Mehrdimensionale Multiressourcenplanung
mit Constraintlösern

von Diplom-Informatiker
Dirk Matzke
aus Berlin

von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. H.-U. Heiß
Gutachter:    Prof. Dr. S. Jähnichen
Gutachter:    Prof. Dr. T. Schaub

Tag der wissenschaftlichen Aussprache: 19. Dezember 2005

Berlin 2006
D 83
Danksagung

Mein Dank gilt vor allem Herrn Prof. Dr. Stefan Jähnichen, dem Leiter des Forschungsinstituts Fraunhofer FIRST, für die gebotene Möglichkeit, Projekt- und Forschungsarbeiten in einer fruchtbaren Symbiose auszuführen und am Dissertationsthema arbeiten zu können. Herrn Prof. Dr. Schaub möchte ich für sein Interesse an dieser Arbeit und für die Unterstützung bei einem großen Anwendungstest danken. Herrn Prof. Dr. Ulrich Geske, dem Leiter unserer Forschungsgruppe, und Herrn Dr. Joachim Goltz möchte ich danken. Über viele Jahre haben sie meine wissenschaftliche Entwicklung durch zahlreiche Anregungen und Diskussionen unterstützt. Das stets angenehme Arbeitsklima wirkte sich förderlich auf die wissenschaftliche Arbeit aus. Ich möchte meinem Kollegen Andrej Pohlmann danken, der das kreative Arbeitsklima des Bereiches Planungstechnik mitgetragen hat.

Für die Unterstützung bei der Erstellung dieser Arbeit möchte ich Herrn Dr. Jochen Kramer, dem Geschäftsführer der ASCI Systemhaus GmbH und Frau Dr. Boelant, der wissenschaftlichen Mitarbeiterin der ASCI Systemhaus GmbH danken, die mir mit vielfältigen Anregungen und Hilfen für meine Forschung zur Seite stand.

Kurzfassung


Bei der Problemzerlegung eines mehrdimensionalen Multiressourcenproblems werden die Vernetzung und die Struktur des Lösungsraums analysiert. Der Lösungsraum wird an geeigneten markanten Stellen, an denen sich weitgehend unabhängige Teillösungsräume ergeben, aufgeteilt. In einzelnen Teillösungsräumen wird separat und nacheinander entsprechend der ursprünglichen Anordnung eine Lösung gesucht. Eventuelle Auswirkungen der Lösung eines Teillösungsraums auf einen oder mehrere andere nachfolgende Teillösungsräume werden weitergegeben.


Die Erweiterung der Problemgröße zur Verarbeitung großer Multiressourcenprobleme wird mit Hilfe des Übergangs zur 64 Bit-Technik für die Datenspeicherung und für die Adressierung innerhalb des Constraintlösers erreicht. Der entwickelte Constraintlöscher kann deshalb unmittelbar die Vorteile einer Hardware mit 64 Bit-Architektur nutzen und damit zu einer weiteren Beschleunigung des Planungsprozesses beitragen.

Abstract

The thesis introduces a constraint solver for discrete multidimensional multiresource problems. Such problems include planning and order problems from the areas of production and personnel planning, timetabling and course planning, container packing, route planning or timetable generation for transport companies.

Using currently available commercial constraint solvers, the effort required to solve multiresource problems increases exponentially with problem size. This effort is significantly reduced for a much larger problem size by the constraint solver for multiresources problems developed and described in this thesis. In the thesis, a real application example is used to show that the increase in problem-solving effort can be drastically reduced for large problems too by decomposing the multidimensional multiresource problem in at least one dimension, by introducing central consistency-checking and range-management mechanisms in special value representations of the domain variables and by implementing special global constraints for modelling resources and order relations.

When decomposing a multidimensional multiresource problem, the interlinking and the structure of the solution space are analyzed. The solution space is split at suitable points, where largely independent partial solution spaces occur. In individual partial solution spaces, a solution is sought separately and successively following the original order. Possible effects of the solution of a partial solution space on one or more other subsequent partial solution spaces are passed on.

For data consistency checking during the solution search and the propagation on the resource's value ranges, there is - unlike when using conventional constraint solvers - no change to complexity-reducing interval propagation with exclusive restriction of the values at the interval borders. Instead, the value-exact restriction method is used, even for large value ranges. Even minimal inconsistencies (holes) are thus detected in otherwise closed value ranges.

To keep the number of constraints to a minimum, the constraint solver for multiresources problems manages with only a few simple constraints and two global constraints. This reduces constraint management effort. The global constraints include the order constraint RC, which enables the sequence of events to be fixed, and the constraint difn2D(), which ensures the non-overlapping allocation of resources. In its current form, the global constraint difn2D() can manage two-dimensional value ranges of up to 10 million values per dimension without interval processing and with low memory consumption. To solve multiresource problems, the two-dimensional global constraint difn2D() can be multiply combined. Here, a dimension is used as a reference dimension.

The extension of problem size for processing large multiresource problems can be achieved by transition to 64-bit technology for data storage and addressing within the constraint solver. The developed constraint solver can thus directly exploit the advantages of hardware with 64-bit architecture, thus helping to further speed up the planning process.

The thesis illustrates the characteristics of the constraint solver for multidimensional multiresource problems using as a concrete example the problem of long-term course planning for further-education institutions. The constraint solver was implemented in C++. The constraint solver was successfully tested not only for course planning but also - in cooperation with the Deutschen Bahn AG - for the highly complex problem of timetable and operation process simulation in rail networks.
Inhaltsverzeichnis

1 Einleitung .................................................. 7

2 Das Stundenplanungsproblem ....................... 11
   2.1 Beschreibung ........................................ 11
   2.2 Das Stundenplanungsproblem als Zuordnungsproblem ...... 11
   2.3 Bedarf an Stundenplansoftware .......................... 14
   2.4 Übersicht über verfügbare Stundenplansoftware .......... 15
   2.5 Constraintlöser für Multiressourcenprobleme .......... 16

3 Constraint Logische Programmierung ........... 19
   3.1 Constraint-Programmierung .......................... 19
      3.1.1 Erweiterung logischer Sprachen um build-in-Prädikate .... 19
      3.1.2 Erweiterung logischer Sprachen um Constraints .......... 21
   3.2 Constraintsysteme und Constraintlöser ................. 22
      3.2.1 Lösungen und Erfüllbarkeit ...................... 23
      3.2.2 Constraintlöser ................................... 25
   3.3 Constraintsystem über endlichen Domänen (fd) ........ 27
      3.3.1 Constraint Satisfaction Problem (CSP) ................ 27
      3.3.2 Konsistenz ....................................... 28
      3.3.3 Lösungssuche (Labeling / Domänenreduzierung) ...... 31
   3.4 Propagationstechniken .................................. 33
   3.5 Heuristische Suche .................................... 35
   3.6 Globale Constraints ................................... 38

4 Constraintlöser für Multiressourcenprobleme CS/4-MR ................................................................................. 47
   4.1 Ressourcen und Dimensionen ............................ 49
   4.2 Domäne und Constraints ................................ 50
      4.2.1 Domäne .............................................. 51
      4.2.2 Domänenzerlegung ................................ 53
      4.2.3 Nichtzeitenhierarchie ............................... 53
      4.2.4 Constraints ......................................... 55
   4.3 Globale Constraints ...................................... 56
      4.3.1 Ressourcen-Constraint ............................. 56
      4.3.2 Reihenfolge-Constraint ............................ 60

5 Suchverfahren für effiziente Planungen ........ 65
   5.1 Problemzerlegung ....................................... 65
      5.1.1 Algorithmus zur Problemzerlegung .................. 65
      5.1.2 Planung .............................................. 72
      5.1.3 Beschreibung der Algorithmen ..................... 74
      5.1.4 Heuristiken ......................................... 87
6 Modellierung von Stundenplanungsproblemen ........................................... 91
   6.1 Methoden: .......................................................................................... 91
      6.1.1 Beweis des Verfahrens zur Darstellung dreidimensionaler Körper in zwei zweidi-
            mensionalen Räumen ..................................................................... 93
      6.1.2 Beweis des Verfahrens zur Darstellung n-dimensionaler Körper in (n-1)-
            zweidimensionalen Räumen ......................................................... 94
   6.2 Vierdimensionales Ressourcenplanungsproblem ................................... 96
   6.3 Modellierung mit Constraints ............................................................ 97
      6.3.1 Domäne ....................................................................................... 97
      6.3.2 Einschränkung der "Nichtzeiten" - Feiertage, Urlaub, Pausen .......... 98
      6.3.3 Überschneidungsfreiheit ............................................................. 99
      6.3.4 Reihenfolgebeziehungen ............................................................. 99

7 Anwendungsbeispiel .................................................................................. 101
   7.1 Stundenplanung für Kurse ................................................................. 101

8 Ergebnisse und Bewertung ...................................................................... 107
   8.1 ConTime 1.0 ...................................................................................... 107
   8.2 Kursplan 1.0 ...................................................................................... 112
   8.3 Vergleich von ConTime und Kursplan .............................................. 116

9 Ausblick: .................................................................................................. 119
   9.1 Erweiterung des Constraint solvers $CS_{f^d-MR}$ um weitere Constraints und eine Optimie-
       rungsmethode .................................................................................... 120
   9.2 Simulation in Eisenbahnnetzen ......................................................... 120

10 Zusammenfassung .................................................................................. 123

A Funktionen des globalen Ressourcen-Constraints diffn2D ........................................ 137

B Parameter bei der Planung .................................................................. 145

C Schnittstelle ........................................................................................ 147
   C.1 Funktionen .................................................................................... 147
   C.2 Eigenschaften und Konstanten ....................................................... 148

D Stundenplanungssysteme .................................................................. 151

E Diagramme von ConTime 1.0 und Kursplan 1.0 ..................................... 153
   E.1 ConTime 1.0 .................................................................................... 155
   E.2 Kursplan 1.0 .................................................................................... 159
   E.3 Vergleich von ConTime 1.0 und Kursplan 1.0 ............................... 161
Kapitel 1

Einleitung


Es gibt für derartige Probleme verschiedene Softwarelösungen, doch mit zunehmender Komplexität und Detailiertheit ist eine schnelle Erzeugung genauer Lösungen derzeit nicht möglich. Die Programme zum Beispiel im Bereich der Stunden- und Fahrplanung liefern, wenn überhaupt, starre und ungenaue Lösungen, die vom Nutzer nicht interaktiv bearbeitet werden können. Die notwendigen Simulationen für die Fahrplanung sind zur Zeit nur ungenau mit einer hohen Abstraktion schnell durchführbar.


Zusammengefasst werden die neuen Modellierungsmethoden in einem Constraintlöser für spezielle Mul-
Die Erkenntnisse flossen in die Entwicklung eines Planungstools für Kurse an Weiterbildungseinrichtungen ein und wurden in der praktischen Anwendungen getestet.
Kapitel 2

Das Stundenplanungsproblem

2.1 Beschreibung

In der Praxis gibt es Planungs- und Simulationsprobleme mit einem hohen Detaillierungsgrad, die über einen großen Zeitraum geplant werden sollen. Zu diesen Problemen zählen zum Beispiel Jahresproduktionspläne in Minutenauflösung, Jahresfahrpläne für den öffentlichen Personennahverkehr (Straßenbahn, Bus), Jahresfahrpläne für den Eisenbahnverkehr (Züge), Jahrespläne für Weiterbildungseinrichtungen und Semesterpläne für Universitäten. Alle diese Pläne benötigen eine hohe Auflösung und beinhalten viele Details und spezielle Bedingungen.


2.2 Das Stundenplanungsproblem als Zuordnungsproblem


In Anlehnung an Kernler [48] werden alle Objekte eines Unternehmens, zu denen man Bedarf und Bestand verwalten kann, als Ressourcen bezeichnet. Im Gegensatz zu klassischen diskreten Zuordnungsproblemen, bei denen zwei diskrete Mengen von Angebots- und Nachfragespezifikationen einander zugeordnet werden, ist bei der kontinuierlichen Zuordnung eine Objektmenge auf Intervalle (meist Zeitintervalle, dann spricht man auch von Scheduling) der anderen Objektmenge abzubilden. Es sind zwei wesentliche Zuordnungen zu planen: die Deckung des Materialbedarfs durch Bestände und die Absicherung des Bedarfs an Kapazitäten der verschiedenen Fertigungseinrichtungen. Dabei ist darauf zu achten, dass
viele Fertigungs schritte mehrere Ressourcen gleichzeitig benötigen; wenn nur eine fehlt, kann die gesamte Fertigung eines Produkts nicht ausgeführt werden. Als Methode kommt zum Beispiel die Strategie **Vorschlagen und Vertauschen** zum Einsatz.


Die Lösung von zeitkontinuierlichen Zuordnungsproblemen erfolgt derzeit meist nach dem folgenden Ablauf:

- **Schritt 1**: Bereitstellung der Daten, Restriktionen und Wünschen
- **Schritt 2**: Auswahl eines Objekts aus der Objektmenge
- **Schritt 3**: Überprüfung der harten Constraints und Bestimmung des Startzeitpunkt des Objekts
- **Schritt 4**: Überprüfung der weichen Constraints
- **Schritt 4a**: Vertauschung der Objekte zur Minimierung der Summe der Prioritäten mit Zeitlimit
- **Schritt 4b**: Übernahme der relativ besten Teillösung
- **Schritt 5**: Aufruf von Schritt 2 bis alle Objekte bearbeitet sind
- **Schritt 6**: Verbesserung der Gesamtösung durch Vertauschung mit Zeitlimit
2.2. DAS STUNDENPLANUNGSPROBLEM ALS ZUORDNUNGSPROBLEM

Im zweiten Schritt wird das nächste einzuplanende Objekt ausgewählt. Hier kann die Reihenfolge der Objekte vorbestimmt werden, zum Beispiel durch eine Vorsortierung der Objekte nach ihrer Dringlichkeit, nach der Nutzung knapper Engpassressourcen, oder es wird das Objekt geplant, das die derzeit knappste Ressource benötigt.

Im dritten Schritt erfolgt die Zuordnung zu den benötigten Ressourcen. Zugeordnet werden aber nicht nur die Ressourcen an sich, sondern es wird auf allen benötigten Ressourcen ein passendes Zeitintervall gesucht, zu dem sie gleichzeitig zur Verfügung stehen. Gewählt wird ein Zeitintervall für das Objekt mit minimaler Verletzung der Randbedingungen. Es wird als sekundäres Kriterium berücksichtigt so, dass durch die Zuordnung zukünftige Zuordnungen möglichst wenig eingeschränkt werden (z.B. indem zu belegende Zeitintervalle einer Ressource so berechnet werden, dass möglichst große Blöcke von noch freien Zeitintervallen übrig bleiben).


Die Multiressourcenprobleme umfassen eine große Anzahl von Daten, so dass ein Lösen allein nur mit den zur Zeit verfügbaren Techniken der Constraintpropagation oder der kombinatorischen Optimierung in vertretbarer Zeit nicht möglich ist. Die Löschsuche erfordert einen zu langen Zeitraum, was eine operative Planung unmöglich macht. In dieser Arbeit wird ein Modell für einen Constraintlösers auf mehrdimensionalen, begrenzten Ressourcen definiert, der große Planungs- und Simulationsprobleme mit kurzen Antwortzeiten löst. Das Modell ist in verschiedenen Constraintdomänen anwendbar, weil die verwendeten Constraints kein domänenspezifischen Wissen enthalten, sondern nur allgemeine mathematische Bedingungen effektiv umsetzen. Die Variablen der Constraints werden durch Propagationsmethoden schrittweise eingeschränkt. Zu diesen Propagationsmethoden gehören vorausschauende

Der Constraintlöser wird prototypisch auf das Problem der Kursplanung in Weiterbildungseinrichtungen angewendet. Die Pläne in Weiterbildungseinrichtungen erstrecken sich über mehrere Wochen bis zu zwei Jahren und haben an jedem Tag einer zu planenden Woche eine andere Stundenzusammensetzung. In diesem Beispiel kann das Planungsproblem nicht auf eine Woche reduziert werden, wie es bei Schulen oder Universitäten möglich ist, da der Plan nicht in jeder Woche des Schuljahres/Semesters gleich ist, sondern in jeder Woche unterschiedlich.

2.3 Bedarf an Stundenplansoftware


Beider vierten und letzten Zielgruppe der Universitäten, Hoch- und Fachschulen wird eine sehr komplexe und skalierbare Stundenplanungssoftware benötigt, die eine Vielzahl von praktikablen Lösungsvorschlägen, die den individuellen Anforderungen der einzelnen Fakultäten und Bildungsstätten angepasst werden können, bereitstellt. Diese Vielzahl von Änderungsmöglichkeiten birgt einen hohen Planungsaufwand in sich.

2.4 ÜBERSICHT ÜBER VERFÜGBARE STUNDENPLANSOFTWARE

In Berlin gibt es zur Zeit ca. 480 Grundschulen, ca. 240 Weiterbildungsträger, ca. 200 Berufsbildende Schulen, ca. 120 Gymnasien, ca. 100 Sonderschulen, ca. 90 Realschulen, ca. 70 Gesamtschulen, ca. 60 Hauptschulen und ca. 10 Volkshochschulen, Fachhochschulen, Hochschulen und Universitäten. Das sind ca. 1370 Bildungsträger in Berlin. Von diesen Bildungsträger sind die Grundschulen sehr zahlreich vertreten, an zweiter Stelle liegen die sonstigen (privaten) Bildungsträger und an dritter Stelle stehen die Berufsbildenden Schulen. Von diesen drei Schultypen haben die sonstigen privaten Weiterbildungsträger, die eine eigene Zielgruppe bilden, die höchsten Anforderungen an die Stundenplanungsoftware. Sie benötigen neben der automatischen und interaktiven Stundenplanerstellung, der schnellen und über- schneidungsfreien Vertretungsplanerstellung auch die Möglichkeit der Editierbarkeit eines vorhandenen Stundenplans. Diese Editierbarkeit umfasst die Erstellung neuer Veranstaltungen, das Löschen von Veranstaltungen und die Änderung von Daten der einzelnen Veranstaltungen.


Im folgenden Abschnitt werden die Funktionalitäten der Planungskomponente von bekannter Stundenplansoftware näher untersucht.

2.4 Übersicht über verfügbare Stundenplansoftware


dem Veranstaltung angelegt, editiert und gelöscht werden können.
Bei der manuellen Stundenplanung können die Veranstaltungen auf einer visualisierten Stundenplanbank platziert werden. Im Modus der automatischen Planung übernimmt die Platzierung der Veranstaltungen die Stundenplanungssoftware.


2.5 Constraintlöser für Multiressourcenprobleme


den marktführenden Produkten, durch die Umsetzung von globalen Constraint nur in Basisvarianten, keine Alternative. Denn die globalen Constraints im ILOG-Scheduler und im CHIP5 ermöglichen eine bessere Modellierung und eine effizientere Suche bei der Problemlösung.


Abbildung 2.1: Lösungsaufwand in Abhängigkeit von der Problemgröße


Bei den meisten kommerziellen Anwendungsfällen von Constraintssystemen handelt es sich um die Planung oder Simulation von begrenzten Ressourcen. Ein Constraintssystem, welches sich auf die Behandlung von größeren Ressourcenproblemen in kürzerer Zeit spezialisiert hat, ist zur Zeit nicht verfügbar.

Kapitel 3

Constraint Logische Programmierung

3.1 Constraint-Programmierung


3.1.1 Erweiterung logischer Sprachen um build-in-Prädikate

Die Constraint-Programmierung entwickelte sich aus der Erweiterung der logischen Sprachen zunächst um build-in-Prädikate und später um Constraints.


Die Programmiersprache PROLOG steht für PROgrammieren in LOGik und bezeichnet eine Familie von logischen Programmiersprachen, wie zum Beispiel GNU-PROLOG und ECLiPSe-PROLOG.

Ein logisches Programm P (nach [43]) besteht aus einer Folge von Regeln der Form:

\[ q(s_1, \ldots, s_m) : \text{q}_1(s_{1,1}, \ldots, s_{1,n}), \ldots, q_k(s_{k,1}, \ldots, s_{k,r}), \quad k \geq 0, \]

wobei \( s_i, s_{i,j}, \text{Term} \) und \( q, q_i \) Prädikat-Symbole. Im Folgenden wird dafür \( Q : Q_1, \ldots, Q_k \) geschrieben. \( Q_i \) nennt man dabei ein Literal. Eine Regel \( Q : Q_1, \ldots, Q_k \), \( Q > 0 \) nennt man Klausel und eine Regel \( Q : Q_1, \ldots, Q_k \), \( k = 0 \), also \( Q \), wird als Fakt bezeichnet.
Bei jeder Regel \( Q \leftarrow Q_1, \ldots, Q_k \) heißt \( Q \) Kopf (head) der Regel und \( Q_1, \ldots, Q_k \) Körper (body) der Regel. Jede Klausel eines logischen Programms repräsentiert eine Formel der HORN-Klausel-Logik. Die Klauseln des Programms sind konjunktiv verbunden. Die Klausel \( Q \leftarrow Q_1, \ldots, Q_k \) ist als \( \forall X . \ Q \leftarrow Q_1 \land \ldots \land Q_k \) zu interpretieren, mit \( X \) als der Menge der in \( Q, Q_i \) vorkommenden Variablen (siehe auch [43]).

Auf Grund der logischen Äquivalenz von \( \forall X . \ Q \land \neg (Q_1 \land \ldots \land Q_k) \) bzw. \( \forall X . \ (Q \land \neg Q_1 \land \ldots \land \neg Q_k) \). Der Fakt \( Q \) repräsentiert die Formel \( \forall X . \ Q \). Die Regeln und Fakten sind so mit universell quantifizierte Disjunktionen von Literalen, bei denen genau ein Literal positiv (erfüllt / true) ist. Eine Horn-Klausel ist eine universon quantifizierte Disjunktion von Literalen, von denen höchstens eines positiv ist. Eine Horn-Klausel ohne positives Literal wird Ziel genannt [43].

Ein Ziel (goal) eines logischen Programms hat die Form \( \exists R_1, \ldots, R_l \). Die leere Klausel \( \exists R_1, \ldots, R_l \) repräsentiert. Die leere Klausel \( \exists \) gibt die Formel false false true, das heißt false wieder. Bei der Auswertung des logischen Programms \( P \) mit dem Ziel \( G = (\exists R_1, \ldots, R_l) \) wird versucht, eine Widerlegung von \( G \) aus \( P \) unter Nutzung der (SLD)-Resolution zu finden. Hierbei wird geprüft, ob \( G \) logische Konsequenz von \( P \) ist. Wenn diese Widerlegung ermittelt werden kann, dann liefert die Berechnung Werte für die Variable aus \( G \). Es lässt sich eine Antworthsubstitution \( \sigma \) ermitteln, so dass \( P \models \forall \sigma \ (R_1 \lor \ldots \lor R_l) \) gilt (nach [43]).

Die Herleitung von Widersprüchen aus einem Programm \( P \) und einem Ziel \( G \) basiert auf der Resolution. Zur Erklärung dieser Methode wird der Begriff Unifikation eingeführt.

Definition **Unifikator, allgemeiner Unifikator, mgu** (nach [43]): Seien \( s \) und \( t \) Terme. Eine Substitution \( \sigma \) mit \( \sigma(s) \) und \( \sigma(t) \) wird Unifikation von \( s \) und \( t \) genannt. Ein Unifikator \( \sigma \) von \( s \) und \( t \) wird allgemeiner Unifikator (most general unifier, mgu) von \( s \) und \( t \) genannt, wenn es für jeden Unifikator \( \phi \) von \( s \) und \( t \) eine Substitution \( \psi \) gibt, so dass \( \phi = \psi \circ \sigma \) gilt.

Beispiel [43]: Gegeben seien die Terme \( s = f(x, g(x)) \) und \( t = f(1, y) \). Es gilt \( \sigma(s) = f(1,g(x)) = \sigma(t) \).

Dann sei die Substitution \( \sigma = x/1, y/g(x) \) der Unifikator von \( s \) und \( t \). Gleichfalls ist die Substitution \( \phi = x/1, y/g(2), z/2 \) ein Unifikator von \( s \) und \( t \), denn es gilt \( \psi(s) \) 0 \( f(1, g(2)) = \phi(t) \).

In diesem Beispiel ist \( \sigma \) der allgemeinste Unifikator von \( s \) und \( t \), da es für \( \sigma \) und \( \phi \) ein \( \psi = z/2 \) gibt, so dass \( \psi \circ \sigma = \phi \) ist.

Die allgemeinste Unifikatoren sind bis auf Umbenennungen gleich, somit genügt es, einen solchen mgu zu berechnen.

Zur Herleitung eines Widerspruchs aus dem Programm \( P \) und dem Ziel \( G \) wird die SLD-Resolution benutzt.

Definition **SLD-Resolutionsschritt** (nach [43]): Gegeben sei ein logisches Programm \( P \) und ein Ziel \( G \equiv \exists R_1, \ldots, R_l \), mit \( l \geq 1 \). Wenn es eine Variable \( C = (Q, Q_1, \ldots, Q_m), m \geq 0 \), einer Regel aus \( P \) gibt, so dass keine Variable in \( G \) und \( C \) gleichzeitig auftritt und es ein \( i \in \{1, \ldots, m\} \) und einen allgemeinsten Unifikator von \( R_i \) und \( Q \) gibt, dann sei \( G' = G \equiv \exists \sigma(R_1), \ldots, \sigma(R_{i-1}), \sigma(Q_1), \ldots, \sigma(Q_m), \sigma(R_{i+1}), \ldots, \sigma(R_l) \). Dann wird \( G' \equiv \exists \sigma \) ein Resolutionsschritt genannt. Wenn \( C \) klar aus dem Kontext hervor geht, kann auch \( G' \equiv \sigma \) geschritten werden.

