Constraint-basierte Fahr- und Kapazitätsplanung für ein Straßenbahnnetz

U. Geske[a], A. Wolf[b], R. Zander[a]

[a] Universität Potsdam, August-Bebel-Str. 89, 14482 Potsdam
{ugeske, raphzan}@uni-potsdam.de

[b] Fraunhofer FIRST, Kekuléstr. 7, 12489 Berlin
Armin.Wolf@first.fraunhofer.de

Abstract:

In dieser Arbeit wird ein Fahrplanungssystem vorgestellt, dass die zu berücksichtigten Bedingungen mit Hilfe so genannter Constraints formalisiert und durch entsprechende Verfahren effizient verarbeitet. Dies erlaubt es, konfliktfreie Fahrpläne innerhalb kurzer Zeit zu erstellen und Fall- und Parameterstudien („Was-wäre-wenn"-Szenarien) mit moderatem Zeitaufwand durchzuführen. So wurde das System zur Kapazitätsanalyse für eine geplante Baumaßnahme eingesetzt. Dabei konnte die Frage beantwortet werden, ob ein Fahrplan trotz eines aktuellen Engpasses einzuhalten ist.

1 Einleitung

Bei der Erzeugung eines Fahrplans für Bahnnetze kann von einem Wunschfahrplan ausgegangen werden, der auf Erfahrung basiert, aber in aller Regel konfliktbehaftet ist. Wird daraus ein gültiger (konfliktfreier) Fahrplan generiert, gibt es Bedingungen unterschiedlicher Priorität zu beachten, die entweder unbedingt einzuhalten sind oder möglichst eingehalten werden sollen. Aus Sicherheitsgründen muss im Allgemeinen darauf geachtet werden, dass ein belegter Block für andere Züge gesperrt ist. Im Eisenbahnverkehr wird das Problem durch Signale gelöst und muss deshalb zwingend bei der Fahrplanerstellung berücksichtigt werden [5], im Straßenbahnverkehr ist dies nur bedingt erforderlich, da die Fahrer dort bis zu einer Geschwindigkeit von 70 km/h generell auf Sicht fahren. Dennoch kann die erprobte Methodik auch bei der Fahrplangenerierung für Straßenbahnnetze angewendet werden. Eine weitere einzuhaltende Bedingung soll sein, dass die Züge einer Linie innerhalb festgelegter Fahrtperioden grundsätzlich in einem bestimmten Takt (z.B. alle 20 Minuten) fahren. Ein Aspekt der Analyse eines generierten Fahrplans kann darin bestehen, ob er einzuhalten ist, wenn sich angenommene Bedingungen ändern, z. B. dadurch dass (veränderte) Ampelschaltungen nachträglich ein Hindernis darstellen. Im Folgenden wird das Softwaresystem TITAN zur Planung von getakteten Straßenbahn-Linienfahrten vorgestellt, das zwingend einzuhaltende Randbedingungen zur Zugsicherung behandelt. Sowohl für die Fahrplanung im Eisenbahnwesen als auch für Straßenbahn- und Busnetze sind eine Vielzahl von Systemen entwickelt worden und in der Anwendung. Der vorliegende Programmier-Ansatz basiert auf einem bisher für diesen Zweck kaum verwendeten IT-Verfahren, das den entscheidenden Vorteil besitzt, vorausschauend

Konflikte in der Planung zu vermeiden und dadurch im Allgemeinen effizient eine widerspruchsfreie Lösung, sofern eine vorhanden ist, erzeugen kann. Die auch als „Constraint-Löser“ bezeichneten Interpreter/Compiler haben sich bei der Fahrplan- und Betriebssimulation für Eisenbahnnetze bewährt [3] und werden in der vorliegenden Arbeit für die Generierung von Fahrplänen und für Kapazitätsuntersuchungen für ein Straßenbahnnetz angewendet. Dabei wird der in [8] beschriebene Constraint-Löser verwendet, der in JAVA realisiert ist und dadurch optimale Voraussetzungen für die Einbettung in Software-Umgebungen bietet. Aus technischer Sicht wird für die Straßenbahn-Fahrplanung auf den Grundlagen der entsprechenden Planung im Eisenbahnwesen, wie sie z. B. in [5] beschrieben sind, aufgebaut. Nach der Beschreibung der Voraussetzungen für die Planung (Datenformate in RailML, verwendetes Berechnungsverfahren), die in den ersten beiden Abschnitten erfolgen, werden in den folgenden beiden Abschnitten die Methode zur Generierung von Fahrplänen sowie die Erkenntnisse aus Simulationsläufen beschrieben.

2 Beschreibung der Eingabedaten

Für die Berechnung und Simulation der Fahrpläne werden Daten für die Infrastruktur und die Linienfahrten benötigt. Aufgrund seiner Herstellerunabhängigkeit, der freien Verfügbarkeit, seiner Quasi-Standardisierung und einfachen informationstechnischen Verarbeitbarkeit wird RailML [4], [6], eine XML-basierte Sprache, als Datenformat verwendet. Während das Format für die Infrastrukturbeschreibung bereits Bestandteil des RailML-Standards ist, ist das Format für die Linienfahrten [2], [7], [9], [10] noch in Diskussion. Eine Linie enthält Informationen über die einzelnen Abschnitte (Linienabschnitte)), in die sie unterteilt ist und Daten über die einzelnen Fahrtperioden (`timing`). Ein Linienabschnitt (`section`) wird definiert durch einen Identifikator, eine Referenz auf das Gleis, welches die Züge in dem Linienabschnitt verwenden, eine Referenz auf den ersten verwendeten Abschnitt des Gleises und die Angabe der durchschnittlichen Durchfahrtszeit der Züge in Sekunden (`trackAverageTime`). Verweist der Linienabschnitt auf eine Haltestelle, so werden zusätzlich noch die minimale und die maximale Haltezeit (`trackStationMinWaitTime` bzw. `trackStationMaxWaitTime`) angegeben. Außerdem kann bei Haltestellen eine zusätzliche Behinderungszeit zur Simulation von Verzögerungen über den Parameter `trackHandicapMax-WaitTime` festgelegt werden. Das Element `endSection` des Elements `section` bestimmt den letzten verwendeten Abschnitt des Gleises und besitzt einen Verweis auf den nächsten Linienabschnitt. Zusätzlich kann festgelegt werden, für welche Wochentage die Informationen gelten. Das Element `timing` enthält Informationen über die Fahrtenperioden einer Linie. Es können beliebig viele Fahrtperioden definiert werden und somit die Daten einer Fahrtperiode in Abhängigkeit von der Tageszeit angepasst werden. Eine Fahrtperiode `period` definiert den Start- und Endbahnhof der Fahrten, außerdem in welchem Zeitintervall (`startMinTime`, `startMaxTime`, `endMinTime`, `endMaxTime`) und an welchen Wochentagen (days) die Züge verkehren sollen. Der erste Zug der Fahrtperiode startet frühestens zum Zeitpunkt `startMinTime`, spätestens zum Zeitpunkt `startMaxTime`, der letzte zwischen `endMinTime` und `endMaxTime`. Der Parameter `timing` gibt das

