit*
Positionierung der Knoten	3969 ms	4125 ms	4041 ms
Ermittlung von Nachbarschaftsbeziehungen	2813 ms	3922 ms	3233 ms
Verbinden der Knoten	16 ms	16 ms	16 ms
Der restliche Generierungsprozess	656 ms	702 ms	672 ms

Tabelle 2: ermittelte Laufzeiten von vier gemessenen Abschnitten

Aus der Tabelle kann man entnehmen, dass die Gesamtlaufzeit dieser vier Teilabschnitte mindestens 7454 ms und höchstens 8765 ms benötigt. Im Durchschnitt ist für diese Spezifikation eine Laufzeit von 7962 ms zu erwarten. Die noch hinzuzufügende Laufzeit des vierten Teilabschnitts bestimmt, aufgrund seiner stark schwankenden Laufzeiten maßgeblich die Gesamtdauer der Generierung. Wenn die Verlegung der Kantenzüge unter 2000 ms durchgeführt wird, ergibt sich für die komplette Generierung eine Gesamtdauer von ungefähr 8 bis 11 Sekunden. Dies geschieht in 63 % der Testläufe. In 75 % der Testläufe ist die Verlegung der Kantenzüge nach spätestens 10 Sekunden beendet. Daher ist die Generierung nach spätestens 19 Sekunden abgeschlossen. In den verbleibenden 25 % der Tests dominiert die Laufzeit bei der Verlegung der Kantenzüge die Gesamtlaufzeit des Generierungsprozesses. Eine exakte Abschätzung der Gesamtlaufzeit ist in diesen Fällen nicht möglich.

Eine Lösungsmöglichkeit für dieses Problem wäre ein implementierter Mechanismus, der nach einem spezifizierten Zeitfenster die aktuelle Generierung abbricht und eine erneute Generierung beginnt. Wird dieses Testszenario ohne die Verlegung der Kantenzüge durchgeführt, sind stabile Laufzeiten von wenigen Sekunden zu erwarten.

6 Zusammenfassung und Ausblick

Im Rahmen dieser Diplomarbeit wurde eine Applikation entwickelt, die in der Lage ist, Daten über Eisenbahnnetze, charakterisiert durch Spezifikationen, zu generieren. In der Implementierung greifen Mechanismen der Constraint-Programmierung in die Verarbeitung ein. Die Testergebnisse zeigen, dass die Generierung in Teilen der Implementierung durch Constraints effizient durchgeführt wird. Besonders in den Programmteilen, in denen globale Constraints aus dem Programmiersystem genutzt werden konnten, kann auf die spezifischen Propagationsmechanismen dieser Constraints zurück gegriffen werden. Zudem wurden Constraints verwendet, um die Sicherheit und Qualität der Applikation auf einem hohen Niveau zu halten. Aufgrund der Möglichkeit, den Inhalt der Variablen durch den Constraint-Löser überwachen zu lassen, werden inkonsistente Programmzustände durch diesen Automatismus verweigert. In der klassischen Programmierung müssen zu diesem Zweck explizit Methoden implementiert werden. Die Parametrisierung des Erzeugungsprozesses durch die Spezifikation ermöglicht zudem eine flexible Anwendung dieser Applikation. Die Nutzung eines abstrakten Modells als Basis für die Software erlaubt genügend Flexibilität, um weitere Implementierungen in die Software einzubetten. Beispielsweise können zu den bestehenden einzelne Constraints zur Formulierung von weiteren Randbedingungen hinzugefügt werden. Bestehende Constraints können auch wieder entfernt werden, ohne die Struktur des Programms zu verändern. Die Komplexität der Parametrisierung stellt in der Hinsicht ein Problem dar, dass es möglich ist, Charakteristiken formulieren zu können, die widersprüchlich zueinander sind. Diese Fälle führen zu keinem Ergebnis. In solchen Situationen kann Rechenaufwand durch eine vorzeitige Erkennung der Widersprüchlichkeit gespart werden. Einige dieser Fälle können aber wiederum nur mit großem Rechenaufwand erkannt werden. Die Einhaltung gültiger Parameter wird bereits von der Applikation überprüft. Trotzdem besteht hier die Möglichkeit, verbesserte Algorithmen zur Konsistenzprüfung der Spezifikation zu entwickeln.