Mit der SLD-Resolution wird die lineare Resolution mit einer Selektionsfunktion für Literal vorliegt, bei der die Definition Klauseln (Horn-Klauseln) abgeleitet werden. Hierbei wird das Ergebnis eines Resolutionsschritts (Resolvent) jeweils im nächsten Resolutionsschritt weiter abgeleitet.

Für den Fall, dass \( P \cup G \) unendlich ist, kann ein Widerspruch mit beliebigen Selektionsfunktionen abgeleitet werden.

Definition **SLD-Ableitung** (nach [43]): Gegeben sei ein Programm \( P \) und ein Ziel \( G \). Eine SLD-Ableitung von \( (P,G) \) ist eine (eventuell unendliche) Sequenz \( G, G_1, G_2, G_3, \ldots \) von Zeilen, so dass \( G \equiv \sigma_1, C_1, G_1 \equiv \sigma_2, C_2, G_2 \equiv \sigma_3, C_3, G_3 \equiv \sigma_4, C_4, \ldots \).
Definition SLD-Refutation, SLD-Ableitung (nach [43]): SLD-Refutation von \((P, G)\) ist eine endliche SLD-Ableitung \(G, G_1, G_2, \ldots, G_n\) von \((P, G)\) mit \(n \in \mathbb{N}\), so dass \(G_n = \). ist.

Beispiel für SLD-Ableitung bzw. SLD-Refutation (siehe auch [43]):

\[
\text{append}([], X, X).
\]
\[
\text{append}(\langle X | GX, GY, Z \rangle) :\text{append}(GX, GY, GZ).
\]

Betrachtet wird eine SLD-Ableitung des Ziels \(G = \) ?- \(\text{append}(A, [5]B, [7, 5, 1])\).

\[
\neg_1 = \{ A/7[GX], X1/7, GY1/5[B], GZ1/5[1] \}, (2)
\]
\[
\neg_2 = \{ GX1/[, B/1], X2/5[1] \}, (1)
\]


3.1.2 Erweiterung logischer Sprachen um Constraints


Wie schon besprochen, wird in Prolog die Arithmetik mittels des built-in-Prädikats is/2 realisiert. Bei der Auswertung eines solchen Ausdrucks müssen die Variablen in den Argumenten vollständig instanziert (an einen Grundterm gebunden) sein. Somit müssen die Variablen an Werte gebunden sein,
damit das is/2 ausgewertet werden kann. Wenn das nicht der Fall ist, tritt ein Laufzeitfehler in PROLOG auf. Ohne die Verwendung eines built-in-Prädikats in Prolog müsste man die Zahlenwerte mit einer Konstante 0 und dem Konstruktor s (successor) ausdrücken. Die Zahl 2 würde durch s(s(0)) abgebildet werden. Bei einem Ziel kann es sich um ein Constraint handeln. Die Variablen eines Constraints müssen bei seiner Auswertung noch nicht an Werte gebunden sein. Daher kann das Constraint am Anfang des Programms stehen.


### 3.2 Constraintsysteme und Constraintlöser


In der logischen Sicht sind Constraints spezielle logische Formeln, deren Erfüllbarkeit gewährleistet werden soll.

**Definition Constraintsystem** (nach [6]):

Ein Constraintsystem $CS = (C,D,X)$ besteht aus:

- einer Menge $C$ von Constraints über Variablen aus $X$,
- einer Menge $D$ von Domänen für die Variablen $X$ und
- einer Menge $X$ von Variablen.

Es gibt verschiedene Constraintsysteme, von denen einige vorrangig bei der Lösung industrieller Probleme eingesetzt werden. Nachfolgend werden die wichtigsten Constraintsysteme betrachtet.

**Boolesches Constraintsystem:**

Das Constraintsystem $CS_{bool}$ wird für Probleme eingesetzt, die sich als aussagenlogische Erfüllbarkeitsprobleme (SAT-Probleme) formulieren lassen. Diese Probleme werden mit booleschen Gleichungen modelliert, deren Variablen entweder die Gültigkeit einer Aussage (Wertbelegung 1) oder deren Ungültigkeit (Wertbelegung 0) repräsentieren. Als Lösung für dieses Problem wird eine Belegung der Variablen mit 0/1 Werten gesucht, die alle booleschen Gleichungen erfüllt (nach [43]).

**Definition:** Das boolesche Constraintsystem:

$$Cs_{bool} = (C_{bool}, D_{bool}, X_{bool})$$ mit

- $C_{bool} \subseteq \{0, 1, \neg, \land, \lor, =\}$,
- $D_{bool} = \{0, 1\}$,
- $X_{bool} \subseteq Var_{bool}$ die booleschen Variablen

basiert auf der booleschen Algebra und enthält aussagenlogische Gleichungen über Variablen, die nur mit den Werten 0 und 1 belegbar sind. Die Menge der Constraints $C_{bool}$ beinhaltet alle booleschen Constraints.

Mit dem booleschen Constraintsystem lässt sich zum Beispiel das Zweifarbenproblem behandeln (siehe
3.2. **CONSTRAINTSYSTEME UND CONSTRAINTLÖSER**

auch [43]). Bei diesem Problem sollen mit 2 Farben 3 jeweils paarweise benachbarte Gebiete der Landkarte unterschiedlich eingefärbt werden. Die einzelnen Gebiete werden durch Variablen \( x, y, z \in X_{\text{bool}} \) und durch die Constraints wie folgt formuliert.

\[
((x \land \neg y) \lor (\neg x \land y)) \land ((x \land z) \lor (\neg x \land \neg z)) \land ((y \land \neg z) \lor (\neg y \land z)) = 1.
\]

Die Lösungen sind \( x = 0, y = 1, z = 0 \) oder \( x = 1, y = 0, z = 1 \).

**Constraintsystem über endlichen Domänen:**
Das Constraintsystem \( CS_{fd} \) wird für komplexe, kombinatorische Probleme eingesetzt. Mit diesem Constraintsystem \( CS_{fd} \) lassen sich viele praktische Probleme, wie die Planung von Personal, Stundenplänen, Kursplänen und Maschinenbelegungsplänen in Werkstätten, lösen. Es gehört zu den am meisten eingesetzten Constraintsystemen. Das Constraintsystem arbeitet auf endlichen Wertebereichen (finite domain - fd) für die Variablen (nach [43]).

Ein einfaches Constraintsystem:

\[
CS_{fd} = (C_{fd}, D_{fd}, X_{fd})
\]

- \( C_{fd} \subseteq \{0, 1, 1, ..., +, \ast, /, -, \neq, >, \le, \ge, \le\} \),
- \( D_{fd} = Z \),
- \( X_{fd} \subseteq Var_{fd} \) den Variablen deren Wertebereiche endliche, ganzzahlige Intervalle sind,

können arithmetische Gleichungen und Ungleichungen gelöst werden. Die Menge der Constraints \( C_{fd} \) (finite domain constraints) enthält die Constraints über endlichen Domänen, wie zum Beispiel: \( x \in \{1, 2, ..., 20\}, x > 2, y / x = 16 \).

Dieses einfache Constraintsystem \( CS_{fd} \) wurde durch verschiedene Erweiterungen den jeweiligen Problembereichen angepasst. So gibt es die wohl bekannteste Erweiterungen mit Constraints, um verschiedene Arbeitsgänge innerhalb einer vorgeschriebenen Zeit auf einer Maschine ohne zeitliche Überlappungen einzulassen. Aufgrund der Relevanz für praktische und industrielle Anwendungen ist dieses Constraintsystem \( CS_{fd} \) Gegenstand der derzeitigen Forschung. Im Kapitel "Constraintlöser für Multiressourcenprobleme" werden Erweiterungen zum Löschen n-dimensionaler Ressourcenprobleme vorgestellt, die Inhalt dieser Arbeit sind.

### 3.2.1 Lösungen und Erfüllbarkeit


**Erfüllbarkeit**

Definition Erfüllbarkeit (nach [43]):

Gegeben sei ein Constraintsystem \( CS = (C, D, X) \), dann ist eine Konjunktion \( \land_{j \in \{1, ..., n\}} c_j \) von Constraints \( c_1, ..., c_n \in C \)

- erfüllbar / konsistent in \( D \), wenn \( D \models \land_{j \in \{1, ..., n\}} c_j \) gilt,
nicht erfüllbar / inkonsistent in \( D \), wenn \( D \models \neg \exists j \in \{1,\ldots,n\} \ c_j \) gilt.

Lösungen

Definition Lösungen (nach [43]):

Gegeben sei ein Constraintsystem \( CS = (C,D,X) \) und eine Belegung \( \sigma: X \rightarrow D \), mit der Abbildung der Variablen \( X \) auf die Domänen in \( D \). Dann ist die Belegung \( \sigma \) eine Lösung einer Konjunktion \( \bigwedge_{j \in \{1,\ldots,n\}} c_j \) von Constraints \( c_1,\ldots,c_n \in C \), wenn \( (D,\sigma) \models \bigwedge_{j \in \{1,\ldots,n\}} c_j \) gilt.


\[(x \cdot y > 3) \land (x + 1 < y) \land (x,y \in [1,10])\]

wären einhundert Belegungen \((x \mapsto 1, y \mapsto 1), (x \mapsto 2, y \mapsto 1), \ldots, (x \mapsto 10, y \mapsto 10)\) also 10 \* 10 = 100 Varianten (Permutationen) durchzuprobieren. Dieses Vorgehen wird als 'Generieren und Testen' ('generate-and-test') bezeichnet.

Beim Verfahren 'Generieren und Testen' werden alle Variablen mit Werten belegt. Danach werden die Constraints getestet und anschließend wird entweder Backtracking ausgeführt, bei Nichterfüllung der Bedingungen (Constraints) mit dem Verwerfen der Wertebelegung der Variablen und dem Generieren einer neuen Wertebelegung der Variablen oder bei Erfüllung der Bedingungen, wurde eine Lösung gefunden. Solch ein naives Verfahren ist sehr aufwendig und schränkt den Wertebereich der Variablen von Constraints \( C_{i,d} \) ein und zwar jeweils genau auf einen Wert.


Beispiel:

Gegeben sei die Konjunktion linearer Ungleichungen \((y \leq 5x) \land (x \leq 5y)\). Man sieht durch den Schnitt der beiden Geraden \( y = 5x \) und \( x = 5y \) und die erfolgte Überlappung der jeweiligen Halbebenen, der \( \leq \)-Seite der Geraden, dass diese Konjunktion in \( D \) erfüllbar ist (Abbildung 3.1). Somit gilt: \( D \models \exists x \ \exists y: (y \leq 5x \land x \leq 5y) \). In der Abbildung 3.1 zeigt der schraffierte Bereich die Überlappung der beiden Halbebenen und damit die Menge der Belegungen von \( x \) und \( y \), die die Constraints erfüllen und damit Lösungen der Konjunktion sind.

Abbildung 3.1: Geometrische Interpretation von \( y \leq 5x \land x \leq 5y \)
Die Nichterfüllbarkeit der Konjunktion \((5y < x) \land (5x < y)\) in \(D\) lässt sich durch den Schnitt der zwei Geraden \(5y = x\) und \(5x = y\) und durch die zu suchende Überlappendung der jeweiligen Halbebenen, die \(<\)-Seite der Geraden, zeigen. Die zwei Halbebenen überlappen einander nirgends, und somit ist die Konjunktion auch nicht erfüllbar. Es gilt: \(D \models \neg \exists x \, \neg \exists y: (5y < x \land 5x < y)\). Die schraffierten Bereiche in Abbildung 3.2 zeigen die Menge der Belegungen von \(x\) und \(y\), die jeweils eines der Constraints in der Konjunktion erfüllen. Da kein Bereich den anderen Bereich überlappt, gibt es keine Lösung für diese Konjunktion.

![Abbildung 3.2: Geometrische Interpretation von \(5y < x \land 5x < y\)](image)

### 3.2.2 Constraintlöser

**Definition Constraintlöser (nach [43]):**

Ein Constraintlöser ist eine Sammlung von Tests und Operationen auf endlichen Konjunktionen von Constraints eines bestimmten Constraintsystems \(CS = (C, D, X)\).

Über Constraintlösern sind folgende Tests und Operationen definiert, wenn \(\Delta C = \{c_1 \land \ldots \land c_m \mid m \in \mathbb{N} \text{ und } c_1, \ldots, c_m \in C\} \text{ und } \delta C = \{C_1 \lor \ldots \lor C_l \mid l \in \mathbb{N} \text{ und } C_1, \ldots, C_l \in \Delta C\}:

**Erfüllbarkeit (Satisfiability / Consistency) (nach [43]):**

Gegeben ist eine beliebige Konjunktion von Constraints \(C \in \Delta C\). Mit dem Test \(\text{solve}: \Delta C \rightarrow \{\text{true, false, unbekannt}\}\) soll versucht werden zu entscheiden, ob die Konjunktion von Constraints \(C\) in der Domäne \(D\) erfüllbar ist oder nicht.

Der Test \(\text{solve}(C)\) muss korrekt sein. Wenn \(\text{solve}(C) = \text{true}\) ist, dann muss \(C\) in \(D\) erfüllbar sein und wenn \(\text{solve}(C) = \text{false}\) ist, dann muss \(C\) in \(D\) nicht erfüllbar sein. Der Test ist vollständig, wenn er für alle \(C \in \Delta C\) deren Erfüllbarkeit in der Domäne \(D\) entscheidet und es gilt \(\text{solve}(C) \in \{\text{true, false}\}\), d.h. es kommt kein \{unknown\} vor. Beim Vorkommen des Ergebnisses \{unknown\} ist der Test unvollständig.

**Folgerbarkeit (Entailment) (nach [43]):**

Gegeben ist eine beliebige Konjunktion von Constraints \(C \in \Delta C\) und eine beliebige Disjunktion von Konjunktionen von Constraints \(CD \in \delta C\) in \(D\). Mit dem Test \(\text{entail}: \Delta C \times \delta C \rightarrow \{\text{true, false, unknown}\}\), mit der Transformation \(T\) zur Umwandlung einer Konjunktion von Constraints \(\Delta C\) in eine Disjunktion von Constraints \(\delta C\), soll versucht werden zu entscheiden, ob aus der Konjunktion von Constraints \(C\) eine Disjunktion von Constraints \(CD\) in der Domäne \(D\) folgt oder nicht.

Der Test \(\text{entail}(C, CD)\) muss korrekt sein. Wenn \(\text{entail}(C, CD) = \text{true}\) ist, dann muss \(D \models \forall (C \rightarrow CD)\) gelten und wenn \(\text{entail}(C, CD) = \text{false}\) ist, dann muss \(D \models \neg \forall (C \rightarrow CD)\) gelten. Der Test ist vollständig, wenn er für alle \(C \in \Delta C\) und alle \(CD \in \delta C\) die Folgerbarkeit in \(D\) entscheidet und es gilt \(\text{entail}(C, CD) \in \{\text{true, false}\}\), d.h. es kommt kein \{unknown\} vor. Beim Vorkommen des
**Projektion / Elimination** (nach [43]):

Gegeben ist eine beliebige Konjunktion von Constraints \( C \in \triangle C \) und beliebige Variablen \( x^* \subseteq \text{var}(C) \), die nicht eliminiert werden sollen. Mit dem Verfahren der Projektion \( \text{proj}: \triangle C \times P(X) \to \triangle C \) sollen in einer gegebenen beliebigen Konjunktion von Constraints \( C \in \triangle C \) Variablen eliminiert werden. Das Verfahren \( \text{proj} \) berechnet dann eine Konjunktion von Constraints \( \text{proj}(C, x^*) \in \triangle C \), die äquivalent zu \( \exists_{x^*} C \) ist und es gilt: \( D \models \forall (\exists_{x^*} C \iff \text{proj}(C, x^*)) \) und für die \( x^* \subseteq \text{var} (\text{proj}(C, x^*)) \subseteq \text{var}(C) \).

Das Verfahren der Variablenelimination heißt Projektion, da es sich geometrisch als Projektion interpretieren lässt. Im \( n \)-dimensionalen Raum sind die Lösungen eines Constraintproblems ein Gebilde, das durch die Variablen des Constraintproblems aufgespannt wird. Die Projektion des Gebildes auf den \( (n-k) \)-dimensionalen Raum, der durch die Variablen \( x^* \) aufgespannt wird, ist dann gleichbedeutend mit der Elimination der Variablen, die nicht in \( x^* \) enthalten sind (Abbildung 3.3).

![Abbildung 3.3: Geometrische Projektion](image)

**Determinationsdetektion**: (nach [43]):

Gegeben ist eine beliebige Konjunktion von Constraints \( C \in \triangle C \) und beliebige Variablen \( x \in X^s \) einer beliebigen Sorte \( s \). Mit dem Test \( \text{det}: \triangle C \times X \to C \) soll getestet werden, ob der Wert einer beliebigen Variable \( x \) durch eine beliebige Konjunktion von Constraints \( C \) eindeutig bestimmt ist. Es wird geprüft, ob es eine Konstante \( z \) der Sorte \( s \) gibt, so dass die Gleichung \( x = \sum z \) aus \( C \) folgt oder nicht. Somit gilt \( \text{det}(C, x) = (x = \sum z) \), wenn \( D \models \forall (C \to x = \sum z) \) für eine Konstante \( z \) ist oder es gilt \( \text{det}(C, x) = (x = \sum z) \), wenn \( D \models \neg \forall (C \to x = \sum z) \) für alle Konstanten \( z \) ist.

Es wird davon ausgegangen, dass die Konstanten eindeutig bestimmt sind, dann ist die Funktion \( \text{det} \) wohldefiniert. Eindeutig bestimmt sind die zwei Konstanten \( a \) und \( b \), wenn für sie \( D \models a = \sum b \) gilt und somit dann für diese Konstanten \( a \equiv b \) gilt.

**Simplifikation**: (nach [43]):

Gegeben ist eine beliebige Konjunktion von Constraints \( C \in \triangle C \). Das Verfahren \( \text{simp}: \triangle C \to \triangle C \) vereinfacht eine beliebige Konjunktion von Constraints \( C \in \triangle C \). Wenn \( \text{simp}(C) = C' \) ist, gilt \( D \models C \iff C' \) und \( C' \) ist einfacher als \( C \). Für diesen Fall wird meistens die Konjunktion von Constraints \( C \) durch die äquivalente, einfachere Konjunktion \( C' \) ersetzt.

Es gilt für den Fall \( \text{simp}(C) = \text{false} \) wenn \( \text{solv}(C) = \text{false} \) ist und es gilt \( \text{simp}(C) = \text{true} \), wenn \( \text{entail}(C, \text{true}) = \text{true} \) ist.
3.3. **CONSTRAINTSYSTEM ÜBER ENDLICHEN DOMÄNEN (FD)**


**Definition Wohlverhalten von Constraintsösern** (nach [43]):
Ein Constraintsöser eines Constraintsystems $CS = (C, D, X)$ verhält sich wohl, wenn er für zwei beliebige Konjunktionen von Constraints $C, C' \in \Delta C$:

- mengenbasierter; das heißt, es gilt $\text{solve}(C) = \text{solve}(C')$, wenn $C = c_1 \land \ldots \land c_n, C' = c'_1 \land \ldots \land c'_m$ und $\{c_1, \ldots, c_n\} = \{c'_1, \ldots, c'_m\}$ gilt.
- monoton ist; das heißt, falls $\text{solve}(C) = \text{false}$ ist, dann auch $\text{solve}(C \land C') = \text{false}$.
- unabhängig von Variablenbenennungen ist; das heißt, $\text{solve}(C) = \text{solve}(C')$ ist, wenn $C$ eine Variante von $C'$ ist.

3.3 **Constraintsystem über endlichen Domänen (fd)**

Im vorigen Abschnitt wurde ein Constraintsystem über endlichen Domänen $CS_{fd} = (C_{fd}, D_{fd}, X_{fd})$ vorgestellt. Dieses Constraintsystem wird in diesem Abschnitt detaillierter untersucht, da es für praktische, kombinatorische Probleme relevant ist.

### 3.3.1 Constraint Satisfaction Problem (CSP)

Es werden $n$-ter Probleme betrachtet, bei denen die an der Problemdarstellung beteiligten Variablen durch Werte in endlichen Wertebereichen eindeutig bestimmt sind.

Für die Menge der Constraints $C_{fd}$ wird angenommen, dass es einstellige Constraints $C_E \{m, n\}$ und weitere einstellige Constraints $C_E \{e_1, \ldots, e_k\}$ gibt, wobei $e_1, \ldots, e_k$ Konstanten einer bestimmten Sorte sind, die definiert ist: für alle $x$ gilt $x \in (C_E \{e_1, \ldots, e_k\})_{fd}$ genau dann, wenn $x \in \{e_{1,fd}, \ldots, e_{k,fd}\}$ ist.

Es wird angenommen, dass es zu jedem Element der Trägermenge der Struktur ein entsprechendes Konstantensymbol gibt. Somit gibt es für jede Sorte $t$ und jedes Element $e \in D_{fd}$ ein Konstantensymbol $c$, so dass $c_{fd} = e$ ist und ein beliebiges Constraint $C_E \{m, n\}$ kann auch als Constraint $C_E \{e, \ldots, n\}$ betrachtet werden.

**Definition Constraint-Satisfaction-Problem (CSP)** (nach [43]):

Gegeben sei eine Konjunktion von Constraints über endlichen Domänen $(C \land x_1 \in D(x_1) \land \ldots \land x_n \in D(x_n)) \in \Delta C_{Sfd}$. Diese Konjunktion von Constraints über endlichen Domänen ist ein Constraint-Satisfaction-Problem (CSP), wenn

- $C = c_1 \land \ldots \land c_k$ ist, wobei $c_1, \ldots, c_k$ keine Constraints der Form $x \in D$ sind,
- $x_1, \ldots, x_n$ paarweise verschiedene Variablen sind,
- $\text{var}(C) \subseteq \{x_1, \ldots, x_n\}$ gilt,
- $x_i \in X^{int}$ und $D(x_i) = \{m_i, n_i\}$ oder $x_i \in X^{int}$ und $D(x_i) = \{e_{i,1}, \ldots, e_{i,m_i}\}$ mit $D(x_i)$ der Domän der Variable $x_i$ mit $i = 1, \ldots, n$ und unter der oben definierten Annahme gilt.
3.3.2 Konsistenz


Knoten-Konsistenz

Es gibt verschiedene Konsistenzarten für Constraint-Satisfaction-Probleme. Die Knoten-Konsistenz (die einfachste Form der Konsistenz eines CSP) liegt vor, wenn die Domänen der Variablen mindestens die einstelligen Constraints erfüllen.

Definition Knoten-Konsistenz (nach [43]):

Gegeben sei ein Constraint Satisfaction Problem CSP \( C \land x_1 \in D(x_1) \land \ldots \land x_n \in D(x_n) \) mit \( C = c_1 \land \ldots \land c_k \). Eine Variable ist knoten-konsistent (node consistent) wenn:

- \( \text{vars}(c) \) nicht einelementig ist (\( |\text{vars}(c)| \neq 1 \)) oder

- \( \text{vars}(c) = \{x\} \) ist und für jeden Wert \( d \in D_{fd}(x) \) eine Belegung mit \( x \mapsto d \) eine Lösung von \( c \) ist.

Somit ist ein gegebenes CSP knoten-konsistent, wenn alle Variablen des CSP’s knoten-konsistent sind.

Beispiele:

\[
(X > 4) \land (Y < 6) \land (X + Y = Z) \land (X \in [0,10]) \land (Y \in [0,10]) \land (Z \in [0,10])
\]

Dieses Beispiel ist nicht knoten-konsistent, da beispielsweise der Wert 3 in der Domäne von \( X \) das Constraint \( X > 4 \) verletzt und der Wert 7 in der Domäne das Constraint \( Y < 6 \) nicht beachtet.

\[
(X > 4) \land (Y < 6) \land (X + Y = Z) \land (X \in [5,10]) \land (Y \in [0,5]) \land (Z \in [0,10])
\]

In diesem Beispiel ist die Knoten-Konsistenz gewährleistet, da die Domänen entsprechend der zu erfüllenden Constraints eingegrenzt wurden.

Die Verfahren zu Prüfung der Knoten-Konsistenz haben ein Laufzeitverhalten von \( O(cd) \) [43] mit \( c \) der Anzahl der Constraints und \( d \) der Größe der größten Domäne, da für jedes Constraint und für jeden Wert in der Domäne ihrer Variable zu prüfen ist, ob der Wert dieses Constraint erfüllt ist.

Kanten-Konsistenz

Eine weitere Form der Konsistenz ist die Kanten-Konsistenz. Mit dieser Konsistenz wird gesichert, dass es zu jedem Wert in einer VariablenDomäne Werte in anderen Domänen gibt, so dass jedes binäre (zweiseitige) Constraint, einzeln betrachtet, erfüllt ist.

Definition Kanten-Konsistenz (nach [43]):

Gegeben sei ein Constraint Satisfaction Problem CSP \( C \land x_1 \in D(x_1) \land \ldots \land x_n \in D(x_n) \) mit \( C = c_1 \land \ldots \land c_k \). Ein Constraint \( c \) in der Konjunktion \( C \) ist kanten-konsistent (arc consistent) wenn:
3.3. CONSTRAINTSYSTEM ÜBER ENDLICHEN DOMÄNEN (FD)

- vars(c) nicht zweielementig ist (|vars(c)| ≠ 2) oder

- vars(c) = \{x, y\} ist und es gibt zu jedem Wert \(d \in D_{fd}(x)\) einen Wert \(e \in D_{fd}(y)\) und umgekehrt gibt es zu jedem Wert \(e \in D_{fd}(y)\) eine Wert \(d \in D_{fd}(x)\), so dass eine Belegung mit \(x \leftrightarrow d, y \leftrightarrow e\) eine Lösung von \(c\) ist.