Taktintervall der Fahrtperiode in Sekunden an. Wenn z.B. `timing` den Wert *1200* hat, fahren die Züge der Linie innerhalb dieser Fahrtperiode im 20-Minuten-Takt. Das Element `trackCutOptions` enthält die Attribute `minTrackCutLength` und `defaultTrackCutLength`, welche definieren, wie lang ein Gleisblock mindestens bzw. höchstens sein darf. Das Element `timeCalculationOptions` enthält die nachfolgend aufgeführte Attribute. Die Attribute `predeposition` und `postdeposition` definieren die Anfahrbeschleunigung bzw. Bremsverzögerung eines Zuges in m/s^2 zur Bestimmung von Vor- und Nachbelegungszeiten eines Gleisblocks. Das Attribut `supposed-StationWaitTime` bestimmt die Dauer der Haltezeit an Haltestellen. Ist der Wert auf `minWaitTime` gesetzt, werden die minimalen Haltezeiten verwendet, bei `maxWaitTime` die maximalen, bei avgWaitTime der Durchschnitt aus beiden und bei `varWaitTime` der flexible Haltezeitraum [`minWaitTime`, `maxWaitTime`]. Mit dem Attribut `supposedHandicapWaitTime` kann eine kann eine zusätzliche Behinderungszeit an Haltestellen zur Simulation von Hindernissen definiert werden. Dies kann zum Zwecke der Simulation und bei einer Kapazitätsanalyse eingesetzt werden, findet aber bei der reinen Fahrplanerstellung keine Verwendung. Das Attribut `defaultTrainLength` bestimmt die Länge eines Zugs in Kilometern. Das Attribut `bufferTime` definiert eine Mindestpufferzeit, die zwischen zwei aufeinander folgenden Blockbelegungen an Haltestellen eingehalten werden muss.
Wenn das optionale Attribut `startTimeInterval` definiert ist, werden die spätesten Startzeiten `startMaxTime` der ersten Züge aller Fahrtperioden mit diesem Wert überschrieben. Dies kann zu Simulationszwecken nützlich sein. Ist der Wert des optionalen Attributs `useSpeedChange` mit dem Wert "*yes*" belegt, werden alle auf Gleisen definierten Geschwindigkeitseinschränkungen berücksichtigt. Das optionale Attribut `speed` wird als globale Durchschnittsgeschwindigkeit interpretiert, sofern es angegeben ist. In diesem Fall wird mittels seines Werts die Dauer der Durchfahrtszeiten aller Gleisblöcke neu berechnet und die standardmäßigen Werte `trackAverageTime` überschrieben. Geschwindigkeitseinschränkungen und Durchschnittsgeschwindigkeit finden nur in der Simulation Verwendung.
Mit dem Element `timetableOptions` kann der Planungshorizont, also der zeitliche Bereich, in dem die Zugfahrten berechnet werden sollen, eingeschränkt werden. Außerdem kann mittels des Attributs `rounding-Method` angegeben werden, ob die geplanten, sekundengenauen Abfahrts- und Ankunftszeiten bei Ausgabe der öffentlich ausgeschriebenen, minutengenauen Daten grundsätzlich abgerundet (down) oder arithmetisch (`arithmetic`) gerundet werden. Beim Abrunden wird die öffentliche Abfahrts- bzw. Ankunftszeit als die frühste mögliche interpretiert, beim arithmetischen Runden darf die Abweichung maximal eine halbe Minute betragen.
Die lösungsspezifischen Elemente schedulingOptions ermöglichen das Teilen des Planungshorizonts in mehrere Intervalle, indem mit dem Attribut `timeSection` die Intervallgröße in Sekunden angegeben wird. Außerdem kann mit dem Attribut `flexibleTimeblock` festlegt werden, ob der Taktblock der Linienfahrten bei den Abfahrtszeiten an den Haltestellen exakt eingehalten werden muss (Einstellung: *`flexibleTimeblock = no`*) oder ob ein gewisser Spielraum zulässig ist. Zusätzlich `solver`-Element für Angaben zur Art und Weise der Fahrplanberechnung vorhanden.

3 Verfahren zur Sicherung der Konsistenz der Fahrpläne

Fahrpläne müssen konsistent, d.h. widerspruchsfrei sein. Weil eine große Zahl von Regeln und Bedingungen wie zur Betriebssicherheit, zu Fahr- und Haltezeiten, zur Kundenfreundlichkeit oder zu Anschlusszeiten eingehalten werden müssen, ist die Konsistenzsicherung das Hauptproblem, insbesondere wenn weitere Aspekte, wie z. B. Umlaufplanung und Personalplanung mit eventuell gewünschten Optimierungen, eine Rolle spielen. Wir verwenden deshalb das Paradigma der Constraint-basierten Systeme. Es unterscheidet sich in der Art und Weise der Verarbeitung bzw. der Berücksichtigung der Bedingungen (Constraints) wesentlich von anderen Ansätzen. Probleme werden aber, wie in anderen Paradigmen im Allgemeinen auch, durch eine Menge von Variablen sowie durch Funktionen und Relationen – insbesondere arithmetischen – zwischen den Variablen repräsentiert. Derartige Relationen beschreiben die formalisierten Planungsbedingungen (sogenannte Constraints), die an eine zulässige Variablenbelegung gestellt werden. In klassischen Berechnungsansätzen können diese Bedingungen erst geprüft werden, wenn allen Variablen V_p der Problembeschreibung Werte (z. B. Abfahrtszeiten) zugewiesen sind, woraus das ineffektive Trial-and-Error-Verhalten resultiert. In der Constraint-Programmierung wird dagegen das Behandeln der Constraints (wie z.B.: *4:55 < Fahrtbeginn_Linie1 < 5:05*) vor die Generierung von Variablenbelegungen gezogen. Die spezifizierten Constraints mit zum Teil gemeinsamen Variablen bilden ein Constraint-Netz mit den Variablen als Knoten und den relationalen Beziehungen als Kanten. Wenn der Wertebereich einer Variablen V_i verändert wird, werden durch den als Dämon arbeitenden Constraint-Löser sofort die Konsequenzen für die anderen Variablen $\{V_k,\ldots\}$, die mit dieser Variablen über Constraints $\{C_{ik},\ldots\}$ verbunden sind, berechnet. Bei diesem Vorgang, der Constraint-Propagation, werden aus den Wertebereichen der Variablen V_i und $\{V_k,\ldots\}$ Werte gelöscht, die die Constraints $\{C_{ik},\ldots\}$ nicht erfüllen können. Dadurch werden der Suchraum und folglich der Berechnungsaufwand verkleinert, ohne dass Lösungen verloren gehen. Algorithmen zur vollständigen Constraint-Propagation, d. h. zum Löschen *aller* Werte, die garantiert nicht zur Lösung beitragen, sind für praktische Anwendungen zu ineffizient. Aus diesem Grund werden unvollständige Verfahren, bei denen eventuell überflüssige Werte in den Wertebereichen verbleiben, verwendet. Für die Elimination dieser Werte ist ein zusätzliches Suchverfahren erforderlich. Constraint-Propagation erfolgt auch innerhalb dieses Suchverfahrens bei der Veränderung des Wertebereichs einer Variablen V_i durch Wertzuweisung (im klassischen Sinn) zu dieser Variablen. Wird bei der Constraint-Propagation der Wertebereich einer Variablen leer - die Variable kann also keinen Wert mehr annehmen - liegt eine Sackgasse vor, die durch Rücksetzen (Backtracking) zu einem alternativen Wert für diese Variable V_i umgangen werden kann. Der entscheidende Vorteil liegt in der Erkennung dieser Situation bevor allen Variablen der Problembeschreibung V_P Werte zugewiesen wurden. Dadurch resultiert eine sehr effiziente Abarbeitung, insbesondere bei einer großen Anzahl zu berücksichtigender Constraints, wie es in der Fahrplanung der Fall ist. Neben einfachen arithmetischen Constraints können auch sogenannte „globale Constraints“ wie z.B. das SingleRessource- oder das Summen-Constraint, verwendet werden. Das SingleRessource-Constraint sichert, dass alle für die auf einer Ressource (Maschine, Gleisblock, …) ausführbaren Aufgaben (Tasks, Zugbelegungen, …) zu unterschied-