Um die Kompatibilität der erzeugten Eisenbahnnetze für weiterverarbeitende Programme zu steigern, wäre das Exportieren der Daten ins RailML-Format ein sinnvoller Ansatzpunkt zur Erweiterung der vorliegenden Arbeit. Weil RailML als Standardformat, für den Zweck des Datenaustauschs zwischen Programmen im Schienenwesen, entwickelt wird, ist dieses Dateiformat dafür gut geeignet.

Eine sinnvolle Erweiterung der vorliegenden Software, wäre die Implementierung von Daten für die

Signalisierung. Die Festlegung der Positionen und Abstände der Vor- und Hauptsignale könnte aufgrund von Geschwindigkeitsangaben sowie modellierten Betriebsweisen an den Strecken und in den Bahnhöfen erfolgen.

Um noch realistischere Eisenbahnnetze zu generieren, könnten Abzweigstellen in weiteren Implementierungen berücksichtigt werden. Beispielsweise könnten zwei in einem Knotenbahnhof einmündende Strecken über eine bestimmte Länge die selben Streckengleise verwenden. Dies führt in der Realität zwar zu Umwegen, ist aber aufgrund der Einsparung von Infrastruktur wirtschaftlicher. Analog zu der in dieser Arbeit formulierten Berechnung von Nachbarschaften zwischen Knoten, könnten Algorithmen basierend auf topologischen Informationen des bestehenden Netzes implementiert werden.

Aus der Sicht des Anwenders, würde eine grafische Benutzeroberfläche den Komfort für die Verwendung dieser Software steigern. Zum Einen ist eine grafische Präsentation aufgrund der Modellierung durch einen Graphen sinnvoll und zum Anderen können erzeugte Teillösungen durch den Anwender bequem visuell analysiert, eventuell verworfen und daraufhin die Generierung erneut angestoßen werden.

Weiterhin können zusätzliche komplexe Modelle weitere Ansatzpunkte zur Spezifizierung von Infrastrukturdaten liefern. Beispielsweise könnten Eisenbahnnetze durch die Angabe von in dem Eisenbahnnetz geplanten Betriebsprogrammen, konkreter charakterisiert werden. Durch die Angabe von Betriebsprogrammen, für welche die Eisenbahninfrastruktur ausgelegt werden soll, könnten beispielsweise die prognostizierten Zugfahrten ein weiteres Kriterium für die Dimensionierung der Gleise einer Strecke sein.

Zusätzliche Details in der Infrastruktur würden eine präzisere Beschreibung der gewünschten Leistungsfähigkeit des Eisenbahnnetzes ermöglichen. Beispielsweise würde die Angabe von Gleisbögen und Langsamfahrstellen sich direkt auf die Fahrgeschwindigkeit der Züge auswirken. Des Weiteren könnte die Spezifizierung von Überwerfungen Fahrstraßenausschlüsse verhindern und parallele Zugfahrten im Eisenbahnnetz ermöglichen.

Abbildungsverzeichnis

Literaturverzeichnis

[Apt03] Apt, Krzysztof R.: *Principles of Constraint Programming.* Cambridge University Press, 2003.

[Bar07] Barták, Roman: *On-Line Guide to Constraint Programming*. http://kti.mff.cuni.cz/~bartak/constraints/, Stand: 08.05.2007.

[Co03a] Cosytec: *CHIP++ Reference Manual.* Cosytec SA, März 2004.

[Co03b] Cosytec: *CHIP Finite domain constraints Reference Manual.* Cosytec SA, Oktober 2005.

[Co03c] Cosytec: *CHIP User's Guide.* Cosytec SA, Oktober 2005.

[DB06a] DB Netz AG: *Trassenpreissystem der DB Netz AG.* http://www.db.de/site/shared/de/dateianhaenge/infomaterial/sonstige/db__netz__trassenpreisbroschuere.pdf, Mai 2006, Stand: 08.05.2007.