Somit ist ein gegebenes CSP kanten-konsistent, wenn die Constraints \(c_1,...,c_k\) kanten-konsistent sind.

Beispiel:

\[(X > 4) \land (Y < 6) \land (X + Y = Z) \land (X \in [5,10]) \land (Y \in [0,5]) \land (Z \in [0,10])\]

Dieses Beispiel ist knoten- und kanten-konsistent, obwohl die Belegung für die Variable \(Z \mapsto 0\) keine Belegung für die Variablen \(X\) und \(Y\) ergibt, um das Constraint \(X + Y = Z\) zu erfüllen, da dieses Constraint mehr als zwei Variablen enthält und noch nicht betrachtete wurde.

Die Verfahren zu Prüfung der Kanten-Konsistenz haben ein Laufzeitverhalten von \(O(c^n d^n)\) [13] mit \(c\) der Anzahl der zweistelligen (binären) Constraints (\(c\) dem konkreten Aufwand) und \(d\) der Größe der größten Domäne (\(y\) dem konkreten Aufwand), da für jedes binäre Constraint alle Wertepaare geprüft werden müssen. Es wird in jedem Teilschritt bei einfachen Verfahren meistens nur ein Wert rausgefiltert, bis alle Domänen eindeutig oder leer sind.

Dieses einfache Verfahren wurde mehrfach verbessert, was zur Entstehung der AC-n Verfahren geführt hat. Es gibt die Verfahren AC1, AC2, AC3, AC4, AC5, AC6 und AC7.

Die Knoten- und Kanten-Konsistenz eignen sich zur Einschränkung der Wertebereiche in Constraint-Satisfaction-Problemen mit unären und binären Constraints und zur unvollständigen Prüfung der Erfüllbarkeit.

**lokale Konsistenz**


**Definition lokal Konsistenz** (nach [13]):

Gegeben sei ein Constraint-Satisfaction-Problem CSP \(C \land x_1 \in D(x_1) \land \ldots \land x_n \in D(x_n)\) mit \(C = c_1 \land \ldots \land c_k\). Ein Constraint \(c\) in der Konjunktion \(C\) ist lokal konsistent (local/hyper-arc consistent) wenn vars\((c) = \{x_1,\ldots,x_m\}\) und es gibt für jedes \(i \in \{1,\ldots,m\}\) und zu jedem Wert \(d_i \in D_{fd}(x_i)\) für alle \(j \in \{1,\ldots,m\} \setminus \{i\}\) Werte \(d_j \in D_{fd}(x_j)\), so dass eine Belegung mit \(x_i \mapsto d_i, \ldots, x_m \mapsto d_m\) eine Lösung von \(c\) ist.

Ein gegebenes CSP ist lokal konsistent, wenn die Constraints \(c_1,...,c_k\) lokal konsistent sind.

Beispiel:

\[(X \neq Y) \land (Y \neq Z) \land (Z \neq X) \land (X \in [0,1]) \land (Y \in [0,1]) \land (Z \in [0,1])\]

Dieses Beispiel ist ein binäres Constraint-Satisfaction Problem und es ist bezüglich der Kanten-Konsistenz lokal konsistent. Dieses binäre CSP hat aber keine Lösung, das heißt es ist inkonsistent, denn wenn man die Domänen untersucht, enthalten alle Domänen nur zwei Wert, es sollen aber laut der Definition des CSP's drei Variablen voneinander verschiedene Werte haben.

Die Verfahren zur lokalen Konsistenz werden zum Testen der Erfüllbarkeit von Constraint-Satisfaction-Probleme verwendet. Wie man im Beispiel sieht, sind die Verfahren der lokale Konsistenz Grenzen bei dem Lösen von CSP's gesetzt. Die notwendige stärkere Einschränkung der Domänen und damit

Verfahren zur Herstellung von lokaler Konsistenz, insbesondere von Kanten-Konsistenz, werden von praktischen Constraintlösern angewendet, um mit vertretbarem Aufwand die Domänen einzuschränken.

Grenzen-Konsistenz

Eine schwächere Form der Konsistenz wird eingesetzt, wenn die Domänen durch Intervalle definiert sind. In diesem Fall hat ein Constraint-Satisfaction-Problem (CSP) die Form:

\[ C \land x_1 \in [\min(x_1), \max(x_1)] \land ... \land x_n \in [\min(x_n), \max(x_n)]. \]

Bei diesem CSP lässt sich nur durch die Betrachtung und Anpassung der Intervallgrenzen effektiv eine Form von Konsistenz schaffen, die die Domänen einschränkt.

Definition Grenzen-Konsistenz (nach [43])

Gegeben sei ein CSP P \( \equiv C \land x_1 \in [\min(x_1), \max(x_1)] \land ... \land x_n \in [\min(x_n), \max(x_n)] \), wobei C = c_1 \land ... \land c_k sei. Ein Constraint c in der Konjunktion C ist grenzen-konsistent (bound consistent), wenn \( \text{var}(c) = \{x_1, ..., x_m\} \) ist, und es für jedes \( i \in \{1, ..., m\} \) und jeden Wert \( d_i \in [\min(x_i)/d, \max(x_i)/d] \) für alle \( j \in \{1, ..., m\} \setminus \{i\} \) Werte \( d_j \in [\min(x_j)/d, \max(x_j)/d] \subseteq R \) gibt, so dass eine Belegung mit \( x_1 \leadsto d_1, ..., x_m \leadsto d_m \) eine Lösung von c ist. Das gegebene CSP ist grenzen-konsistent, wenn die Constraints c_1, ..., c_k grenzen-konsistent sind.

Die Grenzen-Konsistenz ist gegenüber der lokalen Konsistenz abgeschwächt, da es nur zu den Intervallgrenzen der Domäne einer Variablen Werte in den durch die Domänen der restlichen Variablen definierten reellwertigen Intervallen geben muss, so dass die jeweiligen Constraints erfüllt sind. Es werden hierbei nur die Werte der Grenzen der Intervalle einer Variable betrachtete und nicht die einzelnen Werte in den Intervallen der Domäne der Variable.

Beispiel:

Das CSP \( X^2 = 2 \land X \in [1, 2] \) ist nicht kanten-konsistent, da weder \( 1^2 \) noch \( 2^2 \) gleich 2 ist. Dieses CSP ist jedoch grenzen-konsistent, da \( 1 \leq \sqrt{2} \leq 2 \) gilt.

Die Grenzen-Konsistenz wird benutzt, wenn die Herstellung der lokalen Konsistenz zu aufwendig ist, zum Beispiel wenn die Domänen der Variablen durch sehr große Intervalle gegeben sind.

Beispiel:

Das CSP P \( \land D \) mit \( S \equiv 12 \ast X + 107 \ast Y - 17 \ast Z + 5 \ast T = 2 \) und \( D \equiv X \in [1,1000] \land Y \in [1,1000] \land Z \in [1,1000] \land T \in [1,1000] \) benötigt zur Herstellung der lokalen Konsistenz eine Betrachtung von \( 10^{12} \) Wertetupeln.

Der Aufwand ist exponentiell und von der Anzahl der Variablen der Constraints abhängig. Um diesen Aufwand zu vermeiden, werden bei der Anwendung der Grenzenkonsistenz die Intervallgrenzen der Domänen so weit iterativ eingeschränkt, bis sich keine Domäne mehr verändert.

Die Einschränkung der Variablen kann durch die Überprüfung jeder einzelnen Belegung für alle Variablen erfolgen oder durch die Anwendung einer Intervalarithmetik erfolgen. Die Intervalarithmetik schränkt die Intervallgrenzen und damit die Wertebereiche der Variablen ein. Hierzu gelte: für alle beteiligten Variablen \( x_i \in [x_i^\text{min}, x_i^\text{max}] \).
3.3. CONSTRÄNTEBSYSTEM ÜBER ENDLICHEN DOMÄNEN (FD)

Beispiel:
Es sei das CSP \( (x + y > 20) \land (x + 5 < y) \land (x \in [1, 15]) \land (y \in [1, 15]) \) gegeben, für welches die Grenzen-Konsistenz zu prüfen ist.
Zuerst wird die Ungleichung \( (x + 5 < y) \) betrachtet. Es folgt, dass \( x^{\text{max}} < y^{\text{max}} - 5 \) und \( y^{\text{min}} > x^{\text{min}} + 5 \) und somit \( x^{\text{min}} < 15 - 5 \) und \( y^{\text{min}} > 1 + 5 \) gilt. Die Belegungen \( x \mapsto 15, x \mapsto 14, x \mapsto 13, x \mapsto 12, x \mapsto 11, x \mapsto 10 \) oder \( y \mapsto 1, y \mapsto 2, y \mapsto 3, y \mapsto 4, y \mapsto 5, y \mapsto 6 \) sind nun keine Lösungen mehr. Somit können die Domänen der Constraints \( x \in [1, 15] \land y \in [1, 15] \) durch \( x \in [1, 9] \land y \in [7, 15] \) ersetzt werden, ohne dass dadurch potentiell mögliche Lösungen ausgeschlossen worden sind.

\[
x + y > 20 \land x + 5 < y \land x \in [1, 15] \land y \in [1, 15]
\]

wird durch die Betrachtung der Ungleichung \( x + 5 < y \) zu:

\[
x + y > 20 \land x + 5 < y \land x \in [1, 9] \land y \in [7, 15]
\]

Danach wird die Ungleichung \( (x + y > 20) \) unter Beachtung der Einschränkungen der Ungleichung \( (x + 5 < y) \) betrachtet. Somit folgt, dass \( x^{\text{min}} > 20 - y^{\text{max}} \) und \( y^{\text{min}} > 20 - x^{\text{max}} \) und somit \( x^{\text{min}} > 20 - 15 \) und \( y^{\text{min}} > 20 - 9 \) gilt. Die Belegungen \( x \mapsto 1, x \mapsto 2, ..., x \mapsto 5 \) oder \( y \mapsto 1, y \mapsto 2, ..., y \mapsto 11 \) sind nun keine Lösungen mehr. Somit können die Domänen der Constraints \( x \in [1, 9] \land y \in [7, 15] \) durch \( x \in [6, 9] \land y \in [12, 15] \) ersetzt werden.

\[
x + y > 20 \land x + 5 < y \land x \in [1, 9] \land y \in [7, 15]
\]

wird durch die Betrachtung der Ungleichung \( x + y > 20 \) zu:

\[
x + y > 20 \land x + 5 < y \land x \in [6, 9] \land y \in [12, 15]
\]

In jeder einzelnen Betrachtung der zwei Ungleichungen sind weitere Einschränkungen nicht erreichbar. Mit diesen einfachen Verfahren werden die Möglichkeiten der Wertbelegungen ohne Verlust von Lösungen stark eingeschränkt. Von 15 * 15 = 225 Wertbelegungen der Variablen müssen nur noch vier Wertbelegungen je Variable also 16 Wertbelegungen der Variablen betrachtet werden.

Aus dieser Einschränkung ist noch nicht erkenntlich, dass nicht alle 16 Wertbelegungen eine Lösung des CSPs sind. Zum Beispiel ist die Belegung \( x \mapsto 7 \) und \( y \mapsto 12 \) keine Lösung und die Belegungen \( (x \mapsto 8, y \mapsto 14), (x \mapsto 8, y \mapsto 15) \) und \( (x \mapsto 9, y \mapsto 15) \) sind Beispiellösungen.

Wenn man annimmt, dass es zu jedem FD-Constrait (gewichtete Summe in Gleichungen oder Ungleichungen, Maxima, Minima, oder ähnliche Constraints) ein Verfahren zur Herstellung der Grenzen-Konsistenz (bounds-consistency) existiert, lässt sich damit die Grenzen-Konsistenz für beliebige CSP herstellen, deren Domänen der Variablen durch Intervalle bestimmt sind.
Es gilt: \( C_{\phi} \models C \Rightarrow \text{bounds-consistency}(C) \) und \( \text{bounds-consistency}(C) \) ist grenzen-konsistent für jedes beliebige CSP \( C \), dessen Domänen der Variablen durch Intervalle bestimmt sind.

Das Verfahren \( \text{bounds-consistency}(C) \) propagiert die Änderungen der Domänen der Variablen so lange im Constraintnetz, bis keine Veränderungen mehr auftreten (ein Fixpunkt berechnet wurde). Diese Veränderung der Domänen der Variablen hat im schlimmsten Fall einen exponentiellen Aufwand. Dieser Fall ist gegeben, wenn sich bei jeder Anwendung des Verfahrens \( \text{bounds-consistency}(C) \) auf alle Variablen bei jeder Variablen die Intervallgrenze um nur einen Wert vergrößert oder verkleinert, bis ein Widerspruch erkannt wird.

3.3.3 Lösungssuche (Labeling / Domän reduzierung)

Labeling


Domänenreduzierung

Die Verallgemeinerung der Labeling-Prozedur ist eine Domänenreduzierung. Die Wertzuweisung wird durch eine Einschränkung des Domänenbereichs der ausgewählten Variable ersetzt. Im Fall eines Backtracking wird die mengentheoretische Differenz zwischen dem alten Domänenbereich und dem reduzierten Domänenbereich als neuer Domänenbereich der entsprechenden Variablen betrachtet. Es folgt wiederum die Domänenreduzierung bezüglich dieser Variablen.

Lösen durch Rücksetzen (Backtracking)


Definition **Backtracking** (nach [43]):
Für ein beliebiges CSP C der Gestalt

\[ c_1 \land \ldots \land c_k \land x_1 \in D(x_1) \land \ldots \land x_n \in D(x_n), \]

wobei \( c_1, \ldots, c_k \) nicht von der Gestalt \( x \in D \) sind, gilt \( \text{backtrack}(C) = \text{false} \), wenn C inkonsistent ist und

\[ \text{backtrack}(C) = c_1 \land \ldots \land c_k \land x_{i_1} = e_1 \land \ldots \land x_{i_j} = e_j \land x_1 \in \{e_1\} \land \ldots \land x_n \in \{e_n\}, \]

wobei \( x_{i_1} = e_{i_1}, \ldots, x_{i_j} = e_j \) Zuweisungen und \( e_1, \ldots, e_n \) Konstanten sind, wenn C konsistent ist. Des weiteren ist bei Konsistenz von C jede Substitution mit \( x_1 \mapsto e_1^d \), \ldots, \( x_n \mapsto e_n^d \) eine Lösung des CSP C. Somit ist ein Verfahren \( \text{solve} \)

\[ \text{solve}(C) = \text{true}, \text{falls} \ \text{backtrack}(C) \neq \text{false} \text{ist oder} \]
\[ \text{solve}(C) = \text{false}, \text{falls} \ \text{backtrack}(C) = \text{false} \text{ist}, \]

ein wohldefiniertes, vollständiges Lösungsverfahren für beliebige CSP C, welches sich wohl verhält.

Durch die Kombination von Suche und Konsistenzherstellung kann eine Lösung eines CSP bestimmt werden. Hierfür wird das Verfahren \( \text{backtrack}(C) \) angewendet.

Beispiel:
Gegeben sei das CSP C der Gestalt \( (x + y > 20) \land (x + 5 < y) \land (x \in [1, 15]) \land (y \in [1, 15]), \) auf welches das Verfahren \( \text{backtrack}(C) \) unter Verwendung von \( \text{bounds-consistency}(C) \) angewendet wird.
3.4. PROPAGATIONSTECHNIKEN

Dann erhält man nach dem Aufruf von bounds-consistency($C$) das Ergebnis:

\[ C \equiv x + y > 20 \land x + 5 < y \land x \in [6, 9] \land y \in [12, 15]. \]

Zum Finden einer Lösung wird bounds-consistency($C \land x = 6$) aufgerufen und liefert:

\[ C_1 \equiv x + y > 20 \land x + 5 < y \land x \in [6, 6] \land y \in [15, 15]. \]

Eine Lösung für die Ungleichungen $x + y > 20 \land x + 5 < y$ ist $x \mapsto 6, y \mapsto 15$.

Weil die Variable x noch mehrere Werte annehmen kann wird backtrack($C_1 \land x = 7$) aufgerufen und liefert nach Aufruf von bounds-consistency($C_{16}$):

\[ C_2 \equiv x + y > 20 \land x + 5 < y \land x \in [7, 7] \land y \in [14, 14]. \]

Da die Domänen der Variablen nun genau den Wert enthalten, ist dies das Ergebniss des rekursiven Aufrufs von bounds-consistency($C$) und von backtrack($C$). Eine weitere Lösung für die zwei Ungleichungen $x + y > 20 \land x + 5 < y$ ist dann $x \mapsto 7, y \mapsto 14$.

3.4 Propagationstechniken

Propagation


Als Beispiel wird die Platzierung von drei Blöcken in einer Reihe verwendet. Es werden Ungleichungen der Form $Position_i + BreitesBlocks < Position_j$ und Gleichungen der Form $Position_k = Wert$ verwendet. Ausgegangen wird von Positionen zwischen 1 und 20 und einer Breite von 1 für jeden der drei Blöcke. Die drei Blöcke können an jeder beliebigen Position platziert werden. Durch die Definition der Ungleichungen werden die Möglichkeiten der Positionierung der Blöcke eingeschränkt. Es wird die Ungleichung $B_2 + 5 < B_1$ eingeführt, die besagt, dass die Position des Blocks $B_2$ zuzüglich des Abstands 5 kleiner sein muss als die Position des Blocks $B_1$. Außerdem wird die Ungleichung $B_3 + 10 < B_2$ eingeführt, die besagt, dass die Position des Blocks $B_3$ zuzüglich des Abstands 10 kleiner sein muss als die Position des Blocks $B_2$.

Diese Ungleichungen führen durch Constraintpropagation sofort zur Einschränkung der zulässigen Werte für die Positionen der Blöcke, z.B. kann durch die erste Ungleichung die Position des Blocks $B_2$ nur noch im Intervall von 1 bis 14 liegen. Die weiteren Einschränkungen sind im Ablauf der Propagation zu sehen. Im letzten Schritt werden schließlich durch die Festlegung des Wertes für die Position des Blocks $B_1$ auf 18 (aus dem möglichen Bereich von 18 bis 20), wieder durch Constraintpropagation, die Werte der übrigen Positionen fixiert.

$B_1 = B_2 = B_3 = 1 \ldots 20$

Constraint: $B_2 + 5 < B_1$

Propagation der Variablen $B_1$ und $B_2$
$B_1 = 7 \ldots 20$, $B_2 = 1 \ldots 14$, $B_3 = 1 \ldots 20$

Constraint: $B_3 + 10 < B_2$
Propagation der Variablen $B_1$, $B_2$ und $B_3$

$B_1 = 18 \ldots 20$, $B_2 = 12 \ldots 14$, $B_3 = 1 \ldots 3$

Constraint: $B_1$ auf 18
Propagation der Variablen $B_1$, $B_2$ und $B_3$

$B_1 = 18$, $B_2 = 12$, $B_3 = 1$


Forward checking und Look ahead


Forward checking

Mit Forward checking werden die Constraints zwischen der aktuellen Variable und den Variablen, die noch keinen Wert zugeordnet bekommen haben, überprüft.


Look ahead


In der Abbildung 3.4 (nach [6]) wird gezeigt, welche Constraints (Kanten in den gestrichelten Rechtecken) bei der Anwendung der verschiedenen Propagationstechniken getestet werden. Beim Backtracking wird das Constraint zwischen der aktuellen und dem vorhergehenden Variablen (Knoten) getestet.
3.5 Heuristische Suche


Abbildung 3.4: Übersichtsbild (Backtracking, Forward checking, Look ahead)

3.5 Heuristische Suche

In einem Constraintnetz lassen sich durch heuristische Suche Inkonistenzen oder Lösungen finden. Die Vorgehensweise bei der Suche hat Einfluss auf die Geschwindigkeit des Suchprozesses. Für die Gestaltung der Suche lassen sich keine allgemeingültigen Regeln angeben. Es wird deshalb auf bekannte Heuristiken zurückgegriffen, die schon bei ähnlichen Problemen in kurzer Zeit eine gute Lösung gefunden haben. Bei der Suche gibt es zwei nichtdeterministische Stellen:

- die Auswahl der Variablen (Variablenreihenfolge), deren Werte zu bestimmen sind und
- die Auswahl der Werte (Wertereihenfolge) der Variablen.

Die Geschwindigkeit der Suche wird durch die Reihenfolge, in der die Variablen mit Werten belegt werden (Variablenreihenfolge - variable ordering), und durch die Reihenfolge der Werte, mit denen die Variablen belegt werden (Wertereihenfolge - value ordering), bestimmt.

Variablenreihenfolge

Es gibt zwei Möglichkeiten (nach [43]), wie die Variablen bei der Suche belegt werden. Bei der ersten Möglichkeit, der statischen Festlegung, ist die Reihenfolge der Variablen zur Wertebelegung für den gesamten Suchprozess fest vorgegeben. Bei der zweiten Möglichkeit, der dynamischen Festlegung, wird die nächste Variable zur Wertebelegung nach dem Zustand des Suchprozesses dynamisch bestimmt.

Die dynamische Festlegung der Reihenfolge der Variablen benötigt die Auswertung von zusätzlichen Informationen während des Suchprozesses und ist aufwändiger als die statische Festlegung. Der Einsatz der dynamischen Festlegung ist vorteilhaft, wenn der Mehraufwand für die Zusatzinformationen durch eine schnellere Suche als bei der statischen Festlegung wettgemacht wird.

Es gibt problempezifische Auswahlstrategien (Heuristiken) wie die Engpasanalyse von Arbeitsaufträgen, die auf einer Maschine eingelastet werden sollen und problemunabhängige Auswahlstrategien wie das "first-fail"-Prinzip. Bei diesem Prinzip werden zuerst die Variablen versucht zu labeln, bei


Wertereihenfolge

Nach der Auswahl der zu belegenden Variable ist ein Wert aus der Domäne der Variablen für die Belegung der Variable auszuwählen. So wie die Variablenreihenfolge hat auch die Wertereihenfolge Einfluss auf die Geschwindigkeit des Suchprozesses, da nicht nur die richtige Auswahl der Variable sondern auch die richtige Auswahl des Wertes entscheidend für das Finden einer Lösung ist (nach [43]).

Eine mögliche Strategie zur Auswahl der Werte einer Variablen ist das "succeed-First"-Prinzip. Bei diesem Prinzip (nach [43]) wird der erfolgversprechendste Wert einer Variable zuerst ausgewählt. Wenn die Variable feststeht, die als nächstes zu belegen ist, muss ein Wert gefunden werden, mit der die Variable belegt werden soll. Ist das CSP lösbar, dann soll der Wert ausgewählt werden, der Bestandteil der Lösung des CSP's ist. In diesem Fall ist die Reihenfolge der Auswahl der Werte entscheidend, da man einen Wert benötigt, der das CSP löst. Basierend auf diesem Prinzip kann im Constraintnetz bei einer Variablen zuerst einer der Werte ausgewählt werden, der die meisten Alternativen für die unbelegten Variablen offen lässt. Ein Wert einer Variable, der die meisten Alternativen für die unbelegten Variablen offen lässt, ist auch der Wert mit der geringsten Domäne einschränkung, was zu einer geringen Propagation über die Constraints führt und einen erhöhten Suchaufwand zur Folge hat. Die Domänen werden entweder durch die Propagation der Constraints oder durch die Suche auf einen Wert, die Lösung des CSP's, eingeschränkt.

Eine allgemeinere Strategie zur Auswahl der Werte einer Variablen ist das Prinzip der Domäneneinduktions-Prinzip. Bei diesem Prinzip (nach [43]) wird die Wertauswahl solange wie möglich verzögert, um nicht vorzeitig eine definierte Entscheidung zu treffen. Bei der Verzögerung der Wertauswahl wird die jeweilige Variable nicht mit einem Wert belegt, sondern der Wertebereich der Variable mittels Intervallachterhaltung beschränkt.

3.5. HEURISTISCHE SUCHE

Aufbrechen von Symmetrien


Beispiel:
Gegeben seien ein CSP \((A + B = S) \land (A \in D(A)) \land (B \in D(B)) \land (S \in D(S))\) mit \(D(A) = D(B)\). Bei diesem CSP wird die kommutative Operation “+” verwendet, was ein Indikator für probleminhärente Symmetrien ist und zur Erzeugung von symmetrischen Lösung bei dem Suchprozess führt. Durch das Hinzufügen des Constraints \(A \leq B\) wird eine Ordnungsrelation auf den Variablen definiert und die Erzeugung von symmetrischen Lösungen verhindert. Auch mit dem neuen Constraint können alle Lösungen des CSP’s durch die Permutation der Wertebearlegung von A und B erzeugt werden. Es wurden nur die symmetrischen Lösungen durch das zusätzliche Constraint ausgeschlossen.

Das Finden von versteckten Symmetrien und deren Aufbrechen ist bei der Modellierung ein großes Problem und nimmt viel Zeit in Anspruch. In den meisten praktischen Anwendungen werden die Symmetrien erst nach der Modellierung bei der Auswertung der Suchergebnisse entdeckt.

Redundante Constraints

Bei der Modellierung von Constraint-Satisfaction-Problemen können redundante Constraints zur Verbesserung der Propagation und damit zur Beschleunigung des Suchprozesses eingesetzt werden. Das Absetzen zusätzlicher Constraints in einem Constraintnetz hat keinen Einfluss auf die Menge der Lösungen des Problems. Mit den zusätzlichen Constraints soll eine zusätzliche Einschränkung der Domänen der Variablen und damit eine Reduzierung des Suchraums erreicht werden.