lichen Zeiten und damit überlappungsfrei eingeplant werden. Das Summen-Constraint realisiert die Korrektheit der Summenbildung $a_1+...+a_n=b$, auch wenn $a_1,...,a_n$ oder b im Laufe der Programmabarbeitung noch keinen konkreten Wert besitzen.

4. Generierung von Fahrplänen

Das Ziel der Untersuchungen ist die Erzeugung von widerspruchsfreien Fahrplänen für ein Straßenbahnnetz. Für das Streckennetz werden Eingabedaten verwendet, die im standardisierten RailML-Infrastruktur-Format vorliegen, für die zugehörigen Linieninformationen Daten im LineInfo-Format [2], [7], welches in Anlehnung an den RailML-Standard entwickelt und von Zander [9], [10] erweitert wurde. Die Ausgabe des Fahrplans erfolgt als graphischer Gleisblockbelegungsplan oder als XML-Datei, welche das RailML-Timetable-Format erfüllt. Daten im Timetable-Format können auch als Eingabeparameter verwendet werden. Bereits in der Problembeschreibung der Fahrplanung wird dem Aspekt der Betriebssicherheit höchste Priorität zuteil. Dazu wurde, in Anlehnung an das Eisenbahnwesen, ein Algorithmus entwickelt, der die Gleise des Streckennetzes in virtuelle Gleisblöcke zerlegt. Ein Gleisblock kann zu jedem Zeitpunkt nur von maximal einem Fahrzeug befahren werden. Ein zusätzlicher Sicherheitsabstand kann durch Vor- und Nachbelegungszeiten realisiert werden, obwohl das im Straßenbahnverkehr nur bedingt erforderlich ist, da hier bis zu *70 km/h* auf Sicht gefahren wird. Dazu kann diese Blockung weitestgehend aufgehoben werden. Die Züge einer Linie fahren innerhalb festgelegter Fahrtperioden grundsätzlich in einem bestimmten Takt (z.B. alle 20 Minuten), der eingehalten werden sollte. Es kann ein gewisser Spielraum für die exakte Einhaltung des Takts eingeräumt werden. Das lässt sich damit rechtfertigen, dass der öffentliche Taktfahrplan nur minutengenau ausgegeben wird, während der betriebsinterne Fahrplan die Daten sekundengenau speichert. Ausgehend von einer ausgeschriebenen Abfahrtszeit a eines Zugs wäre dann ein Spielraum von $a \pm 30$ Sekunden (bzw. *[a, a + 59 Sekunden]*, falls der Abfahrtstermin des öffentlichen Fahrplans als der frühste mögliche interpretiert wird) für die betriebsinterne Abfahrtszeit akzeptabel. Die Verwendung eines derartigen Intervalls für die Abfahrtszeit macht die Fahrplanerzeugung flexibler und kann die Wahrscheinlichkeit, einen Fahrplan gemäß der Forderungen zu finden, erhöhen, allerdings wird dadurch auch der Suchraum vergrößert, was meist eine längere Laufzeit der Fahrplanberechnung bedingt.

4.1 Generierung widerspruchsfreier Fahrpläne

Die Fahrplangenerierung kann in die sequentiell auszuführende Schritte unterteilt werden:

1. Einlesen der XML-Daten im RailML-Format zur Infrastruktur und zu den Linien
2. (Virtuelle) Zerlegung der Gleise in Gleisblöcke
3. Bestimmung der Blockbelegungen
4. Spezifikation der Taktblöcke
5. Berechnen der Vor- und Nachbelegungszeiten für die Gleisblöcke
6. Berücksichtigung der Übergangszeiten

7. Überschneidungsfreiheit der Blockbelegungen (durch SingleResource-Constraints)
8. Lösungssuche zur Bestimmung eines zulässigen Fahrplans
9. Graphische Darstellung des erstellten Fahrplans
10. Speichern des generierten Fahrplans

Nach dem Erfassen der Daten werden alle längeren Gleise der Infrastruktur in Gleisblöcke zerlegt. Die folgenden Schritte (3–8 der obigen Übersicht) werden für jedes Planungsintervall durchgeführt. Davon werden die Schritte 3–6 sequentiell für die Fahrten der Fahrtperioden aller Routen, die in dem Planungsintervall stattfinden sollen, durchgeführt. Für diese Fahrten werden alle genutzten Gleisblöcke ermittelt und für diese die formelmäßigen Belegungsdaten festgelegt (Schritt 3). Über die Belegungen der Gleisblöcke werden Summen-Constraints definiert, welche gewährleisten, dass die Belegungsdauer der Gleisblöcke, sowie der Taktblock zwischen zwei aufeinander folgenden Fahrten einer Linie, eingehalten werden (Schritt 4). Anschließend werden die Vor- und Nachbelegungszeiten für die Blockbelegungen berechnet (Schritt 5), danach werden die Constraints zur Einhaltung der Umsteigezeiten (sofern definiert) abgesetzt (Schritt 6). Anschließend wird für alle Gleisblöcke je ein SingleResource-Constraint definiert, welches gewährleistet, dass sich die Blockbelegungen nicht überschneiden (Schritt 7). Anschließend wird eine Lösung für alle Belegungen unter Berücksichtigung der definierten Constraints gesucht (Schritt 8). Die gefundenen Werte der Belegungen der Gleisblöcke definieren einen zulässigen Fahrplan. Dieser Fahrplan wird analysiert und graphisch ausgegeben (Schritt 9) und im RailML-Format abgespeichert (Schritt 10).