[DB06b] DB Station+Service AG: *Liste der Bahnhofskategorisierung 2006.* http://www.db.de/site/shared/de/dateianhaenge/infomaterial/sonstige/liste__bahnhofskategorien.pdf, Stand 05.07.2006

[DB98] Deutsche Bahn AG: *LST-Anlagen planen – Richtlinie 819*. eingeführt am 31.08.1998.

[EBO] *Eisenbahn-Bau- und Betriebsordnung (EBO)*, 8. Mai 1967, zuletzt geändert durch Artikel 499 am 31.Oktober 2006, (BGB1 I S.2407; S.2470).

[Fen03] Fenner, Wolfgang; Naumann, Peter; Trinckauf, Jochen: *Bahnsicherungstechnik.* Publicis Corporate Publishing, 2.Auflage, 2003

[Fie05] Fiedler, Joachim: *Bahnwesen: Planung, Bau und Betrieb von Eisenbahnen, S-, U-, Stadt- und Straßenbahnen.* Werner Verlag, 5. Auflage, 2005.

[Fri03] Fries, Nikolaus: *Modellierung einer Eisenbahninfrastruktur in railML®.* Studienarbeit,TU Dresden, 2003

[Ges03] Geske, Ulrich: *Vorlesungsfolien der Vorlesung Constraint-Programmierung an der Universität Potsdam*, 2003.

[Lind04] Linder, Ulrich; Krauß, V.P.; Fries, Nikolaus; Hoffmann Raik: *Gesamtkonzept und Grobstruktur.* Dokumentation zum RailML Schema Infrastruktur, Fraunhofer Institut für Verkehrs- und Infrastruktursysteme, Dresden, 2004

[Pa04] Pachl, Jörn: *Systemtechnik des Schienenverkehrs.* B. G. Teubner Stuttgart, Leipzig, Wiesbaden, 4. Auflage, 2004.

[Pa07] Pachl, Jörn: *Glossar der Systemtechnik des Schienenverkehrs.* http://joernpachl.gmxhome.de/glossar.htm, Stand: 02.04.2007

[Pa98] Pachl, Jörn: *Die verschränkte Dreigleisigkeit.* Tetzlaff Verlag, EI – Eisenbahningenieur (49) 3/1998, S. 27-29

[Pa99] Pachl, Jörn: *Standardisierung der Infrastruktur kleiner Knoten.* Tetzlaff Verlag, EI – Eisenbahningenieur (50) 4/1999, S. 14-20

[Rail04a] RailML.org: *infrastructure reference.* http://railml.org/genesis/infrastructure/v100/Reference/HTML/Infrastructure_V100_Reference_EN.html, 01.12.2004, Stand: 12.05.2007.

[Rail04b] RailML.org: *Infrastruktur Schemendefinition.* http://www.railml.org/schemes/infrastructure/infrastructure_V100.xsd, 01.12.2004, Stand: 12.05.2007.

[Rail04c] RailML.org: *Infrastruktur Beispiel.* http://www.railml.org/schemes/infrastructure/infrastructure_V100.xml, http://www.railml.org/schemes/infrastructure/demonet_V100.pdf, 01.12.2004, Stand: 12.05.2007.

[VCD04] VCD Verkehrsclub Deutschland e.V.: *Bahn21.* veröffentlichte Studie, 2004.

[Wei02] Weigand, Werner; Marx, Jürgen; Bendfeldt J.P.: *Planungen zur Weiterentwicklung der Netzknoten bei der DB Netz AG.* Tetzlaff Verlag, EI – Eisenbahningenieur (53) 7/2002, S. 5-12

[Wi07] *Wikipedia. Die freie Enzyklopädie*. *Sachsen-Anhalt.* http://de.wikipedia.org/wiki/Sachsen_Anhalt, Stand: 14.05.2007.

[Wolf07] Hofstedt, Petra; Wolf, Armin: *Einführung in die Constraint-Programmierung.* Springer, Berlin, 2007.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die beiliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Die Diplomarbeit hat keiner anderen Prüfungsbehörde vorgelegen.

Potsdam, den 15. Mai 2007 .....

(Unterschrift)