Definition redundant Constraints (allgemein) (nach [43]):

Gegeben sei ein Constraintssystem \(CS_{fd} = (C, D, X)\). Ein Constraint \(c \in C\) ist redundant in einer Konjunktion von Constraints \(C \in \Delta C\), wenn \(D \models (C \land c) \iff C\) oder \(D \models C \rightarrow c\) gilt.


Beispiel (Redundante Constraints mit keiner Reduzierung des Suchraums):

Gegeben sei ein CSP \((A + B = S) \land A \in [0,20] \land B \in [0,20] \land S \in [0,20]\). Diesem CSP werden die redundanten Constraints \(A \geq 0 \land A \leq 20 \land B \geq 0 \land B \leq 20 \land S \geq 0 \land S \leq 20\) hinzugefügt. Die hinzugefügten redundanten Constraints enthalten überflüssige Redundanzen, da die Constraints unmittelbar aus den Domänen der Variablen folgen und keine weiteren Informationen zur Reduzierung es
Suchraums enthalten. Die redundanten Constraints verlängern nur die Laufzeit des Suchprozesses.

**Redundante Constraints und Aufbrechen von Symmetrien**


Beispiel (Redundante Constraints und Aufbrechen von Symmetrien mit Reduzierung des Suchraums) (siehe auch [45]):

Gegeben sei ein CSP \((A \cdot A + B \cdot B = C \cdot C) \land A \in [0,100] \land B \in [0,100] \land C \in [0,100]\). Aus der Definition des CSPs folgt, dass \(A \leq C\) und \(B \leq C\) gilt. Dann kann das CSP in folgendes äquivalente CSP \((A \cdot A + B \cdot B = C \cdot C) \land (A < C) \land (B < C) \land A \in [0,100] \land B \in [0,100] \land C \in [0,100]\) überführt werden.

Zwischen den Variablen A und B liegt eine Symmetrie vor, da A und B kommutativ verknüpft sind. Diese Symmetrie kann durch die Forderung, dass \(A \leq B\) gelten soll, aufgebrochen werden. In dem CSP wird zusätzlich das Constraint \(A \leq B\) hinzugefügt:

\[A \cdot A + B \cdot B = C \cdot C \land A < C \land B < C \land A \leq B \land A \in [0,100] \land B \in [0,100] \land C \in [0,100].\]

Im ursprünglichen CSP konnten die Lösungen durch Vertauschung der Belegungen der Variablen A und B erzeugt werden. Aufgrund dieser Eigenschaft kann ein redundantes Constraint \(3 \cdot B \geq 2 \cdot C\) hinzugefügt werden. Dieses Constraint ergibt sich, da \(A \leq B\) gefordert wird und \(2 \cdot B \cdot B \geq C \cdot C\) ist, folglich ist \(\sqrt{2} \cdot B \geq C\), somit \(1.5 \cdot B \geq C\). Die Umformung auf ganze Zahlen ergibt dann \(3 \cdot B \geq 2 \cdot C\). Das CSP wird nun um das redundante Constraint \(3 \cdot B \geq 2 \cdot C\) erweitert:

\[A \cdot A + B \cdot B = C \cdot C \land A < C \land B < C \land A \leq B \land 3 \cdot B \geq 2 \cdot C \land A \in [0,100] \land B \in [0,100] \land C \in [0,100].\]

Durch das Aufbrechen der Symmetrie und die Hinzufügung eines redundanten Constraints hat sich das ursprüngliche CSP wesentlich vergrößert. Zu \((A \cdot A + B \cdot B = C \cdot C) \land A \in [0,100] \land B \in [0,100] \land C \in [0,100]\) wurde \((A \leq B) \land 3 \cdot B \geq C\) und die aus der Definition des CSP’s folgender Constraints \(A < C \land B < C\) hinzugefügt. Diese Erweiterungen des ursprünglichen CSP’s sind nur sinnvoll, wenn dadurch der Suchraum eingeschränkt wird, denn jedes zusätzliche Constraint im Constraintnetzwerk erzeugt zusätzlichen Aufwand in der Constraintverarbeitung, was zur Verlängerung der Laufzeit des Problemlösungsprozesses führt. Das Erkennen und Aufbrechen von Symmetrien, das Folgendes und das abgewägte Hinzufügen redundanten Constraints ist somit an Erfahrungen in der Constraint-Programmierung gebunden.

### 3.6 Globale Constraints

3.6. GLOBALE CONSTRAINTS

Paarweise Verschiedenheit (alldifferent)

Bei der paarweisen Verschiedenheit (siehe auch [43]) sollen alle betrachteten Variablen verschiedene Werte annehmen. Diese Forderung kann man mit paarweisen Ungleichungen \( X_i \neq X_j \) für \( 1 \leq i < j \leq n \) spezifizieren. Für \( n \) Variablen sind dann \( (n \times (n-1))/2 \) Ungleichungen notwendig. Bei 100 Variablen werden \( (100 \times (100 - 1))/2 = 9900/2 = 4950 \) Ungleichungen benötigt. Diese Formulierung der paarweisen Ungleichheit ist aufwendig und wenig effektiv in der Verarbeitung. Sie lässt sich durch ein globales Constraint, das Constraint alldifferent/1 ausdrücken und effektiv im Constraintnetzverarbeitung.

Definition alldifferent/1

Gegeben seien \( n \) Variablen \( X_1, ..., X_n \) die verschiedene Werte annehmen sollen. Das Constraint

\[
\text{alldifferent}([X_1, ..., X_n]) \text{ mit } n \in \mathbb{N}
\]

realisiert, dass gilt:

\[X_i \neq X_j \text{ mit } 1 \leq i < j \leq n.\]

Ein Problem, bei dem \( n \) Variablen mit endlichen Domänen paarweise verschiedene Werte annehmen sollen, ist äquivalent zum Maximalen-Bipartiten-Graphen-Matching. Für dieses Matching sind entsprechende Algorithmen bekannt. Diese Algorithmen des Matchings werden in dem Constraint alldifferent/1 eingesetzt (siehe [67]).

Die Anwendung des globalen Constraints alldifferent/1 mit den Algorithmen des Maximalen-Bipartiten-Graphen-Matching ist wesentlich effektiver als die Modellierung des gleichen Problems mit Ungleichungen.

Belegung einer Ressource (disjunktive/serialize)


Das Ressourcenbelegungsproblem besteht darin, für eine bestimmte Ressource \( n \) Arbeitsgänge mit Startzeiten \( S_1, ..., S_n \in \mathbb{N} \) und den Werten für die Dauer \( D_1, ..., D_n \in \mathbb{N} \) so zu planen, dass sich jeweils zwei unterschiedliche Arbeitsgänge nicht überlagern. Das bedeutet, dass für zwei Arbeitsgänge \( i \) und \( j \) mit \( i \neq j \) entweder \( s_i + D_i \leq S_j \) oder \( s_j + D_j \leq S_i \) gilt.

Diese Probleme könnte man als CSP mit Ungleichungen modellieren. Hierfür müssten alle \( 2^{n \times (n-1)/2} \) möglichen Kombinationen von Ungleichungen zusammen mit den Domänen der Startzeiten als CSP modelliert werden. Diese Modellierung hätte einen exponentiellen Modellierungsaufwand. Diese kombinatorische Explosion der Modelle lässt sich durch die Verwendung von binären Variablen, so genannte Schaltervariablen, durch \( (n \times (n-1))/2 \) Variablen der Form \( B_{i,j} \in \{0,1\} \) mit \( i < j \) und der Formulierung der ursprünglichen Disjunktion als Konjunktion \( S_i + D_i \leq S_j + B_{i,j} \times \text{maxInt} \land S_j + D_j \leq S_i + (1 - B_{i,j}) \times \text{maxInt} \) vermeiden. Bei dieser Definition ist ein \( \text{maxInt} \in \mathbb{N} \) hinreichend groß zu wählen. Das Problem der exklusiven Ressourcenbelegung kann dann als CSP der Form:

\[
(\bigwedge_{1 \leq i < j \leq n} S_i + D_i \leq S_j + B_{i,j} \times \text{maxInt}) \land (S_j + D_j \leq S_i + (1 - B_{i,j}) \times \text{maxInt})
\]

In den heutigen Constraint-Programmiersystemen wird dieses Ressourcenbelegungsproblem aufgrund des hohen Modellierungs- und Konsistenzherstellungsaufwands als globales Constraint schon vom Programmiersystem her realisiert. In den einzelnen Programmiersprachen ist dieses globale Constraint zur überlappendsfreien Ressourcenbelegung mit unterschiedlichen Bezeichnungen und Funktionsumfängen ausgeprägt. Die allgemeinste Form ist das disjunctive/serialize/2-Constraint mit:

\[
\text{disjunctive/serialize}([S_1, \ldots, S_n], [D_1, \ldots, D_n]).
\]


**Globales Constraint (cumulative)**


Dieses Ressourcenproblem kann mit einer einfachen Form des Constraints cumulative/4 modelliert werden. Hierfür wird für jede der n Tasks eine Domänenvariable \( S_i \) für die Startzeit, eine Domänenvariable \( D_i \) für die Dauer der Bearbeitung und eine Domänenvariable \( R_i \) für die Größe der Ressourcenutzung der begrenzten Ressource eingeführt. Mit einer Domänenvariable \( \text{Limit} \) im Bereich von 0 bis zum verfügbaren Ressourcenlimits wird das Limit für die Ressourcenutzung der begrenzten Ressource angegeben. Das Constraint cumulative hat folgende Form:

\[
\text{cumulative}([S_1, S_2, \ldots, S_n], [D_1, D_2, \ldots, D_n], [R_1, R_2, \ldots, R_n], \text{Limit})
\]

Das Constraint cumulative hat folgende mathematische Definition:

\[
\forall i \in [\min_{1 \leq j \leq n}(S_j), \max_{1 \leq j \leq n}(S_j + D_j)] : \sum_{k : s_k \leq \inSJ + D_k} R_k \leq \text{Limit}
\]

Das bedeutet, dass zu jedem Zeitpunkt zwischen dem Start der ersten Task und dem Ende der letzten Task der Wert für die Ressource, die von allen Tasks genutzt wird, zu jedem Zeitpunkt kleiner oder gleich dem gesetzten Limit ist. Die mathematische Definition des Constraints cumulative lässt sich wie in Abbildung 3.5 gezeigt veranschaulichen. In der Abbildung ist das generierte Profil des Constraints cumulative als eine Menge von Tasks dargestellt. In der x-Dimension wird die Zeit aufgetragen und in der y-Dimension wird die Nutzung der Ressource dargestellt. Eine einzelne Task ist als ein Rechteck definiert und beginnt an der linken Seite mit der Startzeit \( S \). Sie hat eine Bearbeitungsduer \( D \) und eine Höhe \( R \). In dem Profil des Constraints cumulative aller Tasks darf das Ressourcenlimit von keiner Task überschritten werden.


Definition GC1 Cumulative-Constraint [24]:

Gegeben sei \( i \) der Index für die folgenden Listen \( L_k \) und \( n \) sei für alle Listen gleich groß, \( L_{ki}, \forall k : 1 \leq i \leq n \) dann sei: \( S = [S_1, \ldots, S_n] \) die Liste der Startzeiten, \( D = [D_1, \ldots, D_n] \) die Liste der Werte für die
3.6. GLOBALE CONSTRAINTS

Dauer der Bearbeitung, \( R = [R_1, \ldots, R_n] \) die Liste der genutzten Ressourcenmenge, \( E = [E_1, \ldots, E_n] \) die Liste der Endzeiten der Tasks und \( F = [F_1, \ldots, F_n] \) die Liste der Flächen und \( \text{Ress} \): die maximale Größe der zur Verfügung stehenden Ressource, \( \text{End} \): der letzte Endzeitpunkt aller Tasks, \( \text{Mitte} \): der mittlere Wert der Ressourcenutzung, \( \text{Über} \): die Fläche oberhalb des Mittelwerts der Ressourcenutzung mit: \( \min_d(X) \) dem minimalen Wert der Domän X und \( \max_d(X) \) dem maximalen Wert der Domän X. Wenn gilt:

\[
\begin{align*}
    a &= \min(\min_d(S_1), \ldots, \min_d(S_n)), \\
    b &= \max(\max_d(D_1) + \max_d(D_n)), \\
    E: \ E_i \text{ mit von } S_i: \forall i: \ 1 \leq i \leq n, \ S_i + D_i = E_i, \\
    F: \ F_i \text{ mit von } S_i: \forall i: \ 1 \leq i \leq n, \ D_i * R_i = F_i, \\
    \text{Ress}: \forall i: \ 1 \leq i \leq n, \ D_i > 0, \forall i: \ 1 \leq i \leq n, \ R_i > 0, \forall i: \ a \leq i \leq b, \ \text{Ress} = \max(\sum R_j), \forall j: S_j \leq i \leq \text{End} = \max(S_i + D_i), \\
    \text{Über, Mitte}: \forall i: \ a \leq i \leq b, \ \text{Über} = \sum (\max(0, \sum R_j * \text{Mitte})), \forall j: S_j \leq i \leq S_j + D_j - 1,
\end{align*}
\]

\( \text{Cum} \) ein Cumulative-Constraint der Form:

\( \text{Cum} = \text{cumulative}(S,D,R,E,F,\text{Ress},\text{End},[\text{Mitte, Über}]) \).\( ^1 \)

Beispiel GC1-1:
In diesem Beispiel sind die Endzeiten der Tasks (\( E = [E_1, E_2, E_3] \)), die maximale Größe der zur Verfügung stehenden Ressource (\( \text{Ress} \)) und dem letzten Endzeitpunkt aller Tasks (\( \text{End} \)) gegeben.

bspg:11(S,E,Ress,End)-
\( S = [S_1,S_2,S_3], \ S :: 1..5, \)
\( D = [2,3,2], \)
\( R = [2,1,2], \)
\( E = [E_1,E_2,E_3], \ E :: 1..5, \)
\( \text{Ress} :: 1..3, \)
\( \text{End} :: 1..5, \)
\( \text{cumulative}(S,D,R,E,\text{Ress},\text{End},), \)
\( \text{min}(\text{labeling}(S),\text{Ress}). \)

\[
\begin{array}{c}
\text{R, Ress} \\
\text{3} & \text{1} \\
\text{2} & \text{3} \\
\text{1} & \text{2} \\
\end{array}
\]

Abbildung 3.5: Beispiel 1 - GC1: Cumulative - Constraint

\( ^1 \)Mit dem Unterstrich _ wird ein ungenutzter Parameter gekennzeichnet.
Beispiel GC1-2:
In diesem zweiten Beispiel sind zu den Endzeiten der Tasks \( E = [E_1,E_2,E_3] \), der maximale Größe der zur Verfügung stehenden Ressource (Ress), der letzte Endzeitpunkt aller Tasks (End) noch der mittlere Wert der Ressourcenutzung (Mitte) und die Fläche oberhalb des Mittelwerts der Ressourcenutzung (Über) gegeben.

bspgc12(S,E,Ress,End):
\[
S = [S_1,S_2,S_3], \quad S :: 1..5,
D = [2,3,2],
R = [1,1,1],
E = [E_1,E_2,E_3], \quad E :: 1..5,
Ress :: 1..3,
End :: 1..5,
Mitte = 2,
Über :: 1..5,
cumulative(S,D,R,E,Ress,End,[Mitte,Über]),
\]
\[
\text{min}(\text{labeling}(S),\text{Über} + \text{Ress}).
\]

Abb. 3.6: Beispiel 2 - GC1: Cumulativ - Constraint


**Globales Constraint (diffn)**

Mit dem globalen Constraint diffn/1 (nach [24]) kann die Nichtüberlappung von einer Menge von n-dimensionalen Rechtecken im n-dimensionalen Raum modelliert werden. Es werden hierdurch multidimensionale Platzierungsprobleme zum Beispiel in der Produktionsplanung, im Zuschnitt von Rohstoffen und mehrdimensionale Packungsprobleme lösbar. Die Idee des Constraints diffn/1 ist es, das Constraint alldifferent/1, welches die paarweise Verschneidung von Domänen sichert, in ein globales Constraint umzusetzen, welches die Überlappingsfreiheit zwischen einer Menge von Objekten im n-dimensionalen Raum modelliert. Die Eigenschaften des Constraints diffn/1 werden im Folgenden im zwei dimensionalen Raum erläutert. Gegeben ist eine Menge von n Rechtecken. Zu jedem Rechteck ist die linke untere Ecke \((X_i, Y_i)\), die Länge \(L_i\) und die Höhe \(H_i\) bekannt. Für diese n Rechtecke wird gefordert, dass sie sich nicht überlappen dürfen und dass für jedes Paar von Rechtecken \(R_i\) und \(R_j\) eine der vier Bedingungen gelten muss:

- \(R_i\) ist oberhalb \(R_j\) \((Y_j + H_j \leq Y_i)\)
- \(R_i\) ist unterhalb \(R_j\) \((Y_i + H_i \leq Y_j)\)
- \(R_i\) ist links von \(R_j\) \((X_i + L_i \leq X_j)\)
- \(R_i\) ist rechts von \(R_j\) \((X_j + L_j \leq X_i)\)

Das Constraint diffn/1 kann demzufolge durch die vier einfachen Bedingungen spezifiziert werden. Hierbei ergibt sich eine Disjunktion von vier arithmetischen Constraints.

\[
(Y_j + H_j \leq Y_i) \lor (Y_i + H_i \leq Y_j) \lor (X_i + L_i \leq X_j) \lor (X_j + L_j \leq X_i)
\]


Das globale Constraint diffn/1 hat für n Rechtecke im zweidimensionalen Raum die folgende Form:

\[
\text{diffn}(\left[ [X_1, Y_1, L_1, H_1], [X_2, Y_2, L_2, H_2], \ldots, [X_n, Y_n, L_n, H_n] \right])
\]

Das Argument des Constraints beinhaltet eine Liste von Listen, welche die Anwendung auch in höheren Dimensionen erlaubt. Hierfür wird die innerste Liste bei jeder weiteren Dimension jeweils um zwei Argumente (Position und Ausdehnung in der neuen Dimension) erweitert werden.

Definition GC2 Diffn-Constraint [24]:

Gegeben sei i der Index für die Liste \( R \) mit \( 1 \leq i \leq m \) mit n-dimensionalen Rechtecken \( r_i \) als ein Tupel von DomänenvARIABLEN der Form \( (O_{i1}, \ldots, O_{im}, L_{i1}, \ldots, L_{im}) \), wobei \( O_{ij} \) den Ursprung und \( L_{ij} \) die Ausdehnung des n-dimensionalen Rechtecks \( r_i \) in der i-ten Dimension ist und folgende Bedingungen gelten:

- \( \forall i \in [1,m], \forall j \in [1,n]: O_{ij} \) ist eine DomänenvARIABLE oder eine ganze Zahl
- \( \forall i \in [1,m], \forall j \in [1,n]: L_{ij} \) ist eine DomänenvARIABLE oder eine ganze Zahl
- \( \forall i \in [1,m], \forall j \in [1,n]: L_{ij} \neq 0 \)
- \( \forall i \in [1,m], \forall j \in [1,m] \ j \neq i, \exists k \in [1,n] \ \setminus (O_{ik} \geq O_{jk} + L_{jk}) \lor (O_{jk} \geq O_{ik} + L_{ik}) \)

dann sei \( \text{Diffn} \) ein Diffn-Constraint der Form:

\[
\text{Diffn} = \text{diffn}(R) = \text{diffn}([O_{11}, \ldots, O_{1n}, L_{11}, \ldots, L_{1n}], \ldots, [O_{m1}, \ldots, O_{mn}, L_{m1}, \ldots, L_{mn}]).
\]

Beispiel GC2:

In diesem Beispiel sind die drei Rechtecke gegeben, die überlappungsfrei platziert werden sollen. Im nachfolgenden Aufruf der Lösungssuche labeling werden den DomänenvARIABLEN Werte zugeordnet. Ein Ergebnis ist in der Abbildung 3.7 dargestellt.

bspgc21():
O1 = [O_{11}, O_{12}, O_{13}] :: 1,4,
O2 = [O_{21}, O_{22}, O_{23}] :: 1,4,
diffn([[O_{11}, O_{21}, 1, 1], [O_{12}, O_{22}, 3, 2], [O_{13}, O_{23}, 1, 3]]),
labeling([O_{11}, O_{21}, O_{12}, O_{22}, O_{13}, O_{23}]).

Abbildung 3.7: Beispiel 1 - GC2: Diffn - Constraint

Das globale Constraint diffn kann zur Modellierung von vielen Typen von Packungs- und Platzierungsproblemen in zwei- oder mehrdimensionalen Problemräumen eingesetzt werden. Im zweidimensionalen Raum können mit dem Constraint die bekannten Zuschnittprobleme verschiedenster Form gelöst werden. Im dreidimensionalen Raum lässt sich das Packungsproblem von rechteckigen Boxen auf Paletten oder in Containern umsetzen. Im vierdimensionalen Raum kann das dreidimensionale Packungsproblem in Container dahingehend erweitert werden, dass der Container, in welchen die Box gepackt wird, auswählbar ist. Die ersten drei Dimensionen beschreiben die Position der Box im Container und mit der vierten Dimension wird beschrieben, in welchen der verfügbaren Container die Box gelegt wird.

---

\footnote{Es existiert für jedes Paar \( i, j \) \( i \neq j \) eines n-dimensionalen Rechtecks eine Dimension \( k \) in der \( i \) nach \( j \) ist oder \( j \) nach \( i \) ist.}
3.6. GLOBALE CONSTRAINTS


Kapitel 4

Constraintlöser für Multiressourcenprobleme $CS_{fd-MR}$


Für die Verfahren zur Herstellung der arc-consistency werden zum Beispiel für den schlimmsten Fall die Laufzeit mit einer Komplexität von $O(d^n)$ [43] mit $n \in N$ abgeschätzt, mit $c$ der Anzahl der Constraints und $l$ der Größe der Domäne. Diese Verfahren wurden mehrfach verbessert und es entstanden die arc-consistency Algorithmen AC-1 bis AC-7 [56] [59] [41] [9] [10]. Der derzeit beste arc-consistency Algorithmus hat im schlimmsten Fall eine Laufzeit mit einer Komplexität von $O(d^2)$. Der Aufwand hängt von der Anzahl und Größe der Domänen ab. Die Anzahl der Domänen kann meist nicht reduziert werden, weil sie aus der Modellierung des Problems herrihren. Mit einer geeigneten Modellierung kann man viele Domänen und Constraints im Constraintnetz reduzieren. Für die weiteren Betrachtungen wird davon ausgegangen, dass die optimale Modellierung mit der geringsten Anzahl von Domänen und Constraints gefunden wurde.

Eine Methode zur Variation der Domänengröße im Lösungsprozess, durch die keine Lösung verloren geht, ist die backtrackbare Domänenreduzierung [34]. Bei der backtrackbaren Domänenreduzierung werden nur einige Werte in der Domäne belassen und der Rest der Domäne wird abgeschnitten. Für den Fall, dass keine Lösung gefunden werden kann, wird der abgeschnittene Teil der Domäne aktiviert und der Lösungsprozess wird durch die Reduzierung an einer anderen Stelle der Domäne fortgesetzt. Der Reduktionsprozess wird solange iteriert, bis die Domänengröße einen vorgegebenen kleinsten Wert unterschreitet.

In dieser Arbeit sollen nun zwei Möglichkeiten der Domänenreduktion vorgestellt werden. Bei der Methode der Domänenzerlegung erfolgt nach der Erzeugung des Constraintnetzes eine Auswertung der


Die Zusammenfassung der Methode der Domänenzerlegung, des Ressourcenconstraint für n-dimensionale Räume und den notwendigen Grundlagen für die Domänen und einigen Basisconstraints, wie das Gleichheitsconstraint, werden in einem Constraintssystem $CS_{FD-MR}$ mit vollständiger Suche (Backtracking) zusammengefasst (siehe Abbildung 4.1). Die dargestellten Komponenten des Constraintsolvers $CS_{FD-MR}$ wurden im Rahmen dieser Arbeit umgesetzt.

Abbildung 4.1: Komponenten des Constraintsolvers $CS_{FD-MR}$
Das Constraintsystem \( CS_{f,d-MR} \):

\[
CS_{f,d-MR} = (C_{f,d-MR}, D_{f,d-MR}, X_{f,d-MR}) \text{ mit}
\]

- \( C_{f,d-MR} \subseteq \{0, 1, \ldots, =, \neq, >, <, \geq, \leq, \text{notin}, \text{diff2D}, \text{RC1}, \text{RC2} \} \),
- \( D_{f,d-MR} = N \),
- \( X_{f,d-MR} \subseteq \text{Var}_f \) den Variablen, deren Wertebereiche endlich, ganzzahlige Intervalle sind.

Im Folgenden werden die grundlegenden Begriffe und Definitionen eingeführt.