(Virtuelle) Zerlegung der Gleise in Gleisblöcke

Wie bereits im Abschnitt 2.2 beschrieben, wird die maximale Blocklänge über einen globalen Parameter der Fahrplanberechnung (Attribut `defaultTrackCutLength` des Elements `solver`) festgelegt. Gleise, die größer als dieser Wert sind, werden in kleinere Blöcke unterteilt. Wenn ein Gleis bzw. ein Gleisrest kleiner als *1,5 x* *`defaultTrackCutLength`* ist, wird er in zwei Abschnitte der Größe Gleisrest/2 aufgeteilt. Je kleiner die maximale Gleisblocklänge ist, desto höher kann die Zugdichte sein, da immer nur kleine Abschnitte für einen kurzen Zeitraum bei der Gleisblockbelegung gesperrt werden müssen. Der Nachteil einer kleinen maximalen Gleisblocklänge ist, dass mehr Speicher und Zeit benötigt wird, da alle verwendeten Constraints auf die einzelnen Gleisblöcke angewendet werden.

Bestimmung der Blockbelegungen

Für die Belegung der Gleisblöcke werden der Belegungsbeginn, die Belegungsdauer und das Belegungsende benötigt. Der früheste Startpunkt des Belegungsbeginns für den ersten Gleisblock der Linienfahrt *i* der aktuellen Fahrtperiode ist die Summe aus der frühesten Abfahrtszeit des ersten Zugs (`startMinTime` des Elements `period`) und dem Produkt aus der Nummer *i* und der Taktblockgröße (`timing` des Elements `period`) der Linienfahrt. Der späteste Startpunkt des Belegungsbeginns (`startMaxTime` des Elements `period`) ist die Summe aus der spätesten Abfahrtszeit des ersten Zugs und dem Produkt aus *i* und *`timing`*. Der Belegungsbeginn aller anderen

Gleisblöcke ist gleich dem Belegungsende `previousReservedTo` des vorherigen durchfahrenen Gleisblocks. Die minimale bzw. maximale Belegungsdauer setzt sich aus der Durchfahrtszeit und optionalen Haltezeiten zusammen. Für Haltestellen ist die Dauer der Haltezeiten in dem Attribut `trackStationMinWaitTime` bzw. `trackStationMaxWaitTime` des Elements `section` festgelegt. Welcher Wert verwendet wird, hängt von der Einstellung `supposedStationWaitTime` des `solver`–Elements ab. Ist der Gleisblock keine Haltestelle, so ist die *Haltezeit = 0*. Das Belegungsende setzt sich aus dem Belegungsbeginn, der Summe der minimalen bzw. maximalen Belegungszeiten der vom Zug der Linienfahrt schon durchfahrenen Gleisblöcke zusammen.

Spezifikation der Taktblöcke

Für Haltestellen liegen nur die Durchfahrts- und Haltezeiten als Eingabedaten vor, jedoch nicht die Abfahrts- und Ankunftszeiten. Da aber die Differenz zwischen dem Belegungsbeginn und der Abfahrtszeit der Fahrt *i* aber für alle Fahrten *i* ($1 \leq i \leq$ *Anzahl Linienfahrten der Fahrtperiode)* gleich ist, reicht es aus, die Einhaltung des Taktblocks bezüglich des Belegungsbeginn BB_0 der Fahrten zu definieren. Es gilt: BB_0 + timing * i = BB_i. Dabei sind BB_i der Belegungsbeginn der *i*-ten Linienfahrt des aktuellen Linienabschnitts und `timing` der Taktblock der Fahrtperiode. Wird der Taktblock aufgeweicht (Parameter: *`flexibleTimeblock = "yes"`*), so kann die maximale Verschiebung der Abfahrtszeit gemäß des Takts durch das Intervall *`[timeBefore, timeAfter]`* ausgedrückt werden. Die Werte `timeBefore` und `timeAfter` definieren die maximal mögliche Verschiebung in Sekunden gegenüber der Angabe in Minuten. Das „Abrundungsverfahren" ist dadurch definiert, dass `timeBefore` auf den Wert *0* und `timeAfter` auf den Wert *59* gesetzt werden. Standardmäßig erhält `timeBefore` den Wert *30*, `timeAfter` den Wert *29*.

Berechnen der Vor- und Nachbelegungszeiten für die Gleisblöcke

Für die Berechnung der Vor- und Nachbelegungslänge werden die Parameter `predeposition` und `postdeposition` verwendet, welche die Anfahrbeschleunigung und die Bremsverzögerung eines Zugs in m/s^2 definieren. Die Länge der Vorbelegungsstrecke ist gleich *0*, wenn der Parameter `predeposition` den Wert *0* besitzt, ansonsten wird sie nach der Formel $l(VB) = v^2/(2+a)$ berechnet, wobei *v* die Durchschnittsgeschwindigkeit ist, mit welcher der Zug den Gleisblock durchfährt und *a* die Anfahrbeschleunigung ($a > 0$) ist. Die Länge der Nachbelegungsstrecke kann analog berechnet werden. Für jeden Gleisblock wird die Länge der Vor- und Nachbelegungsstrecke bei der Belegung in den Variablen `preReservationLength` und `postReservationLength` gespeichert. Da die Vor- und Nachbelegungszeiten erst noch berechnet werden müssen, wird der Wert `durWithSafeDistance` für die Belegungsdauer inkl. Vor- und Nachbelegung vorerst auf den Wert der Belegungsdauer `dur` gesetzt und später angepasst.

Berücksichtigung der Übergangszeiten

Übergangszeiten werden an Haltestellen für zwei verschiedene Linien definiert. Die Mindestübergangszeit wird im Attribut `minDiffTime`, die Höchstübergangszeit *(Mindestübergangszeit+Übergangspufferzeit)* im Attribut `maxDiffTime` gespeichert. Voraussetzung für die Definition einer Übergangszeit ist, dass der Takt der ersten Linie gleich dem oder ein Vielfaches der zweiten Linie ist. Grundsätzlich reicht es aus, die Übergangszeit ($\ddot{U}$) zwischen 2 Fahrten (i und j) der Fahrtperioden (pl_1 und pl_2) über zwei unterschiedliche Linien (l_1 und l_2) zu definieren, die Einhaltung der Übergangszeit für die anderen Fahrten wird durch den Taktblock gewährleistet. Die Übergangszeit definiert die Differenz zwischen der Ankunftszeit der Fahrt $i(Ank_i)$ und der Abfahrtszeit der Fahrt $j(Abf_j)$. Es gilt: $Ank_i(pl_1) + \ddot{U} = Abf_j(pl_2)$.

Bei der Definition der Übergangszeit muss die Fahrt i so gewählt werden, dass $Ank_i(pl_1) + \ddot{U}$ innerhalb des zulässigen Intervalls für die Abfahrtszeit der Fahrt j liegt. Dazu werden die Intervallgrenzen von Ank_i auf den spätesten Beginn der ersten Fahrt der später startenden Fahrtperiode und das früheste Ende der letzten Fahrt der früher endenden Fahrtperiode gesetzt ($Anki \in I\ddot{U} = [max(startMaxTime(pl_1), startMaxTime(pl_2)), min((endMinTime(pl_1), endMinTime(pl_2))]$) und eine Fahrt i der Linie 1 ermittelt, für die Ank_i in $I\ddot{U}$ liegt. Da Ank_i variabel ist, kann während der Planungsphase die Zuordnung von i zu j nicht genau bestimmt werden. Deshalb werden alle Lösungskandidaten j_x mit $(1 \leq x \leq Anzahl\ der\ Fahrten)$ für i ermittelt, welches alle Fahrten der Fahrtperiode pl_2 sind, für die gilt: $BB_{min}(j_x) < BE_{max}(i) \wedge BE_{max}(j_x) > BB_{min}(i)$ (BB ist dabei der späteste Beginn, BE das früheste Ende).

Überschneidungsfreiheit der Blockbelegungen (durch SingleResource-Constraints)

Für jeden Gleisblock wird ein eigenes SingleResource-Constraint erzeugt, dem als Tasks die Belegungsdaten aller Linienfahrten, welche den Gleisblock belegen, übergeben werden. Dieses Constraint sorgt dafür, dass sich die Belegungen zeitlich nicht überlappen. Für jede variable Belegung (`varReservation`) eines Blocks wird eine Task, die durch Beginn, Dauer und Ende der Belegung definiert ist, erzeugt. Dabei wird für die Blockbelegung über die Variablen `reservedSectionsFrom` und `reservedSectionsTo` der Belegungsbeginn (`from`) des ersten Gleisblocks der Vorbelegung bzw. das Belegungsende (`to`) des letzten Gleisblocks der Nachbelegung ermittelt; die Belegungsdauer `duration` steht bereits unter `durationWithSafeDistance` zur Verfügung. Wenn der aktuelle Gleisblock zu einer Haltestelle gehört und für diesen eine Pufferzeit (`bufferTime`) oder eine globale *Pufferzeit* > *0* hinterlegt ist, wird diese zu der Dauer und dem Ende der Belegung hinzuaddiert, ein Task mit den Werten `from`, *`dur = duration + bufferTime`* und *`to = to + bufferTime`* erzeugt, ansonsten wird ein `Task` mit den Belegungszeiten ohne Pufferzeit (`from`, `duration` und `to`) übergeben. Da das Belegungsende variabel ist, muss bei Verwendung einer Pufferzeit außerdem sichergestellt werden, dass die Differenz zwischen dem Belegungsende `to` und dem Belegungsende inkl. Pufferzeit auch wirklich der Pufferzeit entspricht, was durch ein Summen-Constraint realisiert werden kann. Werden mehrere Planungsintervalle verwendet, so muss außerdem

gewährleistet sein, dass sich die festzulegenden Belegungen nicht mit den bereits feststehenden überschneiden.

Zur Fahrzeitberechnung wird die Belegungsdauer für alle Fahrten ermittelt. Dabei können Wartezeiten und Behinderungszeiten in Haltestellen unterschiedlich berücksichtigt werden. Die Halte- und Behinderungszeiten können variabel definiert werden, so dass bei Konflikten unter Umständen nicht eine komplette Fahrt verschoben werden muss, sondern nur die Haltezeit verändert wird. Außerdem kann die Dauer der Vor- und Nachbelegung durch Angabe der Anfahrbeschleunigung bzw. der Bremsverzögerung verändert werden. Mittels Constraint-basierter Techniken wird eine konfliktfreie Gleisblockbelegung ermittelt, wodurch garantiert wird, dass der resultierende Fahrplan widerspruchsfrei ist. Der Einhaltung der Fahrzeiten und der Taktblöcke können gewisse Spielräume (z. B. *-30/+30* Sekunden) gegeben. Das lässt sich damit rechtfertigen, dass der öffentliche Taktfahrplan nur minutengenau angegeben wird, während der betriebsinterne Fahrplan die Daten sekundengenau speichert. Die Verwendung eines derartigen Intervalls für die Abfahrtszeit macht die Fahrplanerzeugung flexibler und kann die Wahrscheinlichkeit, einen Fahrplan gemäß den Forderungen zu finden, erhöhen.

Lösungssuche zur Bestimmung eines zulässigen Fahrplans

Beim Fahrplanungsprozess werden sequentiell alle Planungsintervalle durchlaufen und die Fahrten, die in dem jeweiligen Intervall starten, berechnet. Bei der Zerlegung des Planungshorizonts in Intervalle wird die Tatsache genutzt, dass die Züge einer Linie in einem festen Taktblock fahren und Probleme deshalb immer zyklisch auftreten. Wenn zu einem bestimmten Zeitpunkt *t* ein Konflikt zwischen den Fahrten zweier Linien auftritt und beide Linien den Taktblock *b* haben, tritt dieser Konflikt auch zum Zeitpunkt *t+b* auf. Das gilt zumindest unter der Voraussetzung, dass der Zeitblock starr ist und keine flexiblen Wartezeiten an Haltestellen verwendet werden, welche die Dauer einzelner Fahrten beeinflussen können.

Obwohl für die Durchfahrtszeiten der Gleisblöcke statistische Werte in den LineInfo-Eingabedaten vorliegen, kann für Simulationszwecke eine Durchschnittsgeschwindigkeit (`speed`) angegeben werden, mit der alle Gleisblöcke durchfahren werden, um die Auswirkung einer Veränderung der Durchschnittsgeschwindigkeit auf den Fahrplan festzustellen. Dies stellt ein probates Mittel zur Bewertung der Robustheit des Fahrplans zur Verfügung. Je geringer die Durchschnittsgeschwindigkeit ist, bei der noch eine Lösung gefunden wird, desto robuster ist der Fahrplan gegenüber Verspätungen, da die Gleisblöcke bei einer höheren Durchschnittsgeschwindigkeit nur für kürzere Zeiträume gesperrt werden müssen. Außerdem können für einzelne Gleise Geschwindigkeitsbegrenzungen angegeben werden, um abschätzen zu können, wie sich z. B. Baustellen auf den Fahrplan auswirken. Um Geschwindigkeitsbegrenzungen zu berücksichtigen, muss der Parameter useSpeedChange auf den Wert "`yes`" gesetzt sein.