### 4.1 Ressourcen und Dimensionen

**Definition R1 Ressourcendimension:**

Eine Ressourcendimension \( RD_i \) mit \( i \in N \) der Anzahl der Ressourcendimensionen beinhaltet eine bestimmte Anzahl von Ressourcen \( R \) mit \( R = R_1, \ldots, R_m \) mit \( m \in N \). Die Anzahl der Ressourcen kann für jede Ressourcendimension \( RD_i \) unterschiedlich sein. Mehrere Ressourcendimensionen \( RD_i \) bilden dann eine Ressourcensammlung \( RS \) mit:

\[
RS = RD_1(R_{1,1}, \ldots, R_{m,1}), RD_2(R_{1,2}, \ldots, R_{m,2}), \ldots, RD_n(R_{1,n}, \ldots, R_{m,n}) \text{ mit } m,n \in N.
\]

![Abbildung 4.2: N - Ressourcendimensionen](image)

**Definition R2 Ereignis:**

Ein Ereignis \( E \) benötigt aus jeder Ressourcendimension \( RD_i \) eine Ressource \( R_i \) für eine Dauer \( D_i \) mit \( i \in \text{Ressourcendimensionen} \):

\[
E = ((R_1, D_1), (R_2, D_2), \ldots, (R_n, D_n)) \text{ mit } D_1 = D_2 = \ldots = D_n, n \in N.
\]

Das Ereignis \( E \) ist ein Vorgang, der in den einzelnen Ressourcendimension \( RD_i(\text{RD}_1 = RD_i) \) mit \( i \in \text{Ressourcendimensionen} \) zu einem bestimmten Startzeitpunkt \( ZS_i \) beginnt, eine bestimmte Dauer \( D_i \) andauert und zu einem bestimmten Endzeitpunkt \( ZE_i \) endet, mit:

\[
ZS_i + D_i = ZE_i, i \in N
\]
Für jedes Ereignis $E_j$ mit $j \in N$ können der Startzeitpunkt $ZS_i$, der Endzeitpunkt $ZE_i$, die Dauer $D_i$ und die Ressourcen $R_i$ mit $i \in N$ aufgespannt werden, die Lösungen des Problems liegen können.

$S = RD_1 \times ... \times RD_n$ mit $n \in N$

Definition R3 Suchraum:
Der Suchraum $S$ umfasst den gesamten Raum, der durch die Ressourcendimensionen $RD_i$ mit $i \in N$ aufgespannt wird, in dem die Lösungen des Problems liegen können.

Definition R4 Teilsuchraum:
Für $i \in N$ ist ein Teilstück des gesamten Suchraums $S$.

$\forall i: ST_i \subseteq S$ mit $i \in N$

Definition R5 zerlegter Teilsuchraum:
Für $i \in N$ ist ein Teil des gesamten Suchraums $S$ und der zerlegte Teilsuchraum $zST_j$ kann gleich oder kleiner als ein Teilsuchraum $ST_i$ mit $i \in N$ sein.

$\forall j: zST_j \subseteq ST_i$ mit $i, j \in N$

In der Abbildung 5.1 sind die Definitionen der unterschiedlich großen Suchräume visualisiert.

Abbildung 4.3: Suchraum $S$, Teilsuchraum $ST$ und zerlegter Teilsuchraum $zST$

Definition R6 Kapazität:
Die Anzahl der Werte $WD_i$ einer Ressourcendomäne $D_i$ mit $i \in N$ der Anzahl der Ressourcendomänen in einem festgelegten Suchraum wird als Kapazität $KD_i$ des festgelegten Suchraumes bezeichnet.

Bei dem festgelegten Suchraum kann es sich um den gesamten Suchraum $S$, einen Teilsuchraum $ST$ oder einen zerlegten Teilsuchraum $zST$ handeln. Im Folgenden wird die Kapazität $KD$ für den zerlegten Teilsuchraum $zST$ ermittelt.

$\forall i: KD_i = \sum WD_i$ mit $i \in N$

4.2 Domäne und Constraints

4.2 DOMÄNE UND CONSTRAINTS

4.2.1 Domäne


Definition D1 Domäne:
Eine Domäne D sei eine aufsteigende nach den Domänenwerten geordnete Liste von Paaren \((A_n, E_n)\), mit \(A_n\) dem Anfang und \(E_n\) dem Ende eines lückenlosen Intervalls von natürlichen Zahlen und mit der Form \(D((A_1, E_1), (A_2, E_2), \ldots, (A_n, E_n))\) mit \(n \in \mathbb{N}, A_n, E_n \in \mathbb{N}\).

Die einelementige Domäne D ist definiert mit \(D((A_n, E_n))\) mit \(A_n = E_n\) und (alle Elemente vorhanden) zum Beispiel \(D([1, 1])\).

Die Domäne D wird definiert mit dem Konstruktor :: derart, dass der Anfangswert AW und der Endwert EW der Domäne D mit dem Konstruktor ::, in der Form D :: AW ... EW oder durch die einzelnen Werte \(W_n\) mit \(n \in \mathbb{N}\) in der Form D :: \([W_1, W_2, \ldots, W_n]\), verküpft werden.

\[
\begin{align*}
D &:: AW \ldots EW = D(AW, EW) \\
D &:: [W_1, \ldots, W_n] = D([W_1, W_n]) \text{ mit } n \in \mathbb{N}
\end{align*}
\]

Beispiele D1: \(D :: 1..5 = D([1, 5])\) oder \(D :: [1, 2, 3] = D([1, 3])\).

Auf diesem neuen Domäntenotyp können Operationen ausgeführt werden. Diese Operationen ermöglichen ein Verkleinern (Einschränken), ein Splitten, ein Vergrößern (Erweitern) und ein Zusammenfügen der Domänen.

Definition D2 Operationen auf der Domäne:
Auf der Domäne D gibt es die folgenden Operationen +, -, ---R und - mit \(A_x, E_x, S_x, x \in \{j, k, l, m, n, o, p, q, r\}; j, k, l, m, n, o, p, q, r \in \mathbb{N}\) und \(j < k < l < m < n < o < p < q < r\):

- Hinzufügen + mit

1. (a) \(D((A_1, E_1), \ldots, (A_k, E_k)) = D((A_m, E_m), \ldots, (A_o, E_p)) = D((A_1, E_1), \ldots, (A_k, E_k), (A_m, E_m), \ldots, (A_o, E_p))\), mit \(A_k < E_l < A_m < E_n, E_{l+1} = A_m\), (Zusammenfassung)

2. (a1) \(D((A_m, E_m), \ldots, (A_o, E_p)) = D((A_1, E_1), \ldots, (A_k, E_k))\), mit \(A_k < E_l < A_m < E_n, E_{l+1} = A_m\), (Symmetrie)

3. (b) \(D((A_1, E_1), \ldots, (A_k, E_k)) = D((A_m, E_m), \ldots, (A_o, E_p))\), mit \(A_k < E_l < A_m < E_n, E_{l+1} = A_m\), (Anhängung)

4. (b1) \(D((A_m, E_m), \ldots, (A_o, E_p)) = D((A_1, E_1), \ldots, (A_k, E_k))\), mit \(A_k < E_l < A_m < E_n, E_{l+1} = A_m\), (Symmetrie)

5. (c) \(D((A_1, E_1), \ldots, (A_k, E_k)) + E_m = D((A_1, E_1), \ldots, (A_k, E_k), E_m)\), mit \(A_k < E_l < E_m, E_{l+1} = E_m\), (Erweiterung am Ende - Zusammenfassung)
6. (c1) \( D((A_1, E_j), ..., (A_k, E_l)), ..., (A_k, E_l), ..., (A_k, E_l), (A_m, E_m)) \), mit \( A_k < E_l < E_m, E_{l+1} < E_m \) (Erweiterung am Ende - Anhänge)

7. (d) \( D((A_1, E_m), ..., (A_o, E_p)) + A_k = D((A_k, E_m), ..., (A_o, E_p)) \), mit \( A_k < A_l < E_m, A_{k+1} = A_l \) (Erweiterung am Anfang - Zusammenfassung)

8. (d1) \( D((A_1, E_m), ..., (A_o, E_p)) + A_k = D((A_k, E_m), (A_1, E_m), ..., (A_o, E_p)) \), mit \( A_k < A_l < E_m, A_{k+1} = A_l \) (Erweiterung am Anfang - Zusammenfassung)

9. (e) \( + D((A_1, E_j), ..., (A_k, E_l), (A_m, E_n), ..., (A_o, E_p)) = D((A_1, E_j), ..., (A_k, E_l), ...) \), mit \( A_k < E_l < A_m < E_n, E_{l+1} = A_m \) (Zusammenfassung auf einer Domäne)

- Splitten —, —R mit

1. (a) \( D((A_1, E_j), ..., (A_k, E_l), ..., (A_k, E_l), ..., (A_k, E_l), (A_{l+1}, E_n)) \) mit \( A_k < A_l, E_l < E_n \) (Splitting - einelementig)

2. (b) \( D((A_1, E_j), ..., (A_k, E_n), ..., (A_r, E_m)) = D((A_1, E_j), ..., (A_k, E_{l-1}), (A_{l+1}, E_n), ..., (A_o, E_p)) \), mit \( A_k < A_l, E_m < E_n, A_l < E_q < A_r < E_m, A_k < E_{l-1}, A_{l+1} < E_n \) (Splitting - mehrelementig)

3. (e) \( D((A_1, E_j), ..., (A_k, E_m), ..., (A_o, E_p)) \) --- \( S_l = D((A_1, E_j), ..., (A_k, E_{l-1}), (A_{l+1}, E_m), ..., (A_o, E_p)) \), mit \( A_k < S_l < E_m, A_k = E_{l-1}, A_{l+1} < E_m \) (Splitting durch Propagation)

4. (e) \( D((A_1, E_j), ..., (A_k, E_m), ..., (A_o, E_p)) \) ---R \( S_l = D((A_1, E_j), ..., (A_k, E_l), D((A_{l+1}, E_m), ..., (A_o, E_p)) \), mit \( A_k < S_l < E_m, A_k < E_l, A_{l+1} < E_m \) (Splitting mit Rest)

- Entfernen - mit

1. (c) \( D((A_1, E_j), ..., (A_k, E_l), ..., (A_k, E_{l-1})), \) mit \( A_k < E_l, E_{l-1} < E_l, A_k \leq E_{l-1} \) (Entfernung am Ende)

2. (d) \( D((A_k, E_l), ..., (A_o, E_p)) - A_k = D((A_{k+1}, E_l), ..., (A_o, E_p)) \), mit \( A_k < E_l, A_{k+1} \leq E_l \) (Entfernung am Anfang)

Beispiele D2:
(a) +1: \( D((1, 3), (7, 9) + D((10, 14), (18, 19]) = D((1, 3), (7, 14), 18, 19]) \)
(b) +3: \( D((1, 3), (6, 7)] + D((10, 14), (18, 19]) = D((1, 3), (6, 7), (10, 14), (18, 19]) \)
(c) ---3: \( D((1, 20)) --- 7 = D((1, 6), (8, 20)) \)
(d) ---R(4): \( D((1, 20)) ---R 7 = D((1, 7]), D((8, 20]) \)
(e) ---1: \( D((1, 3), (10, 14]) --- D((12, 12]) = D((1, 3), (10, 11), (13, 14]) \)
(f) ---2: \( D((1, 3), (20, 25]) --- D((22, 24), (26, 26]) = D((1, 3), (20, 21), (25, 25]) \)
(g) -1: \( D((1, 20)] - 20 = D((1, 19]) \)
(h) -2: \( D((1, 20)] - 1 = D((2, 20]) \)
4.2 DOMÄNE UND CONSTRAINTS

Definition D3 DomänenList:
Die Funktion DomänenList(D) erzeugt aus der Domäne D eine aufsteigend sortierte Domänenliste LD der Domänenelemente:

\[ \text{DomänenList}(D[[A_n, E_n]]) = DL[A_n, A_n + 1, \ldots, E_n] \text{ mit } A_n, E_n, n \in N, A_n \leq E_n. \]

Beispiele D3:
(a) DomänenList(D[[2, 5]]) = DL[2, 3, 4, 5],
(b) DomänenList(D[[2, 5], (7, 10)]) = DL[2, 3, 4, 5, 7, 8, 9, 10]

4.2.2 Domänenzerlegung


Beispiel D4:
Gegeben ist eine Domäne D mit 100 Werten, die in Teildomänen \( D_1 \) bis \( D_4 \) mit je 25 Werten aufgeteilt werden soll.

\[
\begin{align*}
1: & \quad D[[1, 100]] \quad \text{R 25} = D_1[[1, 25]], D_{12}[[26, 100]] \\
2: & \quad D_{12}[[26, 100]] \quad \text{R 50} = D_2[[26, 50]], D_{23}[[51, 100]] \\
3: & \quad D_{23}[[51, 100]] \quad \text{R 75} = D_3[[51, 75]], D_4[[76, 100]]
\end{align*}
\]

Es ergeben sich dann aus der Domäne \( D[[1, 100]] \) die vier Domänen:
\( D_1[[1, 25]], D_2[[26, 50]], D_3[[51, 75]], D_4[[76, 100]]. \)

4.2.3 Nichtzeitenhierarchie

Bei vielen Ressourcen- und Platzierungsproblemen stehen von anfang an viele ungültige Werte von Domänen fest.


Mit dem Konzept der Nichtzeitenhierarchie werden die Nichtzeiten, die auf verschiedenen Modellierungsstufen vorliegen, zur untersten Ebene durchgereicht und dort zusammengefasst. Dies erfolgt mit einer Transformation, die die Listen der Nichtzeiten der einzelnen Modellierungsstufen in einem Vorverarbeitungsschritt zu einer Liste zusammenfasst.
Definition D4 Nichtzeitenliste:
Die Nichtzeitenliste $N_{i,p}$ mit $i \in \mathbb{N}$ der Anzahl der Nichtzeitenlisten und $p \in \mathbb{N}$ den Varianten der Pläne enthält die Nichtzeitpunkte und Nichtzeitintervalle $NZ$, die für die Planung nicht benutzt werden dürfen.

Definition D5 Nichtzeitenhierarchie:
Eine Nichtzeitenhierarchie $NH$ besteht aus Nichtzeitenlisten $N_{i,p}$ mit $i \in \mathbb{E}$ den Ebenen der Hierarchie und $p \in \mathbb{P}$ den verschiedenen Plänen. Dabei ist $N_{0,p}$ die Nichtzeitenliste des gesamten Plans $P$, $N_{1,p}$ die Nichtzeitenliste der Hierarchiestufe 1 des Plans $p$, $N_{2,p}$ die Nichtzeitenliste der Hierarchiestufe 2 des Plans $p$ und $N_{n,p}$ die Nichtzeitenliste der Hierarchiestufe $n$ des Plans $p$:

$$NH = N_{1,p}, \ldots, N_{n,p} \text{ mit } n \in \mathbb{E}, \ p \in \mathbb{P}.$$  

Definition D6 Domänenumwandlung:
Die Umwandlung einer Domäne $D_i$ in eine Domäne $D_{i+1}$ mit $i \in \mathbb{E}$ den Ebenen erfolgt nach den Hierarchiestufen. Eine Domäne $D_i$ wird unter Entfernung der Nichtzeiten $NZ$ aus den Nichtzeitenlisten $N_{i,p}$ in die Domäne $D_{i+1}$ überführt:

$$D_i \rightarrow D_{i+1} \ \backslash \ N_{i,p} \text{ mit } i \in \mathbb{E}, \ p \in \mathbb{P}.$$  

Die Einzelschritte haben dann folgende Form:

- \(1\) $D_0 \rightarrow D_1 \ \backslash \ N_{0,p}$
- \(2\) $D_1 \rightarrow D_2 \ \backslash \ N_{1,p}$
- \(3\) $D_2 \rightarrow D_3 \ \backslash \ N_{2,p}$
- \(\ldots\)
- \(n\) $D_n \rightarrow D_{n+1} \ \backslash \ N_{n,p}$

Beispiel D6:

$D_0[(1,20), N_{G,p}[(2,2),(6,6),(10,10),(14,14),(18,18)], N_{K,p}[(3,3),(5,5)], N_{V,p}[(7,7)], N_{P,p}[(15,17)],$

- \(1\) $D_0 \rightarrow D_1 \ \backslash \ N_{G,p}$
  $D_0[(1,20)] - N_{G,p}[(2,2),(6,6),(10,10),(14,14),(18,18)] \rightarrow D_1[(1,1),(3,3),(7,7),(11,13),(15,17)]$

- \(2\) $D_1 \rightarrow D_2 \ \backslash \ N_{K,p}$
  $D_1[(1,1),(3,3),(7,7),(11,13),(15,17)] - N_{K,p}[(3,3),(5,5)] \rightarrow D_2[(1,1),(4,4),(7,7),(11,13),(15,17)]$

- \(3\) $D_2 \rightarrow D_3 \ \backslash \ N_{P,p}$
  $D_2[(1,1),(4,4),(7,7),(11,13),(15,17)] - N_{P,p}[(7,7)] \rightarrow D_3[(1,1),(4,4),(8,9),(11,13),(15,17)]$

- \(4\) $D_3 \rightarrow D_4 \ \backslash \ N_{P,p}$
  $D_3[(1,1),(4,4),(8,9),(11,13),(15,17)] - N_{P,p}[(15,17)] \rightarrow D_4[(1,1),(4,4),(8,9),(11,13)]$
4.2.4 Constraints

In den folgenden Definitionen der Constraints und der globalen Constraints sind die Funktionalitäten des neuen Domäentyps aus der Definition D1 zu Grunde gelegt worden mit \( A_x, E_x, x = \{ i, j, k, l, m, n, o, p \}; i, j, k, l, m, n, o, p \in N \) und \( i < j < k < l < m < n < o < p \).

Definition C1 \(-\) Constraint:
Das Constraint \(-\) fordert die Gleichheit zweier Domänen \( D_1 \) und \( D_2 \):
\[
\forall i, j, m, n : D_1[\ldots, (A_i, E_j), \ldots] = D_2[\ldots, (A_m, E_n), \ldots]
\]
mit \( A_i \equiv A_m, E_j \equiv E_n \).
Beispiel C1: \( D_1[(2, 5)] = D_2[(2, 5)] \)

Definition C2 \( \neq \) Constraint:
Das Constraint \( \neq \) fordert die Ungleichheit zweier Domänen \( D_1 \) und \( D_2 \):
\[
\exists i, j, m, n : D_1[\ldots, (A_i, E_j), \ldots] \neq D_2[\ldots, (A_m, E_n), \ldots]
\]
mit \( A_i \neq A_m, E_j \neq E_n \).
Beispiel C2: \( D_1[(2, 5)] \neq D_2[(8, 10)] \)

Definition C3 \( < \) Constraint:
Das Constraint \( < \) fordert für zwei Domänen \( D_1 \) und \( D_2 \) die Bedingung, dass \( D_1 \) kleiner als \( D_2 \) zu sein hat:
\[
D_1[(A_i, E_j), (A_k, E_l)] < D_2[(A_m, E_n), (A_o, E_p)]
\]
mit \( A_i < E_j < A_k < E_l, A_m < E_n < A_o < E_p \) und \( E_l < A_m \).
Beispiel C3: \( D_1[(2, 5), (7, 10)] < D_2[(12, 16), (18, 20)] \)

Definition C4 \( > \) Constraint:
Das Constraint \( > \) fordert für zwei Domänen \( D_1 \) und \( D_2 \) die Bedingung, dass \( D_1 \) größer als \( D_2 \) zu sein hat:
\[
D_1[(A_i, E_j), (A_k, E_l)] > D_2[(A_m, E_n), (A_o, E_p)]
\]
mit \( A_i < E_j < A_k < E_l, A_m < E_n < A_o < E_p \) und \( A_i > E_p \).
Beispiel C4: \( D_1[(12, 15), (17, 20)] > D_2[(2, 6), (8, 10)] \)

Definition C5 \( \leq \) Constraint:
Das Constraint \( \leq \) fordert für zwei Domänen \( D_1 \) und \( D_2 \) die Bedingung, dass \( D_1 \) gleich oder kleiner als \( D_2 \) zu sein hat:
\[
D_1[(A_i, E_j), (A_k, E_l)] \leq D_2[(A_m, E_n), (A_o, E_p)]
\]
mit \( A_i < E_j < A_k < E_l, A_m < E_n < A_o < E_p \) und \( (E_l < A_m) \lor (A_i = A_m, E_j = E_n, A_k = A_o, E_l = E_p) \).
Beispiel C5: \( D_1[(2, 5), (7, 10)] \leq D_2[(12, 16), (18, 20)] \)
\[
D_1[(2, 5), (7, 10)] = D_2[(2, 5), (7, 10)]
\]

Definition C6 \( \geq \) Constraint:
Das Constraint \( \geq \) fordert für zwei Domänen \( D_1 \) und \( D_2 \) die Bedingung, dass \( D_1 \) gleich oder größer als \( D_2 \) zu sein hat:
\[
D_1[(A_i, E_j), (A_k, E_l)] \geq D_2[(A_m, E_n), (A_o, E_p)]
\]
mit $A_i < E_j < A_k < E_l$, $A_m < E_n < A_o < E_p$ und $(A_i > E_p) \lor (A_i = A_m, E_j = E_n, A_k = A_o, E_l = E_p)$.

Beispiel C6: $D_1([12, 15], (17, 20]) > D_2([2, 6], (8, 10])$
$D_1([2, 5], (7, 10]) = D_2([2, 5], (7, 10])$

Definition C7 Notin-Constraint

Das Constraint Notin habe die Form $\text{notin}(D, \text{Von}, \text{Bis})$ und fordert für die Domäne $D$, dass die Domäne $D$ alle Wert zwischen den Domänenwerten $A_k$ ($\text{Von}$) und $E_l$ ($\text{Bis}$) nicht enthält (entfernt), wenn diese in der Domäne $D$ existieren sollten:

$$D[\ldots, (A_i, E_j), (A_k, E_l), (A_m, E_n), \ldots]$$
$$D[\ldots, (A_i, E_j), (A_k, E_l), (A_m, E_n), \ldots]$$

mit $A_i < E_j < A_k < E_l < A_m < E_n$ und wenn $\exists A_k, E_l$ in $D$.

Beispiel C7: $D([2, 5], (8, 10], (17, 20])$, $\text{notin}(D, 8, 10], D([2, 5], (17, 20])$
$D([2, 5], (8, 10], (17, 20])$, $\text{notin}(D, 7, 15], D([2, 5], (17, 20])$

4.3 Globale Constraints

4.3.1 Ressourcen-Constraint


4.3. GLOBALE CONSTRAINTS

Domänen erfordert, wird die Sicherung der Konsistenz durch die Abfrage aller zentralen Domänen mit einer nachfolgenden Intersektionsbildung gewährleistet. Mit Hilfe der Intersektionsmethode werden dem Suchprozess nur Werte angeboten, die in allen zentralen Domänen frei sind und die Zusatzbedingungen werden danach wieder im Suchprozess getestet. Bei der Betrachtung mehrerer zentraler Domänen kann es folgende Fälle geben:

- a) Ein Körper benötigt in zwei zentralen Domänen einen gleichen Wert, dann muss dieser Wert in beiden zentralen Domänen frei sein

- b) Zwei oder mehrere Körper sollen den gleichen Wert haben (Gleichheitsconstraint), dann wird für die zwei oder mehreren Körper geprüft, ob so ein Wert in allen betreffenden zentralen Domänen vorhanden ist. Nur wenn dies der Fall ist, wird der ermittelte Wert als Position für die Platzierung der Körper verwendet.


Dieser Ansatz lässt sich für Probleme einsetzen, bei denen die Überlappingsfreiheit ohne oder mit Gleichheitsconstraints und Reihenfolgeconstraints zu sichern ist und für Probleme, bei denen Domänen mit sehr großen Werten auftreten und somit in der Regel keine Propagation im Constraintnetz durch die Constraints auftritt.

Ressourcen-Constraint diffn2D


Die allgemeinen Eigenschaften des Constraints diffnD/1 werden im Folgenden im zwei dimensionalen Raum erläutert. Gegeben ist eine Menge von n Körpern. Zu jedem Körper, in diesem zweidimensionalen Fall sind es Rechtecke, ist die linke untere Ecke \((X_i, Y_i)\), die Länge \(L_i\) und die Höhe \(H_i\) bekannt. Für diese Rechtecke wird gefordert, dass sie sich nicht überlappen dürfen und dass für jedes Paar von Rechtecken \(R_i\) und \(R_j\) eine der vier Bedingungen gelten muss: \(R_i\) ist oberhalb \(R_j\) \((Y_j + H_j \leq Y_i)\), \(R_i\) ist unterhalb \(R_j\) \((Y_i + H_i \leq Y_j)\), \(R_i\) ist links von \(R_j\) \((X_i + L_i \leq X_j)\) oder \(R_i\) ist rechts von \(R_j\) \((X_j + L_j \leq X_i)\). Das Constraint diffnD/1 kann demzufolge durch die vier einfachen Bedingungen allgemein spezifiziert werden. Hierbei ergibt sich eine Disjunktion von vier arithmetischen Constraints: \((Y_j + H_j \leq Y_i) \lor (Y_i + H_i \leq Y_j) \lor (X_i + L_i \leq X_j) \lor (X_j + L_j \leq X_i)\).