Mittels des Rundungsverfahrens (`roundingMethod`) wird definiert, nach welchem Verfahren sekundengenaue Zeiten bei der Umrechnung in minutengenaue Werte gerundet werden. Hat der Parameter den Wert "`down`" (Standard-Einstellung), werden

alle sekundengenauen Zeiten auf die volle Minute abgerundet, sonst (`roundingMethod=arithmetic`) wird bis *29* Sekunden abgerundet, ab *30* Sekunden aufgerundet. Z. B. wird die sekundengenaue Uhrzeit 08:00:30 nach dem Abrundungsverfahren auf 08:00 Uhr abgerundet, nach arithmetischer Rundung auf 08:01 Uhr aufgerundet, während die Uhrzeit 08:00:29 nach beiden Verfahren auf 08:00 Uhr abgerundet wird. Das Rundungsverfahren findet sowohl bei der Ausgabe der öffentlichen Fahrplandaten, als auch bei der Verwendung von flexiblen Taktblöcken Anwendung.

Zur Verbesserung der Fahrplanqualität können Umsteigepunkte definiert werden, welche die Einhaltung von Übergangszeiten zwischen zwei Linien sicherstellen. Zur Reduzierung der durchschnittlichen Wartezeiten der Fahrgäste kann eine globale Pufferzeit als Parameter übergeben werden, die an allen Haltestellen zwischen je zwei aufeinander folgenden Zugfahrten eingehalten werden muss. Daraus resultiert im Allgemeinen eine gleichmäßigere Verteilung der Fahrten. An Haltestellen können Umsteigepunkte zwischen zwei Linien mit einem definierten Übergangszeitintervall festgelegt werden, dessen Einhaltung durch Constraints gewährleistet wird.

5 Simulation von Fahrplänen

Grundlage für die Benchmark-Tests und Simulationen sind reale Daten für ein Straßenbahnnetz. Das Netz besitzt 7 Straßenbahnlinien, die zum großen Teil gemeinsame Strecken nutzen und ist etwa 30 km lang. Die jährliche Streckenleistung liegt bei über 5 Mio. gefahrenen Kilometern. Ziel der Studie war es, Parameter-Werte zu finden, bei der in möglichst kurzer Rechenzeit ein qualitativ hochwertiger, konfliktfreier Fahrplan erzeugt wird. Die Tests wurden größtenteils auf einem Notebook mit Pentium-4-Prozessor, Taktrate 2.4 GHz und 512 MB RAM durchgeführt.

5.1 Approximation des Wunschfahrplans

5.1.1 Abhängigkeit von der Größe des Startzeitintervalls

Mit dem Simulationssystem wurde der Einfluss verschiedener Parameter auf das Ergebnis (den ermittelten Fahrplan) untersucht. Ein Startzeitintervall für die jeweils erste Linienfahrt einer Linie ist erforderlich, um trotz eventuell in Konflikt stehender Wunschzeiten für die Linienfahrten einen widerspruchsfreien Plan generieren zu können. Der Startzeitraum der ersten Linienfahrt wird standardmäßig durch die Parameter `startMinTime` und `startMaxTime` festgelegt. In dieser Testreihe wird untersucht, wie sich eine Einschränkung des Startzeitintervalls auf die gesamte Verschiebung und die Dauer der Fahrplangenerierung auswirkt. In der Tabelle1 sind das Startintervall, die durchschnittliche Abweichung und die Abweichung für jede einzelne Linie dargestellt. Dabei berechnet sich die Abweichung einer Linie aus dem Durchschnitt der Verspätungen der Linienfahrten aller Fahrtperioden. Für die Simulationen werden starre Taktblöcke und maximale Längen der Gleisblöcke von 50 Metern verwendet. Die Simulationen zeigen, dass bei der Verwendung von Gleisblöcken mit einer Maximallänge von 50 Metern starren Taktblöcken das Startzeitintervall bis auf 75

Sekunden ohne signifikante Auswirkungen auf die Rechenzeit und die Größe der Abweichungen vom Wunschfahrplan eingeschränkt werden kann.

Tabelle 1: Einfluss des Startzeitintervalls auf Abweichungen

Start-zeit intev. [min]	*Gen.-Dauer [sec]*	*Ø-Abw. [mm:ss]*	*Linie* *1*	*2*	*3*	*4*	*5*	*6*	*7*
			[mm:ss]						
3	7,5	00:12	00:00	00:00	00:00	00:00	01:00	00:16	00:31
2	6,5	00:12	00:00	00:00	00:00	00:00	01:00	00:16	00:31
1,5	6,4	00:12	00:00	00:00	00:00	00:00	01:00	00:16	00:31
1,24	6,3	00:12	00:00	00:00	00:00	00:13	00:43	00:16	00:31

5.1.2. Abhängigkeit von der Größe des Planungsintervalls

Die Tabelle 2 zeigt die Größe der Gesamtabweichung der Fahrplanzeiten des berechneten Fahrplans vom Wunschfahrplan in Abhängigkeit von der Größe des simulierten Zeitintervalls. Auf die Gesamtsumme der Verschiebungen aller Fahrten des Wunschfahrplans, die in diesem Benchmark-Test als Qualitätsmaßstab für den Fahrplan interpretiert wird, hat die Planungsintervallgröße keine Auswirkungen.

Tabelle 2: Gesamtabweichungen vom Wunschfahrplan in Abhängigkeit von der Größe des simulierten Zeitintervalls

Dauer der Zeitintervalle [h]	*Anzahl der Zeitintervalle*	*Ausführungszeit der Simulation [sec]*	*Speicherplatz-bedarf [MB]*	*Abweichung vom Wunschfahrplan [hh:mm:ss]*
1	21	-		
2	11	-		
3	7	10,8	50	06:19:41
4	6	11,5	60	06:19:41
6	4	12,8	80	06:19:41
8	3	14,9	100	06:19:41
10	3	16,5	110	06:19:41
12	2	18,7	120	06:19:41
24	1	28,9	130	06:19:41

5.1.3 Abhängigkeit von der Blocklänge

Die Tabelle 3 zeigt die Größe der Gesamtabweichung der Fahrplanzeiten des berechneten Fahrplans vom Wunschfahrplan in Abhängigkeit von der Blocklänge. Die Planungsintervallgröße ist auch hier 3 Stunden. Das Simulationsergebnis widerlegt die These, dass eine kleinere Blocklänge – und schließlich eine Blocklänge 0 – eine flexiblere Planung erlaubt.

Tabelle 3: Gesamtabweichungen vom Wunschfahrplan in Abhängigkeit von der Blocklänge

Max. Blocklänge [m]	*Ausführungsdauer der Simulation [sec]*	*Speicherplatzbedarf [MB]*	*Abweichung vom Wunschfahrplan [hh:mm:ss]*
1000	10,5	50	06:47:55
500	11,1	50	06:19:41
200	14,3	60	03:53:39
100	19,9	90	03:48:09
50	32,3	130	04:04:27
38	40,6	160	04:07:06

Der Nachteil kleiner maximaler Blocklängen ist, dass mehr Speicher und Zeit benötigt wird, da mehr Berechnungsaufwand erforderlich ist. Die Grenzen für die Wahl der Blocklänge werden durch die Länge des längsten Gleises des untersuchten Netzes (die zwischen 500 und 1000 Metern liegt) und der Zuglänge der Straßenbahnen (38 Meter) bestimmt. Die Abweichung vom Wunschfahrplan ist bei ca. 100 Metern Blocklänge am kleinsten. Bei diesem Wert halten sich auch Rechenzeit und Speicherplatzbedarf im Rahmen und er ist der Fahrweise auf Sicht im Straßenbahnwesen angemessen. Sehr große Werte für die Blocklänge bestätigen die intuitive Erwartung, dass lange Blöcke lange Sperrzeiten für die betroffenen Strecken und damit auch größere Abweichungen bedeuten.

5.3 Leistungsfähigkeit von Strecken

Eine Untersuchung der Leistungsfähigkeit einer Strecke und/oder Linie bzw. von Störungen dient dazu, um entweder eine bedarfsgerechte Dimensionierung von Anlagen anhand geplanter oder prognostizierter Betriebsprogramme oder eine Prüfung der Durchführbarkeit von Betriebsprogrammen auf einer gegebenen Infrastruktur vorzunehmen. In der vorliegenden Arbeit interessiert die Frage, ob ein bereits geplanter Fahrplan mit dem gewünschten Linienaufkommen trotz eines Engpasses einzuhalten ist. Um die Kapazität einer Infrastruktur zu untersuchen, können analytische Verfahren oder Simulationsverfahren eingesetzt werden. Analytische Verfahren beruhen auf zu bestimmenden Kenngrößen, wie z. B. dem Belegungsgrad einer Trasse, um Aussagen zu treffen. Definitive Aussagen über die Durchführbarkeit eines konkreten Fahrplanes sind damit kaum möglich. Es wird ein leistungsfähiges Simulationsverfahren benötigt, das über das Erkennen von Konflikten hinaus diese zu vermeiden imstande ist. Während des Ablaufs der Simulation erhobene Daten können zur Visualisierung, zur Berechnung von Kenngrößen und zur weitergehenden Analyse verwendet werden. Die Annahmen für die durchzuführenden Simulationen waren:

- Es ist ein integraler Taktfahrplan gegeben, d.h. Züge einer Liniefahren in einem gewissen Zeitintervall in einem festen Zeitabstand, z.B. alle 10 min.
- Eine geringfügige Flexibilisierung des Fahrplans ist zugelassen, um einen adaptierten konfliktfreien Fahrplan zu ermitteln.
- Die Störungen sind so beschaffen, dass eine Variation des Störungsgeschehens (wie Zeitdauern von Sperrungen, Länge von Ampelphasen) möglich ist, um Bedingungen zu finden, die dem Soll-Fahrplan weitgehend entsprechen.

Für den integralen Taktfahrplan wird Sekundengenauigkeit vorausgesetzt, da ein minutengenauer Plan tatsächliche Konflikte verbergen aber vermeintliche Konflikte suggerieren könnte. Der zweite Punkt ist wesentlich, da z.B. Rotphasen von Ampelschaltungen eine strenge Einhaltung des Fahrplans im Allgemeinen unmöglich machen und letztlich die Fahrten trotz des Engpasses durchgeführt werden sollen. Auch mit diesem Ansatz könnte eine Linie, die eventuell Hauptkonfliktverursacher ist, identifiziert und eventuell herausgenommen werden.

Das konkret untersuchte Szenario stellt einen Engpass im Bereich zweier Haltestellen dar. Die auf insgesamt zwei Spuren reduzierte Trassenführung im Engpass muss sowohl von den Straßenbahnen als auch von den Bussen befahren werden und liegt im Kreuzungsbereich des Autoverkehrs, so dass auch die Ampelschaltungen berücksichtigt werden müssen. Die zu untersuchende Frage besteht darin, ob der bei der Fahrplanerstellung nicht vorhandene Engpass zu Fahrzeitverspätungen oder Staus führen kann und was eventuelle Auswege daraus sein könnten.

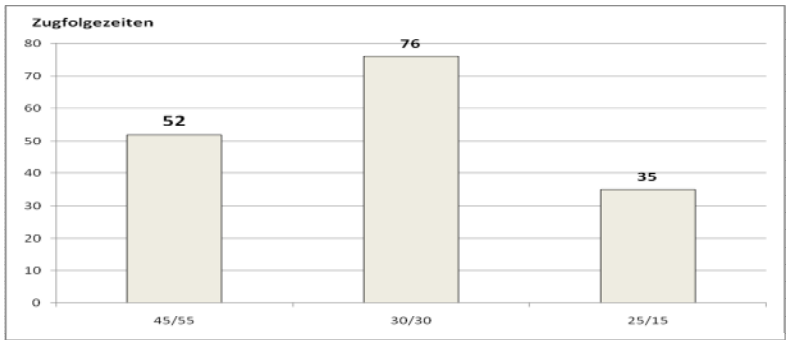

Abbildung 1: Zugfolgezeiten an den Verkehrsampeln

Die Simulationsparameter beziehen sich auf die Zeitintervalle für die Verschiebung von Abfahrzeiten und die Einhaltung der Taktgenauigkeit, die Fahrgeschwindigkeit, die Haltezeiten in Haltestellen, die Größe der Sperrzeiten für Gleisbelegungen, auf die Vor- und Nachbelegungszeiten und die Dauer der Ampelphasen. Die Auswertung erfolgt anhand der gemessenen Abweichungen vom Sollfahrplan, der Anzahl der Zugfolgezeiten unterhalb eines bestimmten Wertes (z.B. *20 sec*) und dem Belegungsgrad. Je größer diese Werte jeweils sind, umso größer ist die Wahrscheinlichkeit von Staus.

Um nicht den RailML-Standard erweitern zu müssen, wird eine Verkehrsampel in der Infrastruktur als „virtuelle“ Haltestelle, deren Streckenlänge 1 m beträgt, beschrieben. Zu der Linienbeschreibung wird eine „Ampellinie“, die eine virtuelle Haltestelle als Start- und Zielhaltestelle bedient, hinzugefügt. Mithilfe des Parameters `averageWaitTime` wird die Länge der zu simulierenden Rotphase angegeben. Die Blockzeit `timing` gibt die Gesamtdauer der Ampelphase an. Der Parameter `flexibleTimeblock` lässt die Verschiebung der Abfahrtszeiten aller Fahrten im

Taktblock, außer der ersten, um einen bestimmten Betrag (z.B. *30 sec*) zu. Um die Ampelphasen konstant zu halten, werden „Ampellinien“ von der Flexibilisierung der Taktblöcke ausgenommen.