Definition GCR1 diffnD / diffn2D-Constraint:

Gegeben sei i der Index für die folgende Liste \( LRe \) mit \( LRe = [Re_1, \ldots, Re_m] \) mit \( Re_i \), \( 1 \leq i \leq m \); \( m \in \mathbb{N} \) und \( m \) sei die Anzahl der \( n \)-dimensionalen Körper, dann seien \( n \)-dimensionale Körper \( Re_i \) gegeben durch ein Tupel von Domänenvariablen der Form \( (O_{i1}, \ldots, O_{in}, V_{1i}, \ldots, V_{ni}) \), wobei \( O_{ij} \) der Ursprung und \( V_{ij} \) die Ausdehnung des \( n \)-dimensionalen Körpers in der \( i \)-ten Dimension ist. Außerdem sei \( LD_{\text{max}} \) die Liste der maximalen Ausdehnungen der Dimensionen \( D_{\text{max}} \) mit der Form: \( LD_{\text{max}} = [D_{1 \text{max}}, \ldots, D_{n \text{max}}] \) mit \( D_{i \text{max}} \); \( 1 \leq i \leq n \); \( n \in \mathbb{N} \) und \( n \) sei die Anzahl der Dimensionen und \( LSp \) die Liste der \( n \)-dimensionalen gesperrten Bereiche \( Sp_k \) mit der Form: \( LSp = \{Sp_1, \ldots, Sp_o\} \) mit \( Sp_k \); \( 1 \leq k \leq o \); \( o \in \mathbb{N} \) und \( o \) sei die Anzahl der \( n \)-dimensionalen gesperrten Bereiche, dann seien \( n \)-dimensionale gesperrte Bereiche \( Sp_k \) gegeben durch ein Tupel von Domänenvariablen der Form \( (SO_{k1}, \ldots, SO_{kn}, SV_{1k}, \ldots, SV_{nk}) \), mit \( SO_{kl} \) dem Ursprung, \( SV_{kl} \) der Ausdehnung der \( n \)-dimensionalen gesperrten Bereiche in der \( k \)-ten Dimension, wobei folgende Bedingungen gelten:

- \( \forall i \in [1, m], \forall j \in [1, n]: O_{ij} \) ist eine Domännvariable oder eine ganze Zahl
- \( \forall i \in [1, m], \forall j \in [1, n]: V_{ij} \) ist eine Domännvariable oder eine ganze Zahl
- \( \forall k \in [1, o], \forall l \in [1, n]: SO_{kl} \) ist eine Domännvariable oder eine ganze Zahl
- \( \forall k \in [1, o], \forall l \in [1, n]: SV_{kl} \) ist eine Domännvariable oder eine ganze Zahl
- \( \forall i \in [1, n]: D_{i \text{max}} \) ist eine ganze Zahl
- \( \forall i \in [1, m], \forall j \in [1, n]: V_{ij} \neq 0 \)
- \( \forall k \in [1, o], \forall l \in [1, n]: SV_{kl} \neq 0 \)
- \( \forall i \in [1, m], \forall j \in [1, n], j \neq i, \exists p \in [1, n] \setminus \{O_{ip}, V_{jp}\} \wedge (O_{ip} \geq O_{jp} + V_{jp}) \vee (O_{jp} \geq O_{ip} + V_{ip}) \) \(^1\)
- \( \forall k \in [1, o], \forall l \in [1, n], k \neq l, \exists q \in [1, n] \setminus \{SO_{kq}, SV_{iq}\} \wedge (SO_{kq} \geq SO_{iq} + SV_{iq}) \vee (SO_{iq} \geq SO_{kq} + SV_{kq}) \) \(^2\)

dann sei \( \text{DiffnD} \) ein \( \text{diffnD-Constraint} \) der Form:

\[
\text{DiffnD} = \text{diffnD}(LRe, LD_{\text{max}}, LSp) =
\]

\[
\text{diffnD}([O_{11}, \ldots, O_{1n}, V_{11}, \ldots, V_{1n}], \ldots, [O_{mn}, \ldots, O_{m1}, V_{m1}, \ldots, V_{mn}], [D_{1 \text{max}}, \ldots, D_{n \text{max}}], [SO_{11}, \ldots, SO_{1n}, SV_{11}, \ldots, SV_{1n}], \ldots, [SO_{on}, \ldots, SO_{11}, SV_{11}, \ldots, SV_{1n}], \ldots, [SO_{on}, \ldots, SO_{on}, SV_{on}, \ldots, SV_{on}]) \mod m, n, o \in \mathbb{N}
\]

und es sei \( \text{Diff2D} \) ein zweidimensionales \( \text{diffnD-Constraint} \) (\( \text{diffn2D-Constraint} \)) mit \( n \in \{1, 2\} \):

\[
\text{Diff2D} = \text{diffn2D}(LRe, LD_{\text{max}}, LSp) =
\]

\[
\text{diffnD}([O_{11}, \ldots, O_{1n}, V_{11}, \ldots, V_{1n}], \ldots, [O_{mn}, \ldots, O_{m1}, V_{m1}, \ldots, V_{mn}], [D_{1 \text{max}}, \ldots, D_{n \text{max}}], [SO_{11}, \ldots, SO_{1n}, SV_{11}, \ldots, SV_{1n}], \ldots, [SO_{on}, \ldots, SO_{on}, SV_{on}, \ldots, SV_{on}]) \mod n \in \{1, 2\}, m, o \in \mathbb{N}
\]

Beispiel GCR1:


\(^1\)Es existiert für jedes Paar \( i, j \) \((i \neq j)\) eines \( n \)-dimensionalen Körpers eine Dimension \( k \) in der \( i \)-nach \( j \) ist oder \( j \)-nach \( i \) ist.

\(^2\)Es existiert für jedes Paar \( k, l \) \((k \neq l)\) eines \( n \)-dimensionalen gesperrten Bereichs eine Dimension \( q \) in der \( k \)-nach \( q \) ist oder \( q \)-nach \( k \) ist.
4.3. GLOBALE CONSTRAINTS

bspgcr1():
\[O_1 = [O_{11}, O_{12}, O_{13}] \colon 1 \ldots 7,\]
\[O_2 = [O_{21}, O_{22}, O_{23}] \colon 1 \ldots 6,\]
diff2D([[O_{11}, O_{21}, 1, 1], [O_{12}, O_{22}, 3, 2], [O_{13}, O_{23}, 1, 3]], 7, 6, [[6,4,2,2]]),
labeling([O_{11}, O_{21}, O_{12}, O_{22}, O_{13}, O_{23}]).

Abbildung 4.4: Beispiel GCR1: diff2D - Constraint

Das globale Ressourcen-Constraint \texttt{diff2D} setzt die Repräsentation eines zweidimensionalen Lösungsraums durch ein zweidimensionales Feld um. Diese Repräsentation ermöglicht durch mehrfache Verwendung von zweidimensionalen Lösungsräumen die Abbildung von \(n\)-dimensionalen Räumen (Beweis siehe Kapitel 6.1.2 "Beweis des Verfahrens zur Darstellung \(n\)-dimensionaler Körper in (n-1)-zweidimensionalen Räumen"). Somit können \(n\)-dimensionale Körper in einem \(n\)-dimensionalen positiven Raum positioniert werden und das Entstehen von Überschneidungen der \(n\)-dimensionalen Körper vermieden werden. Dieser Raum ist definiert durch seine Dimension \(R^n\), \(n \in \mathbb{N}\). Für jede Position dieses Raumes ist ein Funktionswert \(f(O^n)\) definiert. Das globale Constraint wurde in C++ implementiert. Für die Domänenrepräsentation wird der Datentyp short verwendet. Die Körper werden durch einen Ortsvektor \(O^n\) und einen aufspannenden Vektor \(V^n\) definiert. Die im Anhang "Funktionen des globalen Ressourcen-Constraints \texttt{diff2D}\" aufgeführten Funktionen können für ein globales RessourcenConstraint \texttt{diff2D} ausgeführt werden. Mit Hilfe der Funktionen wird eine dynamische Planung mit Einplanen, Verschieben und Ausplanen realisiert.

Das Constraint \texttt{diff2D} unterstützt direkt einen zweidimensionalen Raum, wobei sich daraus mehrdimensionale Räume modellieren lassen. Um dies zu erleichtern, stehen spezielle Funktionalitäten wie das Bilden von raumübergreifenden Schnittmengen zur Verfügung. Das ist zwar mit einem Mehraufwand bei der Entwicklung der entsprechenden Applikation verbunden, hat aber den Vorteil, dass die Anwendung später performanter und leichter zu skalieren ist. Bildlich lässt sich das globale Constraint \texttt{diff2D} anhand eines zweidimensionalen Feldes beschreiben, in das verschiedene Rechtecke platziert werden. Dabei können Bedingungen definiert werden, die die Operationen auf dem Lösungsraum einschränken. Der Funktionswert \(f(x, y), x, y \in \mathbb{N}\) wird Status genannt. In Abhängigkeit vom Statuswert können somit verschiedene Operationen zugelassen oder untersagt werden und verschiedene andere Eigenschaften definiert werden. Rechtecke werden definiert durch einen Ortsvektor \(O = (o_1, o_2)\) und einen aufspannenden Vektor \(V = (v_1, v_2)\). So kann man zwar Rechtecke mit einem Status von \(f(x, y) \leq -1\) einfügen, aber nicht wieder löschen, verschieben oder einen anderen vertauschen. Diese eignen sich besonders für konstante Einschränkungen (Sperrflächen). Dagegen sind Rechtecke mit einem Status von \(f(x, y) \geq 0\) flexibler und können hinzugefügt, gelöscht, verschoben und vertauscht werden. So ist zum Beispiel der Fall denkbar, dass positive Statuswerte als Indizierung eines externen Containertyps dienen. Dies ermöglicht das Hinterlegen von zusätzlichen Sta-
4.3.2 Reihenfolge-Constraint

Abbildung der Reihenfolgen


Das Reihenfolge-Constraint dient zur Festlegung der Reihenfolge von Ereignissen \( E_i, i \in \text{in eine Ressourcendimension } RD_i \). Vorzugsweise wird es die Ressourcendimension der Zeit sein (RD).

Eine Reihenfolgebeziehung von Ereignissen \( E_i \) mit den derzeit zur Verfügung stehenden Constraints <, <=, > oder >= wie folgt ausgedrückt: (1): \( E_1 < E_2 < ... < E_i < ... < E_n \), (2): \( E_1 <= E_2 <= ... <= E_i <= ... <= E_n \), (3): \( E_1 > E_2 > ... > E_i > ... > E_n \) oder (4): \( E_1 >= E_2 >= ... >= E_i >= ... >= E_n \) mit \( n \in N \).

Die Verknüpfung der Ereignisse \( E_i \) erfolgt in den derzeitigen Constraintsystemen über die Domänenvariablen. Für den Startzeitpunkt \( ZS_i \), den Endzeitpunkt \( ZE_i \) und die Dauer \( DD_i \) werden jeweils Domänenvariablen \( ZSD_i, ZED_i \) und \( DD_i \) angelegt. Zur Verknüpfung der Ereignisse \( E_i \) und \( E_{i+1} \) reichen die Startzeitpunktdomänen \( ZSD_i, ZSD_{i+1} \) und die Dauerdomänen \( DD_i, DD_{i+1} \) aus. Die Endzeitpunktdomänen \( ZED_i, ZED_{i+1} \) enthalten die redundanten Informationen, die sich aus der Addition der Startzeitpunktdomäne \( ZSD_i / ZSD_{i+1} \) und der Dauerdomäne \( DD_i / DD_{i+1} \): \( ZSD_i + DD_i = ZED_i / ZSD_{i+1} + DD_{i+1} = ZED_{i+1} \) ergeben. Die Reihenfolgebeziehung wird durch die Verknüpfung der drei Startzeitpunktdomänen \( ZSD_i \) und \( ZSD_{i+1} \) und der Dauern \( DD_i \) und \( DD_{i+1} \) der beiden direkt aufeinander folgenden Ereignisse \( E_i \) und \( E_{i+1} \) mit einem Constraint < (aufsteigende Domänenwerte der Ereignisse) oder > (fallende Domänenwerte der Ereignisse) hergestellt.

\[
\begin{align*}
ZSD_i + DD_i &< ZSD_{i+1} + DD_{i+1} \\
ZSD_i + DD_i &> ZSD_{i+1} + DD_{i+1}
\end{align*}
\]

Durch die Verknüpfung mit den Constraints werden die Domänen durch die Propagation des Constraintsystems auf die möglichen Werte eingeschränkt. Durch die Propagation werden die Anfänge und Enden der Domäne um die Dauer der Ereignisse verringert (siehe Bild 4.5).

Abbildung 4.5: Reihenfolgebeziehung von drei Ereignissen mit Domänen


Bei der Definition von Reihenfolgebeziehungen durch den Nutzer kann es vorkommen, dass die Größe des Suchraums nicht für alle Ereignisse in der Reihenfolgebeziehungskette ausreicht. Die Constraintsysteme stellen das nicht im Einzelnen fest. Sie geben nur aus, dass das Problem nicht löschbar ist. Der Nutzer erhält keine Hinweise darauf, was er falsch gemacht hat. Man kann die Größe der Reihenfolgebeziehungskette vorher ausrechnen, aber man kann die Wirkung der anderen Constraints des Constraintnetzes nicht abschätzen. Deshalb weiß man nicht, ob das System wirklich keine Lösung hat oder ob nur die Reihenfolgebeziehungskette zu groß war.


**Reihenfolge-Constraints**

In diesem Abschnitt wird das globale Reihenfolge-Constraint beschrieben. Mit diesem Constraint können drei verschiedene Fälle von Reihenfolgebeziehungen (Ereignistypen) umgesetzt werden.

**Ereignistypen ET1, ET2 und ET3**

Die zu verarbeitenden Ereignisse $E_i$ können mit Zusatzinformationen in drei Ereignistypen ET1, ET2 und ET3 in dem Reihenfolge-Constraint angegeben werden. Zu den Zusatzinformationen gehören die
Anzahl $\text{An} \, z_i$ der Ereignisse $E_i$ im Teilsuchraum und die parallelen Ereignisse $EP_i$, die parallel zum Ereignis $E_i$ stattfinden. Sie werden in einer Liste $LP_i$ zusammengefasst.

- In dem ersten Ereignistyp $\text{ET1}$ wird nur das Ereignis $E_i$ zur Definition benötigt.
- In dem zweiten Ereignistyp $\text{ET2}$ kann zu dem Ereignis $E_i$ die Anzahl $\text{An} \, z_i$ der Ereignisse angegeben werden, die in einem Teilsuchraum geplant werden sollen.
- In dem dritten Ereignistyp $\text{ET3}$ kann zu dem Ereignis $E_i$ die Anzahl $\text{An} \, z_i$ der Ereignisse angegeben werden und eine Liste $LP_i$ von Ereignissen $EP_i$ die parallel zum Ereignis $E_i$ stattfinden sollen. Das Ereignis $E_i$ und die Liste $LP_i$ der parallelen Ereignisse $EP_i$ sollen in einem zergliederten Teilsuchraum gleichzeitig geplant werden.

$\text{ET1}: ET_{1i} \subseteq \{ E_i \}$,
$\text{ET2}: ET_{2i} \subseteq \{ (E_i, \text{An} \, z_i) \}$,
$\text{ET3}: ET_{3i} \subseteq \{ (E_i, \text{An} \, z_i, [EP_1, EP_2, ..., EP_n]) \}$ mit $i, n \in N$

Ereignistypenlisten $\text{ETL}$

Die Ereignistypen werden in Ereignistypenlisten $\text{ETLi} = [ET_{x_1}, ..., ET_{x_n}]$ mit $x \in \{1, 2, 3\}$ und $n \in N$ zusammengefasst. Die einzelnen Ereignistypen $ET_{xi}$ können miteinander kombiniert werden. Es sind folgende Kombinationen zulässig

$\text{ETL1}: ET_{L1} = [ET_{11}, ET_{12}, ..., ET_{1n}]$
$\text{ETL2}: ET_{L2} = [ET_{21}, ET_{22}, ..., ET_{2n}]$
$\text{ETL3}: ET_{L3} = [ET_{31}, ET_{32}, ..., ET_{3n}]$
$\text{ETL4}: ET_{L4} = [ET_{11}, ET_{12}, ..., ET_{1n}, ET_{21}, ET_{22}, ..., ET_{2n}]$
$\text{ETL5}: ET_{L5} = [ET_{11}, ET_{12}, ..., ET_{1n}, ET_{31}, ET_{32}, ..., ET_{3n}]$
$\text{ETL6}: ET_{L6} = [ET_{11}, ET_{12}, ..., ET_{1n}, ET_{21}, ET_{22}, ..., ET_{2n}, ET_{31}, ET_{32}, ..., ET_{3n}]$


Reihenfolge-Constraint RC1

Definition Reihenfolge-Constraint RC1

Das Reihenfolge-Constraint RC1 hat eine Liste von Ereignisstypenlisten $ETL_x$ mit $x \in \{1, 2, 4, 5, 6\}$. In diesem Reihenfolge-Constraint dürfen die unterschiedlichen Ereignisstypenlisten $ETL1, ETL2, ETL4, ETL5$ oder $ETL6$ miteinander kombiniert werden.

Beispiel RC1:

1. $\text{RC1}([[E_1, E_2, E_3], [E_6, E_8, E_{10}]]),$
2. $\text{RC1}([[E_1, \text{An} \, z_1], (E_2, \text{An} \, z_2), (E_3, \text{An} \, z_3)], [(E_5, \text{An} \, z_5), (E_{13}, \text{An} \, z_{13}), (E_{20}, \text{An} \, z_{20})]),$
3. $\text{RC1}([[E_1, \text{An} \, z_1, (E_2, E_3)], (E_5, \text{An} \, z_2), (E_7, \text{An} \, z_7)], [(E_8, \text{An} \, z_8), (E_{10}, \text{An} \, z_{10}), (E_{17}, \text{An} \, z_{17})]]),$
4.3. GLOBALE CONSTRAINTS

Reihenfolge-Constraint RC2

Definition Reihenfolge-Constraint RC2
Das Reihenfolge-Constraint RC2 hat eine Liste von Ereignistypenlisten $ETL_x$ mit $x \in \{3\}$. In diesem Reihenfolge-Constraint darf nur die Ereignistypenliste $ETL_3$ mit dem Ergebnistyp $ET_3$ verwendet werden.

Beispiel RC2:

- $RC2([(E_1, Anz_1, [E_2, E_3]), (E_7, Anz_7, [E_9, E_{10}, E_{11}]), [(E_{14}, Anz_{14}, [E_{15}, E_{16}, E_{17}, E_{18}])]])$

Spezifikation der Reihenfolge-Constraints RC1 und RC2

Die Reihenfolge-Constraints RC1 und RC2 schränken die Domänen in der Ressourcendimension $RD_j$ mit $j \in N$ der Anzahl der Ressourcendimensionen ein, in dem sie die Ereignisse einen zergleiten Teilschraum $zST_k$ mit $k \in N$ der Anzahl der Teilschraume zuordnen. Die reduzierten Domänen können auch wieder erweitert werden, indem eine andere Zuordnung zu einem zergleiten Teilschraum $zST_k$ erfolgt.

Gegeben seien die Mengen der Ressourcenkapazitäten $MK_{j,h}$ für die Kapazitäten $KD_{i,j,k}$ mit $i \in N$ der Anzahl der Kapazitäten der Ressourcendimensionen $RD_j$ für jeden zergleiten Teilschraum $zST_k$ und eine Menge von Ereignissen $ME$ mit $E_i: 1, \ldots, 1, \ldots, n \in N$ der Anzahl der Ereignisse. Es gibt drei Gruppen von Ereignissen: $E_{gb}, E_{vb}, E_{fj}$ die in Abhängigkeit von ihrer Variabilität in der Zuordnung zu den zergleiten Teilschraumen unterschieden werden:

- Die Ereignisse $E_{gb}$ werden genau einem zergleiten Teilschraum zu ordnet.
- Die Ereignisse $E_{vb}$ werden von einem zergleiten Teilschraum beginnend bis zu einem letzten zergleiten Teilschraum fortlaufend zugeordnet, solange sie noch nicht eingeplant sind.
- Die Ereignisse $E_{fj}$ sind frei in ihrer Zuordnung zu einem zergleiten Teilschraum und werden bei vorhandenen Kapazitäten in einem zergleiten Teilschraum diesem zergleiten Teilschraum zugeordnet.

und $RCx(ME, zST_k, MK_{j,h})$ für $x \in \{1, 2\}$:

$$
RCx([],[zST_k, MK_{j,h}]),
RCx([E_{gb}, E_{vb}, E_{fj}],[zST_k, MK_{j,h}]) -
$$
- ermittele-nichtgeplante-Ereignisse($zST_k, ENichtgeplantiste$),
- verschiebe-nichtgeplante-Ereignisse($zST_k, ENichtgeplantiste, MK_{j,h}$),
- bearbeitet($E_{gb}, E_{vb}, E_{fj}),(zST_k, MK_{j,h})$,
- $RCx(Res, zST_{k+1}, MK_{j,k+1})$.

Die Ermittlung nichtplanbarer Ereignisse im zergleiten Teilschraum $zST_{k-1}$ (Teilschritt 1) erfolgt in ermittele-nichtgeplante-Ereignisse($zST_{k-1}, ENichtgeplantiste$). Jedes Ereignis $E_i$ besitzt eine Eigenschaft geplant, die den Planungszustand angibt. Es gibt die zwei Planungszustände geplant und ungeplant. Ermittelt werden alle Ereignisse $E_i$, die ungeplant sind und diese Ereignisse werden in der Liste $ENichtgeplantiste$ weitergegeben.

Die Verschiebung nichtplanbarer Ereignisse in den nächsten zergleiten Teilschraum $zST_k$ bei gleichzeitigen Abgleich der benötigten Ressourcen in den einzelnen Ressourcendimensionen $RD_j$ (Teilschritt 2) erfolgt in verschiebe-nichtgeplante-Ereignisse($zST_k, ENichtgeplantiste, MK_{j,h}$). Die Ereignisse $E_i$ der Liste $ENichtgeplantiste$ werden entsprechend ihrer Zuordnungsmöglichkeiten zu den zergleiten Teilschraumen zugeordnet. Die Ereignisse aus der Gruppe $E_{gb}$, die nur genau einem zergleiten Teilschraum zugeordnet werden konnten, werden nicht neu zugeordnet und bleiben ungeplant. Die Ereignisse der Gruppe $E_{vb}$ werden im Bereich der angegebenen, zergleiten Teilschraumen neu zugeordnet. Wenn
der Bereich der zerlegten Teilschräume ausgeschöpft ist und keine Einplanung erfolgen konnte, bleiben die Ereignisse ungeplant. Die Ereignisse der Gruppe $E_i^h$ werden dem nächsten zerlegten Teilschraum zugeordnet. Erst wenn der gesamte Suchraum bearbeitet wird und das Ereignis nicht eingeplant werden konnte, bleibt es ungeplant. In den Zustand ungeplant kann ein Ereignis $E_i^h$ kommen, wenn zum Beispiel zu geringe Ressourcen vorhanden sind oder zu viele Ereignisse nur wenige spezielle Ressourcen nutzen können.

Die Einplanung der zu planenden Ereignisse in den zerlegten Teilschraum $zST_k$ (Teilschritt 3) erfolgt in bearbeiten($E_{t-el}^{h,i}, zST_k, MRK_{i,k}$). Die in der Liste $E_{t-el}^{h,i}$ dem zerlegten Teilschraum $zST_k$ zugeordneten Ereignisse werden nun in einem Suchprozess unter Berücksichtigung der bekannten gegebenen Constraints eingeplant. Hierbei werden in den Mengen der Ressourcenkapazitäten $MRK_{i,k}$ Kapazitäten $RD_{i,j,k}$ der jeweiligen Ressourcendimension $RD_j$ verbraucht. Hierdurch erfolgt der Aufruf des Constraints für den nächsten zerlegten Teilschraum $zST_{k+1}$.


Im interaktiven Modus der Stundenplanung können Ereignisse innerhalb eines oder über mehrere Teilschräume ausgeplant und wieder eingeplant werden (Verschieben von Ereignissen). Hierfür wird das Ereignis ausgeplant und als ungeplant markiert. Es kann in der Reihenfolgekette belassen werden oder durch Angabe eines neuen Planungszeitpunktes (interaktiv) in der Reihenfolgekette neu platziert werden. Das globale Reihenfolge-Constraint entfernt dann das Ereignis aus der Reihenfolgekette, setzte die Vorgängerbeziehung des nachfolgenden Ereignisses auf das davorliegende Ereignis um und fügt es an der neuen Position (entsprechend des interaktiv festgelegten Zeitpunkts) wieder in die Reihenfolgekette, durch Änderung der Vorgänger und Nachfolgerbeziehungen ein. Es handelt sich hierbei um die Positionsänderung innerhalb eines globalen Constraints und nicht um die Änderung mehrerer einzelner Constraints < oder >, dadurch muss nur ein Constraint geändert werden, was sich effizienter durchführen lässt. Bei dem dann folgenden Absetzen der Constraints brauchen nicht viele einzelne Constraints < oder > abgesetzt werden sondern nur wenige Reihenfolge-Constraints, was den Speicherverbrauch verringert und das Absetzen der Constraints beschleunigt.

Im folgenden Kapitel "Suchverfahren für effiziente Planungen" wird der in der Prozedur bearbeiten($E_{t-el}^{h,i}, zST_k, MRK_{i,k}$) verwendete Suchprozess zur effizienten Planung näher erläutert.
Kapitel 5

Suchverfahren für effiziente Planungen

5.1 Problemzerlegung


Im Folgenden wird ein Algorithmus zur Zerlegung des Suchraums in zerlegte Teilsuchräume erläutert.

5.1.1 Algorithmus zur Problemzerlegung

Der Zerlegungsalgorithmus setzt sich aus mehreren Teilschritten zusammen. Im ersten Teilschritt wird die Struktur und die Vernetzung des Problems analysiert und es werden die zerlegten Teilsuchräume des gesamten Suchraums bestimmt. Im zweiten Schritt erfolgt die Ermittlung der zur Verfügung stehenden Kapazitäten der zerlegten Teilsuchräume.

Analyse der Struktur und Vernetzung und Sortierung des Suchraums (Vorgang 1)


Durch die einzuhaltenen Bedingungen erfolgt meistens eine Verringerung der Problemgröße und die
Festlegung von zeitlichen Bereichen in denen ein Vorgang stattfinden kann (siehe Bild 5.1 Schritt 1). Wenn man diese Eigenschaft ausnutzt, kann der Suchraum der Lösung reduziert werden. Zu diesem Zweck können die einzelnen Bereiche in eine Treppenstruktur umsortiert werden, wie in Bild 5.1 Schritt 2 zu sehen ist.

Abbildung 5.1: Sortierung der Domänen und Bildung einer Treppenstruktur


Vor dem Absetzen der Constraints muss im ungünstigsten Fall der gesamte Suchraum bearbeitet werden. Das ergibt bei der Länge der Domänen X und der Anzahl der Domänen Y einen Aufwand von:

\[ O(Y \times X) \text{ mit } X, Y \in N. \]

Nach dem Absetzen der Constraints müssen nur die Aufwände der potentiellen Bereiche für die Lösung des Problems betrachtet werden, in denen noch Werte in den DomänenvARIABLEN vorhanden sind. Die Summe der Aufwände für die potentiellen Bereiche des Suchraums ergibt dann den Gesamtaufwand und berechnet sich wie folgt, wenn J die Anzahl der potentiellen Bereiche des Suchraums ist:

\[ \sum_{i=1}^{J} O(Y_i \times X_i). \]

Abbildung 5.2: Abschätzung der oberen Grenze des Suchaufwandes

Die Aufwandsberechnungen für das obige Beispiel (siehe Bild 5.2 Schritt 3a und 3b) ergeben die folgenden Werte.
5.1. PROBLEMZERLEGEN

In Schritt 3a ergibt sich der Aufwand aus der Anzahl der Domänen und der Länge der Domänen mit \( O(16 \times 45) = O(720) \) und im Schritt 3b ist der Gesamtaufwand die Summe aus den Aufwänden der einzelnen Bereiche, die sich aus der Anzahl der Domänen und der Länge der Domänen ergeben, mit \( O(5 \times 10) + O(5 \times 20) + O(3 \times 15) + O(3 \times 45) = O(50) + O(100) + O(45) + O(135) = O(330) \).


Das obige Prinzip der Sortierung kann auch innerhalb der gefundenen Bereiche genutzt werden (siehe Bild 5.3 Schritt 4), um eine feinere Strukturierung in den gefundenen Bereichen durch erneute Sortierung zu erreichen. Wenn es keine nennenswerte Veränderungen gibt (weniger als 10 %), wird eine weitere Sortierung des Suchraumes nicht mehr durchgeführt.

Abbildung 5.3: Unterteilung der Suchräume

Im folgenden Abschnitt wird auf die Sortierung des Suchraumes in Bereiche aufgesetzt und die Zerlegung in Teilsuchräume erläutert.

Zerlegung in Teilsuchräume (Vorgang 2)


Im Beispiel ist folgende Anzahl von gleichlangen Bereichen von Werten der Domänen der Ressourcen (auch Ressourcodomänen genannt) ermittelt worden:
Abbildung 5.4: Bestimmung der Zerlegungspositionen

<table>
<thead>
<tr>
<th>Gruppe</th>
<th>Länge eines Bereichs</th>
<th>Anzahl der Domänenwerte</th>
<th>Anzahl gleichgroßer Bereiche</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>56</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>28</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>27</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

In der Gruppe 1 (< 50 % der Teilbereichsgröße) werden die Bereiche der Werte der Domäne der Resource ausgewählt, die den größten Bereich haben und gleichzeitig eine hohe Anzahl haben. Es wird das Maximum \( ZTSR_0 \) über die Anzahl und die Größe der Bereiche gesucht. Das Maximum \( ZTSR_0 \) ergibt dann die vorläufige Größe des zerlegten Teilraumes.

\[
ZTSR_0 = \max(\text{Größe}, \text{Anzahl})
\]

\[
ZTSR_0 = \max(9, 4) \rightarrow 9.
\]

Der zerlegte Teilraum \( zST \) soll gleichmäßig sein und sich an den Grenzen überlappen. Für die Überlappung wird der zerlegte Teilraum an beiden Seiten um einen Ausgleichswert der Domäne der Ressourcen erhöht. Für die gleichmäßige Größe wird ein Ausgleichsfaktor ermittelt, der auch die nicht vorhandene Werte der Domäne der Resource berücksichtigt und sich aus dem aufgerundeten Wert des Quotienten aus dem größten Wert der Domäne der Ressourcen des Bereichs \( MW \) und der Größe \( ZTSR_0 \) des ausgewählten zerlegten Teilraums \( zST \) ergibt. Die endgültige Größe \( ZTSR \) des zerlegten Teilraums \( zST \) ergibt sich nach folgender Formel:

\[
ZTSR = \max((ZTSR_0 + 3), \text{rund}(MW / ZTSR_0))
\]

\[
ZTSR = \max((9 + 3), \text{rund}(56 / 9)) = \max(9 + 3, 6) = \max(12, 6) = 12.
\]

Im Schritt 7 wird dann nach der endgültigen Größe \( ZTSR \) des zerlegten Teilraumes der Teilsuchraum in mehrere kleinere zerlegte Teilsuchräume \( zST \) in horizontaler Ebene umgruppiert (siehe Bild 5.5 Schritt 7). Die Teilsuchräume können am Anfang und Ende des Suchbereichs kleiner sein als die Größe des zerlegten Teilraums \( ZTSR \). Die Größe des zerlegten Teilraums \( ZTSR \) ist nur ein durchschnittlicher Wert, der als Anhaltspunkt gilt, der am Anfang, am Ende und im Planungshorizont den jeweiligen Beginn- und Endpunkten der Bereiche der Domänen angepasst werden muss.

Beispiel ZL1:

Betrachtet wird nun das Problem der Stundenplanung an Weiterbildungseinrichtungen. Bei Weiterbildungseinrichtungen erfolgt der Unterricht meistens in der Zeit von 07:00 Uhr bis 16:00 Uhr von Montag bis Freitag. Am Wochenende erfolgt kein Unterricht. Die Teilnehmer kommen aus verschiedenen Städten.
5.1. PROBLEMZERLEGUNG

Abbildung 5.5: Zerlegung des Teilsuchraums an den Zerlegungspositionen

und reisen zum Anfang der Woche an und am Ende der Woche wieder ab. Im Stundenplan wird hiefür eine An- und Abreisezeit von zwei Stunden eingeplant. Die ersten zwei Stunden am Montag und die letzten zwei Stunden am Freitag einer Woche sind für die An- und Abreise reserviert.


Auf der Zeitachse und damit in den Domänen der Ressource Zeit ergeben sich aufgrund der gegebenen Rahmenbedingungen Unterteilung in Wochen und auch in Tage. Die Unterteilung in den Domänen in die Wochen entsteht durch die unterrichtsfreien Wochenenden, da die Zeiten an den Wochenenden zum Planen von Veranstaltungen durch die Constraints aus den Domänen entfernt werden. Bei der Reduzierung auf die Unterrichtszeiten an den einzelnen Wochentagen werden die Zeiten vor 07:00 Uhr bei 00:00 Uhr beginnend und nach 16:00 Uhr bei 24:00 Uhr endend aus den Domänen wieder durch die Constraints entfernt. Somit ergibt sich eine Unterteilung in die Tage mit Unterrichtszeiten an den Tagen von 07:00 Uhr bis 16:00 Uhr. Am Montag und Freitag werden dann zusätzlich die Domänen um die Reisezeiten reduziert.

Wenn man den Algorithmus zur Problemzerlegung anwendet, werden zusammenhängende Domänenbereiche nur an den Tagen selbst, von 07:00 Uhr bis 16:00 Uhr, gefunden. Es ergeben sich nur zwei Bereichsarten von Domänen, die reduzierten Bereiche am Montag (Tag 1) und Freitag (Tag 5) und die Bereiche am Dienstag (Tag 2), Mittwoch (Tag 3) und Donnerstag (Tag 4). In der nach folgenden Abbildung 5.6 werden im Schritt 5 die Domänen der Ressourcen auf die Häufigkeit des Vorkommens von Werten untersucht und in zwei Gruppen eingeteilt. In der ersten Gruppe sind alle Domänen, deren Anzahl der Werte in den einzelnen Bereichen kleiner als die Hälfte der Gesamtgröße des Suchraums ist (Schritt 5 im oberen Bereich des Bildes 5.6). In der zweiten Gruppe sind alle Domänen, deren Anzahl der Werte der Ressourcen in den einzelnen Bereichen größer und gleich der Hälfte der Gesamtgröße des Suchraums sind. In diesem Beispiel existieren keine Domänen für die Zuordnung in die zweite Gruppe. Auf dieser Gruppeneinteilung der Domäne werden die Häufigkeiten des Auftretens der einzelnen

Abbildung 5.6: Bestimmung der Zerlegungspositionen
Werte in den Domänen der Ressourcen ermittelt. Nach der Ermittlung der Häufigkeiten des Auftretens
von Werten in den Domänen werden die nebeneinander liegenden Summen von Häufigkeiten zu einem
Bereich zusammengfasst. Danach wird aus der Anzahl der Domänenwerte in den zusammenhängenden
Bereiche von aufaddierten Werten der Domänen der Ressourcen, die Längen der zusammenhängenden
Bereiche bestimmt (siehe Bild 5.6 Schritt 6).
In der Tabelle sind die Längen der entstehenden Bereiche der Gruppe 1 und 2 dargestellt mit einer
Einheit gleich einer Stunde. In diesem Beispiel gibt es keine zusammenhängenden Domänen über die
gesamte Zeitachse, somit entfallen die Bereiche in der Gruppe 2, da die größte gefundene Domäne nur
an einem Wochentag (Dienstag (Tag 2), Mittwoch (Tag 3) oder Donnerstag (Tag 4)) vorliegt. Die größte
Domäne hätte für eine Woche die Größe von 24 * 7 = 168, da die Domänen durch das Wochenende
unterbrochen werden.

<table>
<thead>
<tr>
<th>Gruppe</th>
<th>Länge eines Bereichs</th>
<th>Anzahl der Domänenwerte</th>
<th>Anzahl gleichgroßer Bereiche</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>168</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

\[ ZTSR0 = \max(\text{Größe, Anzahl}) - \]
\[ ZTSR0 = \max(9, 3) \rightarrow 9 \]
\[ ZTSR = \max((ZTSR0 + 3), \text{rund}(MW / ZTSR0)) \]
\[ ZTSR = \max(9 + 3, \text{rund}(168 / 9)) = \max(12, 19) = 19 \]

Die Ermittlung der größten Bereiche mit der größten Anzahl an Domänen ergibt die Größe für die
zerlegten Teilsuchräume \( zST \) und wird auf die Domänenbereiche an den Wochentagen Dienstag (Tag
2) bis Donnerstag (Tag 4) in der Zeit von 07:00 Uhr bis 16:00 Uhr als Vorschlag festgelegt. Wenn man
die Zeiten, an denen kein Unterricht stattfindet, mitbetrachtet, ergibt sich eine regelmäßige Aufteilung
der Zeitachse, bei Zerlegung an den Tagesgrenzen, genau nach 24 Stunden. Es ergibt sich eine Aufteil-
ing in Tage, die in Wochen zusammengefasst werden. In der Abbildung 5.7 sind die Positionen für die
Zerlegung dargestellt. Der Übersicht halber sind nur die Tage Montag (Tag 1) bis Freitag (Tag 5) und
nicht das Wochenende mit Samstag (Tag 6) und Sonntag (Tag 7) dargestellt.

Abbildung 5.7: Zerlegung des Teilsuchraums an den Zerlegungspositionen

Die Zusammenfassung in Wochen bietet sich wegen des großen zeitlichen Abstands zwischen den Tagen
an und ist auch wegen der allgemeinen Erstellung von Wochenstundenplänen sinnvoll. Um eine variable
Gestaltung der Unterrichtseinheiten in Einzelstunden oder zusammenhängenden Doppelstunden zu
erlauben, werden die Pausenzeiten hier nicht betrachtet. Aus den Domänen werden hierfür keine
Werte für die Abbildung der Pausenzeiten entfernt.

Beid der weiteren Betrachtung des Beispiels der Stundenplanung an Weiterbildungseinrichtungen wird
von einer Zerlegung in Tagen ausgegangen, die in Wochen zusammengefasst sind.
5.1. PROBLEMZERLEGGUNG

Kapazitätsermittlung der zerlegten Teilsuchräume (Vorgang 3)

Im nächsten Vorgang 3 wird zu jeder Ressource die Kapazität der Ressource aus der Größe der dazu gehörigen Domäne bestimmt. Die ermittelten Ressourcenkapazitäten werden für alle zerlegten Teilsuchräume und alle Ressourcedimensionen ermittelt.

**Definition Kapazität KD**

Die Kapazität $KD_{i,j,k}$ der Ressourcedomäne $D_{i,j,k}$ mit $1 \leq i \leq n$ mit $n$ der Anzahl der Ressourcen, $j > 1$; $2 \leq j \leq m$ mit $m$ der Anzahl der Ressourcedimensionen und $1 \leq k \leq p$ mit $p$ der Anzahl der zerlegten Teilsuchräume, die sich in Abhängigkeit von der Ressource $R_i$ der Ressourcedimension $RD_1$ ergibt, wenn mit der Ressourcedimension $RD_1$ die Resource Zeit $t$ modelliert ist. Die Ressource $R_i$ der Ressourcedimension $RD_1$ wird dann mit $R_t$ bezeichnet.

\[
KD_{i,j,k} = \text{AnzahlElemente}(D_{i,j,k})
\]

mit $i = 1, \ldots, n$; $j = 2, \ldots, m$; $k = 1, \ldots, p$; $n$, $m$, $p \in N$

**Definition Ressourcenkapazität MRK**

Die Ressourcenkapazität $MRK_{j,k}$ sei die Menge der Kapazitäten $KD_{i,j,k}$ der Ressourcedimensionen $RD_1$ der zerlegten Teilsuchräume $ST_k$ für jede Ressource $R_{i,j}$ einer Ressourcedimension $RD_1$ mit $1 \leq i \leq n$ mit $n$ der Anzahl der Ressourcen, $j > 1$; $2 \leq j \leq m$ mit $m$ der Anzahl der Ressourcedimensionen und $1 \leq k \leq p$ mit $p$ der Anzahl der zerlegten Teilsuchräume.

\[
MRK_{j,k} = \bigcup K D_{i,j,k}
\]

mit $i = 1, \ldots, n$; $j = 2, \ldots, m$; $k = 1, \ldots, p$; $n$, $m$, $p \in N$

Beispiel ZL2:

Gegeben sind eine Ressourcedimension $RD_2$ eines zerlegten Teilsuchraums $ST_k$ mit sechs Ressourcen $R_{i,2}$ mit $1 \leq i \leq 6$ der Ressourcedimension $RD_2$ und den dazugehörigen Ressourcedomänen $D_{i,2}$ über der Ressource Zeit $R_i$ der Ressourcedimension $RD_1$, für die die Menge der Ressourcenkapazitäten $MRK_{2,1}$ der einzelnen Kapazitäten $KD_{i,2,1}$ ermittelt wird.

Abbildung 5.8: Kapazitäten der Ressourcedimension $RD_2$ zur Ressourcedimension $RD_1$

Die Menge der Ressourcenkapazitäten $MRK_{2,1}$ der einzelnen Kapazitäten $KD_{i,2,1}$ umfasst dann folgende Elemente $MRK_{2,1} = \{24, 17, 16, 20, 19, 13\}$.

Die ermittelten zerlegten Teilsuchräume $ST_k$ und deren ermittelte Mengen der Ressourcenkapazitäten $MRK_{j,k}$ mit $1 \leq j \leq n$, $1 \leq k \leq n$ $n \in N$ werden danach der Planungskomponente übermittelt.
5.1.2 Planung


Initialisierung


Die ungeplanten Veranstaltungen sind zu Beginn der Planung bereits vorsortiert. Um die Suche zu beschleunigen, ist der gesamte Planungszeitraum mehrfach unterteilt. Er ist zum einen in Tage und zum anderen in Wochen gegliedert. Die Tage dienen vor allem der Sicherstellung der meisten Restriktionen wie zum Beispiel der überschneidungsfreien Anordnung der Veranstaltungen, die Verwendung von Dozenten und Räumen eines Ortes oder aber auch die Einhaltung der maximalen Anzahl von Teilnehmern in einem Raum oder bei einem Dozenten.

Abbildung 5.9: Aufbau der Planungskomponente für das Beispiel der Kursplanung an Weiterbildungseinrichtungen
5.1. PROBLEMZERLEGUNG


Das Einplanen


Durch diese Vorgehensweise wird eine übermäßige Generierung von Freistunden verhindert, da die Veranstaltung immer zum frühesten möglichen Zeitpunkt eingeplant wird. Dies ist eine Heuristik, in deren Ergebiss keine oder nur wenige Freistunden auftreten. Um zusätzlich die lokale Anhäufung von Veranstaltungen des gleichen Fachs zu verhindern, findet für jede Woche des Planungszeitraumes eine Bewertung jedes Wochentages statt. Dazu wird eine Liste generiert, in der die Wochenstage nach Kriterien, wie die Anzahl der bereits verplanten gleichen Veranstaltungen an dem Tag und der Reihenfolge der Tage sortiert sind. Dazu kann der Nutzer einen Schwellenwert festlegen, der bestimmt, wie viele glei-
che Veranstaltungen pro Tag eingeplant werden dürfen. Dabei werden Tage, die diesen Schwellenwert überschritten haben, denen, an denen dieser unterschritten wurde oder keine Veranstaltungen dieser Art eingeplant wurden, benachteiligt.

Nach jedem Einplanen einer Veranstaltung ist es möglich, den Planungsprozess interaktiv zu unterbrechen, um dem Anwender ein individuelles interaktives Ein- oder Umplanen zu jedem beliebigen Stand der Planung zu ermöglichen. Der Anwender kann Einfluss auf die Vorgehensweise bei der automatischen Planung durch zahlreiche veränderbare Parameter und Variablen nehmen, die in den nächsten Abschnitten näher erläutert werden.

**Das Ausplanen**


### 5.1.3 Beschreibung der Algorithmen

**Definition verwendeter Größen**

Sei $E$ der Lehrplan. Dann sind $e_1, e_2, \ldots, e_n \in E$ die einzelnen Veranstaltungen. Die Veranstaltungen werden aus der Veranstaltungsdefinitionen generiert. Die Menge der Definitionen $ED$ und $ed_1, ed_2, \ldots, ed_n \in ED$ sind die einzelnen Definitionen. Sie enthalten zahlreiche Attribute, die die Eigenschaften der Veranstaltungen näher beschreiben. So legen diese Eigenschaften die Anzahl der zu generierenden Veranstaltungen, sämtliche Ressourcenwünsche für den Raum, den Dozenten und die Zeit, den zugehörigen Kurs und noch weitere Zusatzbedingungen fest. Zu jeder Veranstaltungsdefinition gehört eine bestimmte Menge von Veranstaltungen $E_{ed_i} \subseteq E$. Die Wünsche und die bevorzugten Ressourcen sind definiert für $Resource \in \{Raum, Dozent, Zeit\}$, dabei wird die Menge der entsprechenden Ressource mit $R_{Resource}$ betitelt. Folgendermaßen definieren sich die Prioritätstufen:

- Alle möglichen Ressourcen: $A_{Resource_{ed_i}}$
- Alle bevorzugten Ressourcen: $P_{Resource_{ed_i}}$
- Alle Wünsche: $W_{Resource_{ed_i}}$

Es gilt: $W_{Resource_{ed_i}} \subseteq P_{Resource_{ed_i}} \subseteq A_{Resource_{ed_i}} \subseteq R_{Resource_{ed_i}}$

Als Einschränkung ist zu erwähnen, dass es keine bevorzugten Zeiten, sondern nur Zeitwünsche gibt. Um diesen Sachverhalt hier aber nicht als Sonderfall betrachten zu müssen, wird einfach davon ausgegangen, dass $W_{Resource_{ed_i}} = P_{Resource_{ed_i}}$ ist.

Um schnell auf diese Mengen zugreifen zu können, werden sie in nach Prioritäten sortierten Listen hinterlegt. Dabei gibt es für jede Ressource eine Liste, in der nacheinander erst die Wünsche, dann die bevorzugten Ressourcen und letztendlich die möglichen Ressourcen abgelegt sind. Um die einzelnen Untermengen auch isoliert verwenden zu können, existieren Variablen, die jeweils mit dem Wert des letzten Wünsches, der letzten bevorzugten sowie der letzten möglichen Ressourcen initialisiert werden. Es ergeben sich folgende Listen:
5.1. PROBLEMZERLEUGUNG

- Für Resource \( \in \{ \text{Dozent, Raum} \} \):
  \[ PL_{cdi}^{\text{Resource}} = \left\{ p_1, \ldots , p_a, p_{a+1}, \ldots , p_b, p_{b+1}, \ldots , p_n \mid p_1, \ldots , p_a \in W_{cdi}^{\text{Resource}}, p_{a+1}, \ldots , p_b \in P_{cdi}^{\text{Resource}}, p_{b+1}, \ldots , p_n \in A_{cdi}^{\text{Resource}} \right\} ; \]

  lastWish_{cdi}^{\text{Resource}} = a;
  lastPreferred_{cdi}^{\text{Resource}} = b;

  lastPossible_{cdi}^{\text{Resource}} = n; \text{mit } 1 \leq a \leq b \leq n; a, b, n \in N

- Für Resource \( \in \{ \text{Zeit} \} \):
  \[ PL_{cdi}^{\text{Resource}} = \{ p_1, \ldots , p_n \mid p_1, \ldots , p_n \in W_{cdi}^{\text{Resource}} \} ; \]

  lastUsedWish_{cdi}^{\text{Resource}} = a; \text{mit } 1 \leq a \leq n; a \in N


Die Menge der Wochen wird WEEKS genannt und deren einzelnen Elemente week1, week2, \ldots , week_n \in WEEKS. Des Weiteren sind in jeder Woche drei Listen vorhanden: Die erste Liste umfasst alle Veranstaltungen, die in der Woche einzuplanen wären. Die zweite Liste beinhaltet alle Veranstaltungen, die bereits in dieser Woche eingeplant sind. Die dritte Liste ist eine Teilmenge der ersten Liste und umfasst die Veranstaltungen, die in diese Woche eingeplant werden können und noch nicht eingeplant wurden.

In diese Liste werden nur die Veranstaltungen aufgenommen, die in dieser Woche die benötigten Ressourcen zur Verfügung haben, alle anderen Veranstaltungen werden nicht in diese Liste eingefügt. Diese dritte Liste ist letztendlich ausschlaggebend für den Planungsprozess, da diese Liste die Veranstaltungen enthält, die bei der automatischen Planung eingeplant werden. Zusammenfassend gilt folgendes:

- Menge aller Veranstaltungen, die in Woche week_i einzuplanen sind:
  \[ L_{week_i}^1 \subseteq E; l_1^i, \ldots , l_n^i \in L_{week_i}^1 \]

- Menge aller Veranstaltungen, die in Woche week_i bereits eingeplant wurden:
  \[ L_{week_i}^2 \subseteq E; l_1^2, \ldots , l_n^2 \in L_{week_i}^2 \]

- Menge aller Veranstaltungen, die in Woche week_i einzuplanen, aber noch nicht eingeplant sind:
  \[ L_{week_i}^3 \subseteq L_{week_i}; l_1^3, \ldots , l_n^3 \in L_{week_i}^3 \]

Dementsprechend wird die Menge der Tage DAYS genannt und deren einzelne Elemente (day1, day2, \ldots , day_n \in DAYS). Die Funktionen zum Einplanen können Parameter übergeben werden. Die Parameter param stellen eine Teilmenge aller Parameter dar. So kann zum Beispiel festgelegt werden, ob die Wünsche bei der Planung zu beachten oder ob die Räume, Dozenten oder die Zeiten die höchste Priorität besitzen und damit die Freistunden in der jeweiligen Ressourcedimension minimiert werden. Die einzelnen Bedeutungen der Parameter werden im Anhang näher erläutert. Es gilt somit:
In der nachfolgenden Beschreibung der Algorithmen im Pseudocode werden Aufrufe von Methoden, die Funktionen für Anfragen oder Zuweisungen in den globalen Constraints diffn2D anstoßen, mit !!! gekennzeichnet. Es gibt Aufrufe für das Erstellen von freien Zeiträumen/-abschnitten einer Ressource und für das Einplanen von Veranstaltungen (Ereignisse) $e_i$ mit $i \in N$ zu einem Zeitpunkt.

- **freelist**($e_i, \text{ZeitraumID}_v, \text{KursID}, \text{Freelist}$)
  Es wird eine Liste Freelist mit freien Zeiträumen für den Kurs/-gruppe KursID des Ereignisses $e_i$ für einen Zeitraum Zeitraumv erstellt.

- **freelist**($\text{ZeitabschnittKurs}, \text{Ressource}, \text{Zeitraumv}, \text{KursID}, \text{Freelist}$)
  Es wird eine Liste Freelist mit freien Zeiträumen, die die Schnittmenge von Zeitabschnitt des Kurses Kurs und freien Zeiten der Ressource Ressource enthält.

- **isfreeresource**($\text{Ressource}, (((\text{Beginn}, \text{Ende}), \text{Result})$)
  Es wird geprüft, ob die Ressource Ressource im Zeitraum (Beginn, Ende) verfügbar ist.

- **einplanen**($\text{Zeitpunkt}_v, e_i$)
  Plant zum Zeitpunkt Zeitpunktv das Ereignis $e_i$ ein.