Insgesamt passieren 17 Linien diesen Engpass und die Verkehrsampeln. Zu den Hauptverkehrszeiten besteht ein 10min-Takt. Zunächst liefert ein Simulationslauf ohne Berücksichtigung von Vor- und Nachbelegungszeiten Informationen über das Konfliktpotential. Für die Ampelschaltung 30/30 (30 Sekunden Grün-, 30 Sekunden Rotphase) ergeben sich, verglichen mit den anderen untersuchten Schaltungen, die meisten Zugfolgezeiten unter 20 sec und damit das Potential für eine höhere Störanfälligkeit [1]. Tatsächlich liefert die zusätzliche Berücksichtigung von Vor- und Nachbelegungen nur Lösungen (Bildfahrpläne), wenn Trassenverschiebungen um mehr als 60 sec (bis zu 120 sec) zugelassen werden. Die Analyse der Weg-Zeit-Diagramme ergab, vorausgesetzt, dass alle relevanten Aspekte der realen Situation durch die Simulationsparameter bereits erfasst werden konnten, die Umwandlung der dem Engpass nachfolgenden Haltestelle in eine Doppelhaltestelle auch diese Ampelschaltung, wenn gefordert, noch vertretbar machen würde.

6. Zusammenfassung und Ausblick

Die Arbeit beschreibt ein Verfahren zur Fahrplanung für ein Straßenbahnnetz unter Verwendung von Constraint-basierten Programmiertechniken, die eine effiziente Berücksichtigung variabler Haltezeiten, flexibler Taktblöcke und von Übergangsmöglichkeiten zulässt. Das entwickelte Fahrplanungssystem weist folgende Funktionalitäten auf:

- Das System verwendet als Eingabedaten standardisierte XML-Dateien. Zur Beschreibung der Linieninformationen und Parameter bezüglich der Lösungssuche wurde das bestehende LineInfo-Format des RailML-Standards den Bedürfnissen der Aufgabenstellung angepasst und Erweiterungen vorgenommen.
- Die Eingabedaten werden in ein verarbeitungsgerechtes Format transformiert und die Gleise werden in Gleisblöcke definierter Länge zerlegt.
- Zur Fahrzeitberechnung wird die Belegungsdauer für alle Fahrten ermittelt. Dabei können Wartezeiten und Behinderungszeiten in Haltestellen unterschiedlich berücksichtigt werden. Die Halte- und Behinderungszeiten können variabel definiert werden, so dass bei Konflikten unter Umständen nicht eine komplette Fahrt verschoben werden muss, sondern nur die Haltezeit verändert wird. Außerdem kann die Dauer der Vor- und Nachbelegung durch Angabe der Anfahrbeschleunigung bzw. der Bremsverzögerung verändert werden.
- Eine konfliktfreie Gleisblockbelegung sowie die Einhaltung der Taktungen werden durch Modellierung geeigneter Constraints gewährleistet, wodurch die Konfliktfreiheit des Fahrplans garantiert wird. Weiterhin kann durch Anwendung entsprechender Constraints die sekundengenaue Einhaltung der Taktung vermieden werden, ohne dass Auswirkungen auf den minutengenauen, öffentlichen Fahrplan entstehen.

- Zur Reduzierung der durchschnittlichen Wartezeiten der Fahrgäste kann eine globale Pufferzeit als Parameter übergeben werden, wodurch eine gleichmäßigere Verteilung der Fahrten resultiert.
- An Haltestellen können Umsteigepunkte zwischen zwei Linien mit einem definierten Übergangszeitintervall festgelegt werden. Die Einhaltung der Übergangszeit wird durch entsprechende Constraints gewährleistet.
- Optional werden als Ausgabeparameter für alle Haltestellen der Belegungsgrad und die durchschnittliche Wartezeit der Fahrgäste sowie für beide Parameter der Gesamtdurchschnitt, die Gesamt- und die Durchschnittsabweichung vom Wunschfahrplan zur Verarbeitung in Tabellenkalkulationsprogrammen (csv-Format) gespeichert. Anhand dieser Daten können z.B. die Betriebsqualität und Robustheit mit verschiedenen Parametern erzeugter Fahrpläne verglichen werden. Die Daten können auch für eine Kapazitätsanalyse der Bahnanlage genutzt werden.
- Eine Erweiterung des Fahrplanungssystems berücksichtigt Ampelschaltungen, so dass die Leistungsfähigkeit von Strecken untersucht werden kann.
- Der erzeugte Fahrplan wird graphisch auf dem Bildschirm ausgegeben und optional in einer XML-Datei gespeichert. Beim Exportieren in das Timetable-Format können formatbedingt nur die Belegungsdaten der Haltestellen gespeichert werden. Im `Timetable`-Format vorliegende Daten können eingelesen und visualisiert werden. Zukünftige Arbeiten sollen Mehrfachhaltestellen, eingleisigen Betrieb und eine umfangreiche Berücksichtigung von Hindernissen berücksichtigen können.

Literatur

[1] Coym, M., 2007, Constraint-basierte Kapazitätsanalyse eines Verkehrsengpasses im öffentlichen Personennahverkehr. Diplomarbeit, Universität Potsdam

[2] Franz, Ch., 2005, Constraint-basierte Planung für ein Straßenbahnnetz: Effektive Ressourcennutzung in der Umlaufplanung, Diplomarbeit, Universität Potsdam

[3] Geske, U., 2005, Railway Scheduling with Declarative Constraint Programming, Lecture Notes in Artificial Intelligence, Vol. 4369

[4] Hürlimann, D., 2003, Krauß, V. P.: RailML. Einheitliche Datenschnittstellen für Eisenbahnen, 19. VWT

[5] Pachl, J, 2004, Systemtechnik des Schienenverkehrs - Bahnbetrieb planen, steuern und sichern. Wiesbaden : Teubner Verlag

[6] railML.org-Initiative: http://www.railml.org (Stand: 20. Juni 2009)

[7] Schmidt, T., 2005, Constraint-basierte Planung für ein Straßenbahnnetz: Widerspruchsfreie Generierung von Fahrplänen, Diplomarbeit, Universität Potsdam

[8] Wolf, A., 2006, Object-Oriented Constraint Programming in Java Using the Library firstcs. In: Fink, Michael (Hrsg.) ; Tompits, Hans (Hrsg.) ; Woltran, Stefan (Hrsg.): 20th Workshop on Logic Programming, Vienna, Austria, Bd. 1843-06-02, S.21-32, Technische Universität Wien

[9] Zander, R., 2006, Constraint-basierte/objekt-orientierte Fahrplan-Erzeugung für ein Straßenbahnnetz mit Behandlung von Zugangs- und Übergangszeiten, Diplomarbeit, Universität Potsdam

[10] Zander, R., 2006, Vorschlag zur Darstellung von Linien, 10. Konferenz der railML.org-Initiative, http://www.railml.org/de/public/conferences.html (Stand: 20. Juni 2009)