Für das Einplanen mit Wünschen werden folgende Aufrufe in der Darstellung als Pseudocode benötigt:

- **!!freelist**($e_i, \text{tag}_v, \text{Kurs}, \text{FT}^{e_i}_{\text{Kurs}}$)!!
  Ermittelt die Liste FT$^{e_i}_{\text{Kurs}}$ mit freien Zeiträumen für den Kurs/-gruppe Kurs des Ereignisses $e_i$ an Tag $\text{tag}_v$.

- **!!freelist**($\text{FT}^{\text{Kurs}}_{e_i}, \text{P}^{\text{highPrior}}_{\text{highRes}}, \text{day}_v, \text{Kurs}, \text{FT}^{\text{highPrior}}_{e_i}$)!!
  Erstellt die Liste FT$^{\text{highPrior}}_{e_i}$ mit freien Zeitabschnitten, die die Schnittmenge von Zeitabschnitt (FT$^{\text{Kurs}}_{e_i}$)$_v$ des Kurses Kurs$_i$ und freien Zeiten der Ressource P$^{\text{highPrior}}_{\text{highRes}}$ enthält.

- **!!freelist**($\text{FT}^{\text{highPrior}}_{e_i}, \text{P}^{\text{lowPrior}}_{\text{lowRes}}, \text{day}_v, y, \text{FT}^{\text{lowPrior}}_{e_i}$)!!
  Erstellt die Liste FT$^{\text{lowPrior}}_{e_i}$ mit freien Zeitabschnitten, die die Schnittmenge von Zeitabschnitt (FT$^{\text{highPrior}}_{e_i}$)$_y$ und freien Zeiten der Ressource P$^{\text{lowPrior}}_{\text{lowRes}}$ enthält.

- **!!einplanen**($\text{FT}^{\text{lowPrior}}_{e_i}$)$_1, e_i$)!!
  Plant zum Zeitpunkt (FT$^{\text{lowPrior}}_{e_i}$)$_1$ das Ereignis $e_i$ ein.

Beim Einplanen einer Veranstaltung zu einem festen gegebenen Zeitpunkt sind folgende Aufrufe in der Darstellung als Pseudocode auszuführen:

- **!!isfreeresource**($\text{Kurs}, (\text{ft}, \text{ft} + l), \text{Result})$)!!
  Es wird geprüft, ob der Kurs Kurs im Zeitraum $(\text{ft}, \text{ft} + l)$ verfügbar ist.

- **!!isfreeresource**($\text{P}^{\text{highPrior}}_{\text{highRes}}, ((\text{ft}, \text{ft} + l), \text{Result})$)!!
  Es wird geprüft, ob die Ressource P$^{\text{highPrior}}_{\text{highRes}}$ im Zeitraum $(\text{ft}, \text{ft} + l)$ verfügbar ist.
5.1. PROBLEMZERLEGUNG

- **!listfreeresource** \((p_{\text{lowPrio}}^\text{free}, (ft, ft+l))!!\)
  Es wird geprüft, ob die Ressource \(p_{\text{lowPrio}}^\text{free}\) im Zeitraum \((ft, ft+l)\) verfügbar ist.
- **!einueplanen** \((ft, e_i)!!\)
  Planiert zum Zeitpunkt \(ft\) das Ereignis \(e_i\) ein.

Zum Erhalt der Übersichtlichkeit wurden bei den Aufrufen nicht alle Haupt- und Untermethoden dargestellt, die durchlaufen werden, bis letztendlich der Aufruf des globalen Constraints \texttt{diffn2D} stattfindet. Am Beispiel des Einplanens einer Veranstaltung innerhalb des Planungsalgorithmus **!!** wird der prinzipielle Ablauf aller Methoden der Aufrufe der globalen Constraints \texttt{diffn2D} dargestellt.

**einueplanen** \((v, e_i)\)

\[
\text{add}(v, e_i, ft_v, k_{\text{Kurs}}, k_{\text{Raum}}, k_{\text{Docent}}, D_i, K_{\text{Kurs}}, K_{\text{Raum}}, I_{\text{Docent}}, \text{state}, \text{canOver}) \\
\text{add}(v, e_{ij}) \\
\text{Look}(ft_v, k_{\text{Kurs}}, D_i, K_{\text{Kurs}}, \text{state}, \text{canOver}) \\
\text{add}(v, e_{ij}) \\
\text{Look}(ft_v, k_{\text{Raum}}, D_i, K_{\text{Raum}}, \text{state}, \text{canOver}) \\
\text{add}(v, e_{ij}) \\
\text{Look}(ft_v, k_{\text{Docent}}, D_i, K_{\text{Docent}}, \text{state}, \text{canOver})
\]

Beim Einplanen einer Veranstaltung \(e_i\) innerhalb des Planungsalgorithmus **einueplanen** \((v, e_i)\) wird im ersten Schritt die Methode **add/12** innerhalb des Tages \(v\) aufgerufen. Die Methode **add/12** bildet die Hauptmethode und stellt die benötigten Werte für die Aufrufe der Untermethode **add/2** und der Funktion Lock bereit. Hierfür wird der Zeitpunkt \(ft_v\), die verwendeten Ressourcen \(k_{\text{Kurs}}, k_{\text{Raum}}\) und \(k_{\text{Docent}}\), die Dauer \(D_i\) des Ereignisses \(e_i\), die Größe der Ressourcenzuordnung \(K_{\text{Kurs}}, K_{\text{Raum}}\) und \(K_{\text{Docent}}\), der Status \(\text{state}\) und das Flag \(\text{canOver}\) ersetzt. Die Hauptmethode **add/12** hat die Form:

\[
\text{add}(v, e_i, ft_v, k_{\text{Kurs}}, k_{\text{Raum}}, k_{\text{Docent}}, D_i, K_{\text{Kurs}}, K_{\text{Raum}}, I_{\text{Docent}}, \text{state}, \text{canOver})
\]

und versucht in der Kurs/Zeit-, Raum/Zeit- und Dozent/Zeit-Relation den Zeitraum für die einzuplanende Veranstaltung \(e_i\) entsprechend zu sperren, was durch den aufeinanderfolgenden Aufruf der Untermethode **add/2** mit der Form **add**(\(v, e_{ij}\)) mit \(j \in \{\text{Kurs}, \text{Raum}, \text{Dozent}, \text{Zeit}\}\) erfolgt. Diese Untermethode **add/2** sichert die Abfrage der zugeordneten Relation und aktiviert dann die Anwendung der Lock-Funktion auf das globale Constraint **diffn2D**. Die Lock-Funktion testet die Verfügbarkeit der Ressource zum angebenden Zeitpunkt und plant bei erfolgreichem Test die Veranstaltung ein. Durch die erfolgreichen Einplanung der Veranstaltung ist dieser Zeitpunkt für alle noch nicht geplanten Veranstaltungen besetzt und steht somit nicht mehr zur Verfügung. Die Anfrage von Veranstaltungen für diesen Zeitpunkt ist dann nicht erfolgreich und in der Angabe von freien Zeitpunkten wird dieser Zeitpunkt nicht mehr genannt. Die freien Zeitpunkte werden global in den Relationen verwaltet und bei Bedarf dort abgefragt. Die erfolgreiche Ausführung aller Lock-Funktionen und der dazugehörigen **add**(\(v, e_{ij}\)) Untermethoden der Relationen wird der Hauptmethode **add**(\(v, e_i, ft_v, k_{\text{Kurs}}, k_{\text{Raum}}, k_{\text{Docent}}, D_i, K_{\text{Kurs}}, K_{\text{Raum}}, I_{\text{Docent}}, \text{state}, \text{canOver}\)) signalisiert und damit wird das Einplanen der Veranstaltung \(e_i\) abgeschlossen. Für den Fall, dass eine Lock-Funktion auf dem globalen Constraint **diffn2D** nicht erfolgreich ist, werden alle anderen Lock-Funktionen innerhalb der Hauptmethode **add**(\(v, e_i\)) zurückgenommen und damit die entsprechenden Planungsbereiche wieder freigegeben. Die folgende Abbildung 5.10 zeigt diesen Vorgang schematisch. Alle anderen Aufrufe zum Erinnern von freien Zeiträumen/abschnitten einer Ressource entsprechen im Grundschatze der Vorgehensweise des Sequenziogramms zum Einplanens einer Veranstaltung aus Abbildung 5.10. Bei den Zwischenschnitten der Untermethoden können zusätzlich weitere Prüfungen zusätzlicher Restriktionen (identischer Ort von Raum und Dozent, Dozentenstunden und andere) stattfinden, was die Schachtelung der Aufrufe und damit die Aufrufhiefe bis zum eigentlichen Aufruf notwendig macht. Dadurch können verschiedenen zusätzliche Funktionalitäten zur Bedingungsüberprüfung und Optimierung effektiv gekapselt werden.
Die Initialisierung


Es werden drei Listen für die Planung erzeugt, die Liste der geplanten Veranstaltungen, die Liste der ungeplanten Veranstaltungen, von der Woche auf Grund der Kapazitätsabschätzung zu geordnet wurden und die Liste der Veranstaltungen, die in dieser Woche unbedingt einzuplanen sind unter Einhaltung der vorgegebenen Randbedingungen (Reihenfolge oder genauer Zeitpunkt). Die Listenstellung erfolgt mit dem Wochenmanager, in dem die init-Funktion aufgerufen wird.

Mit der Funktion Initialisiere-Planungskomponente(Blöcke) wird hinzugefügt, wie die einzelnen Veranstaltungen in Blöcken zusammengefasst werden sollen, und wie sie voneinander abhängen. Die

Abbildung 5.10: Ablauf des Einplanens einer Veranstaltung mit Zeit, Dozent und Raum
5.1. PROBLEMZERLEGUNG

einzelnen Blöcke können mit dieser Initialisierung zu einer größeren Einheit zusammengefasst werden. 
Die Funktion Initialisiere-Planungskomponente(Veranstaltungs eigenschaften) aktualisiert die 
Eigenschaften der Veranstaltungen, die den Status geplant haben. Hierbei werden statistische 
Informationen, Zugehörigkeiten zu Blöcken und Wochenzusammenordnungen gesetzt. Die Funktion Initialisiere- 
Planungskomponente(Stundentüberwachung) überwacht für jeden Dozenten, dass der Dozent 
icht zu viele Wochenstunden hat und nicht über ein gewisses Limit hinaus Überstunden erreicht.

initialisierung() 
{
  vorhandene Veranstaltungen
  E = \{e_1, \ldots, e_m\}
  grundlegende Initialisierungen
  initialisierung()
  Sortiere nun die Veranstaltungen ihren Definitionen zu
  sort-Def-Veran()
  Versuche nun vorhandene Veranstaltungen einzuplanen
  einplan-Veran-Versuch()
  Initialisiere nun die Verteilung auf die Wochen, die Blöcke, die Eigenschaften zu den 
  Veranstaltungen und Überwachung weiterer Bedingungen zum Beispiel der Stunden der Dozenten
  Initialisiere-Planungskomponente(Wochen)
  Initialisiere-Planungskomponente(Blöcke)
  Initialisiere-Planungskomponente(Veranstaltungseigenschaften)
  Initialisiere-Planungskomponente(Stundentüberwachung)
  Merke, dass die Planungskomponente initialisiert ist
  Planung initialisiert = ja
}

In der Funktion initialisierung() werden in den Funktionen Init-Variablen (Startdatum, täglicher 
Beginn, tägliches Ende, Genauigkeit) die entsprechenden Werte des Plans zugewiesen und die 
notwendigen Speicherservierungen ausgeführt.

In der Funktion Init-Konfiguratoren-Relationen([Zeit/Dozent, Zeit/Raum, Zeit/Kurs]) gibt es für jede Relation, zum Beispiel für Zeit/Kurs, einen eigenen Konfigurator. In dem Konfigurator sind Informationen enthalten, wie die Anzahl der Ressourcen, die Ressourcen selbst und vorverarbeitet Daten, die den Zugriff beschleunigen. Die vorverarbeiteten Daten werden nur einmal ermittelt und können dann bei Bedarf ohne zusätzliche Zeit für die Ermittlung der gewünschten Information benutzt werden.

In der Funktion Init-Zusatzrestriktionen(Arbeitszeit-Dozenten) werden die benötigten Objekte 
instanziert, welche die zusätzlichen Restriktionen verarbeiten.

In der Funktion Erstelle TageMenge(DAYS) mit DAYS={day_1, \ldots, day_n} werden die Scheduler für 
die einzelnen Tage mit ihren einzelnen Relationen tageweise aufgebaut. Hierfür werden die Relationen 
instanziert und die eigenen Felder generiert.

Hierbei wird auf die zuvor instanzierten Konfiguratoren für die einzelnen Relationen und deren Re- 
striktionen zurückgerufen. Des Weiteren werden die Sperrenzeiten in jeder Relation ergänzt.

initialisierung() 
{
  Init-Variablen (Startdatum, täglicher Beginn, tägliches Ende, Genauigkeit)
  Init-Konfiguratoren-Relationen([Zeit/Dozent, Zeit/Raum, Zeit/Kurs])
  Init-Zusatzrestriktionen(Arbeitszeit-Dozenten)
  Nun werden die einzelnen Tage initialisiert, hierbei werden bereits alle 
  Sperrenzeiten und andere Abhängigkeiten in den zweidimensionalen Feldern ergänzt
  Erstelle TageMenge(DAYS) mit DAYS={day_1, \ldots, day_n}
  Erstelle-Veranstaltungs-/Veranstaltungsdurchschnittsdefinition Relation \{ed_i\}
  i=1
while (i ≤ |ED|?)
{
    Vertausche Element in der Ressourcenliste zufällig, um locale
    Häufung bei einer Ressource während der Planung zu vermeiden
    Hole-Liste $PL_{ed_i}^{Resource}$; Ressource $\in \{Raum, Dozent\}$
    Vertauschung der Elemente innerhalb der Teillisten von Wünschen, bevorzugten und
    möglichen Ressourcen
    Vertausche-zufällig-Elemente($PL_{ed_i}^{Resource}$)
    i=i+1
}

In der Funktion sort-Def-Veran() werden die Veranstaltungen sortiert nach der Zugehörigkeit zu ei- 
ner Variante des Plans. In einem Plan können unterschiedliche Varianten enthalten sein, aber nur eine 
Variante kann aktuell sein. Die anderen Varianten sind dann nicht im Plan enthalten. Des Weiteren 
werden ermittelt, ob die Anzahl der Veranstaltungen sich in den einzelnen Varianten geändert hat.

sort-Def-Veran()
{
    while (i ≤ |E|?)
    {
        if (Veranstaltung $e_i^{ed_j}$ gehört zur aktuellen Variante?)
            if (Veranstaltung $e_i^{ed_j}$ nicht in die Relation $ed_j/\{e_i^{ed_j}, \ldots, e_k^{ed_j}\}$ eingefügt?)
                { 
                    Dieser Fall tritt ein, wenn die Anzahl der Veranstaltungen von Definition $ed_j$
                    nach der letzten Planung heruntergesetzt wurde
                    Lösche-Veranstaltung $e_i^{ed_j}$
                }
        Nimm nächste Veranstaltung: i=i+1
    }

Die Funktion einplan-Veran-Versuch() überprüft alle geplanten Veranstaltungen, ob die angegeben 
Daten der Veranstaltungen unter Beachtung der Ressourcennutzung korrekt sind. Hier werden Fehler 
bei der Datenübertragung oder Datenmanipulationen erkannt und die betroffenen Veranstaltungen nicht 

einplan-Veran-Versuch()
{
    j=1
    while (j ≤ |ED|?)
    {
        Hole-Relation $ed_j/\{e_1^{ed_j}, \ldots, e_k^{ed_j}\}$ der Definition $ed_j$
        i=1
        while (i ≤ k ?)
        {
            if (Veranstaltung $e_i^{ed_j}$ bereits verplant?
                { 
                    Ermitte-Tag $day_t$, an dem die Veranstaltung $e_i^{ed_j}$ eingeplant werden soll
                    if(Einplanen von Veranstaltung $e_i^{ed_j}$ erfolglos?)
                        Ergänze-Veranstaltung $e_i^{ed_j}$ bei ungeplanten Veranstaltungen
                }
            }
        }
    }
}
5.1. PROBLEMZERLEGENG

}  
  else
  {  
    Ergänze Veranstaltung $e_i^{ed}$ bei ungeplanten Veranstaltungen
  }
  Nehme nächste Veranstaltung: i=i+1
  }
  Nehme nächste Definition von Veranstaltungen: j=j+1

Ergänze fehlende Veranstaltungen in jeder Relation $ed_j/\{e_1^{ed}, \ldots, e_k^{ed}\}$, so dass $k=n$

In der Funktion Einplanen von Veranstaltung $e_i^{ed}$ erfolglos? innerhalb der Funktion einplan
Veran-Versuch() sind zwei Fälle zu unterscheiden. Sind bereits verplante Veranstaltungen vorhanden,
so wird versucht, diese einzuplanen. Hierfür wird jede vorhandene Veranstaltung betrachtet und geprüft,
ob sie eingepflanzt ist. Es wird dafür der Tag (das Objekt des Tages) ermittelt, an dem die Veranstal-
tung eingepflanzt werden soll. Nach der Ermittlung des korrespondierenden Schedulers für diesen Tag
wird nun versucht, diese Veranstaltung an diesem Tag einzuplanen. Hierbei wird in jeder Relation an
der entsprechenden Position im globalen Constraint eine Lock-Funktion aufgerufen und die versucht,
diese Position für die zu planende Veranstaltung zu reservieren. Schlägt die Reservierung fehl (die
Lock-Funktion scheitert), so ist diese Position bereits durch eine andere Veranstaltung oder Sperrzeit
belegt. Zur Einplanung einer Veranstaltung müssen alle drei Relationen betrachtet werden, und in jeder
Relation wird eine Lock-Funktion aufgerufen. Wenn nur eine Lock-Funktion von den drei notwendigen
Lockfunktionen fehlt, wird das Einfügen der Veranstaltung rückgängig gemacht.

Die Übermittlung von inkonsistenten und falschen Planungsdaten kann zum Ausplanen einer schon einge-
planten Veranstaltung führen. Hier erfolgt eine Überprüfung der gegebenen Daten und nur die Daten
d von geplanten Veranstaltungen, die einen insgesamt konsistenten Zustand ergeben, werden akzeptiert.

Das Einplanen aller ungeplanten Veranstaltungen

Wird diese Funktion aufgerufen, so sind die Veranstaltungen bereits auf die einzelnen Wochen verfor-
telt und enthalten auch die vorab interaktiv geplanten Veranstaltungen. Die Listen mit den verteilten,
geplanten Veranstaltungen werden von der ersten bis zur letzten Woche des Planungshorizontes abge-
arbeitet und es wird versucht, alle Veranstaltungen einzuplanen. Der Algorithmus bietet des Weiteren
die Möglichkeit, von außen unterbrochen zu werden, sobald der Planungsprozess für eine Veranstaltung
beendet ist.

schedule(param)
{
  Setze lastUsedDay, = day_1 und lastUsedWishResSource = 1
  Hole-Menge WEEKS = \{week_1, \ldots, week_n\}
  i=1
  while (i \leq m)  
    {  
      Hole $L_3^{\text{week}} = \{l_1, \ldots, l_n\}$ von week_i
      j=1
      while ( Soll fortgesetzt werden und j \leq n? )
        {  
          Versuche Veranstaltung $l_j^{ed}$ einzuplanen; rufe Schedule($l_j^{ed}$, param) auf
          Frage ab, ob Prozess abgebrochen werden soll
          Nehme nächste ungeplante Veranstaltung: j=j+1
        }
    }
  }
}
Nehme nächste Woche: i=i+1
}

\( \text{Die erstmögliche Tage sind nun nicht mehr der erste Tag des} \)
\( \text{Planungszeitraumes - setze die Variablen auf diesen zurück} \)
Setze lastUsedDayed = day1 und lastUsedWished, = 1
}

Das Einplanen einer Veranstaltung

Der folgende Algorithmus der Prozedur \( \text{schedule}(e_{current}, param) \) plant eine Veranstaltung \( e_{current} \) ein und beachtet hierbei sowohl Zeitwünsche als auch Ressourcenpräferenzen. Des Weiteren werden zusätzliche Restriktionen wie definierten Planungszeitraum für Veranstaltung und Kurs berücksichtigt. Beim Einplanen werden Heuristiken angewendet, wie zum Beispiel die Reduzierung von Freistunden durch das frühest mögliche Einplanen.

\[ \text{schedule}(e_{current}, param) \]
\[
\{ \\
\text{if (Veranstaltung } e_{current} \text{ bereits eingeplant?)} \\
\text{Lösung gefunden} \\
\}
\]

Ermittle Definition \( ed_{current} \) von der Veranstaltung \( e_{current} \)

\[
\text{if (} (\text{param} \cap \{SCHEDULE\_TIME\} = \emptyset) \text{ } \vee \text{ } (\text{param} \cap \{PRIORITY\_TIME\} \neq \emptyset)) \\
\{ \\
\text{Wenn Wunschdozenten und Wünschräume vorhanden sind, dann gibt es auch} \\
\text{bevorzugte Dozenten und Räume, dies wurde vorher sichergestellt} \\
\text{Hole } \{pl_i \in P_{ed_{current}} | pl_i \in W_{ed_{current}} \} \\
i=lastUsedWished_{ed_{current}} \\
\text{while (} i \leq n \text{?)} \\
\{ \\
\text{if (} pl_i \text{ ist Zeitpunkt?)} \\
\{ \\
\text{Versuche zu dem Zeitpunkt einzuplanen} \\
\text{Breche ab, wenn Lösung gefunden} \\
\text{setze lastUsedWished_{ed_{current}} = i} \\
\} \\
\text{elseif (} pl_i \text{ ist Zeitspanne?)} \\
\{ \\
\text{Ermittle Startzeit, Endzeit} \\
\text{while (Startzeit} \leq \text{Endzeit?)} \\
\{ \\
\text{Versuche, zum Zeitpunkt Startzeit einzuplanen} \\
\text{Breche ab, wenn Lösung gefunden} \\
\text{setze lastUsedWished_{ed_{current}} = i} \\
\text{Erhöhe Startzeit} \\
\} \\
\} \\
\text{Nimm nächste Wunschzeit: } i = i + 1 \\
\}
\]

\( \text{Der Punkt wird erreicht, wenn die Wunschzeiten zu keinem Resultat geführt haben} \)
\( \text{if (} \text{param} \cap \{SCHEDULE\_TIME\} = \emptyset) \) \\
\text{Breche ab, ohne eine Lösung gefunden zu haben} \)
5.1. PROBLEMZERLEGUNG

Ermittle ersten Tag, an dem gesucht werden soll
Ermittle-Woche week_j, in die die Veranstaltung eingeplant werden soll
Setze den ersten Tag firstDay gleich dem ersten Tag der Woche week_j
if (param \cap \{USELASTINSERTION\} \neq \emptyset und firstDay < lastUsedDay_{ed_{\ldots,\ldots}})
{
  erster Tag ist gleich dem Tag, an dem zuletzt
  eine Veranstaltung dieses Typs eingeplant wurde
  firstDay = lastUsedDay_{ed_{\ldots,\ldots}}
}
schedule-week(e_{current}, param, firstDay)
}

In der Prozedur schedule-week(e_{current}, param, firstDay) wird die Veranstaltung in einer Woche an
einem bestimmten Tag geplant. Die Wunschzeiten konnten nicht erfüllt werden. Es wird nun versucht
die Veranstaltung in der aktuellen Woche unter Beachtung der Ressourcenauslastung und -wünsche zu
einem andern Zeitpunkt an dem gewählten Tag firstDay einzuplanen.

schedule-week(e_{current}, param, firstDay)
{
  Ermittle aktuelle Woche week_j des Tages firstDay
  Als nächstes sind die Ressourcen, nach Priorität zu verteilen
  Die Ressourcenlisten sind vorsortiert: erst kommen die Wünsche, dann die
  Bevorzugten und als letztes die Möglichkeit, außerdem sind sie indizierbar
if (param \cap \{PRIORITYROOM\} \neq \emptyset)
  { 
    hohe Priorität haben Räume, also highPrio = Raum
    niedrige Priorität Dozenten, also lowPrio = Dozent
  }
else
  { 
    hohe Priorität haben Dozenten, also highPrio = Dozent
    niedrige Priorität Räume, also lowPrio = Raum
  }

  Lege die letztmöglichen Ressourcen fest
if (param \cap \{USEALL, USEPOSSIBILITIES\} = \emptyset)
  {
    Setze letztmögliche Ressourcen mit dem Index des letzten Wunsches gleich
    lastUsedResource_{ed_{\ldots,\ldots}} = lastWishedResource_{ed_{\ldots,\ldots}} ; Ressource \in \{Dozent, Raum\}
  }
elseif(((param \cap \{USEPOSSIBILITIES\} \neq \emptyset) \land (param \cap \{USEALL\} = \emptyset))
  {
    Setze letztmögliche Ressourcen mit dem Index der letzten bevorzugten Ressource gleich
    lastUsedResource_{ed_{\ldots,\ldots}} = lastPreferredResource_{ed_{\ldots,\ldots}} ; Ressource \in \{Dozent, Raum\}
  }
else
  { 
    Setze letztmögliche Ressourcen mit dem Index der letzten möglichen Ressource gleich
    lastUsedResource_{ed_{\ldots,\ldots}} = lastPossibleResource_{ed_{\ldots,\ldots}} ; Ressource \in \{Dozent, Raum\}
  }
}
schedule-day($e_{current}$, $param$, $firstDay$)

In der Prozedur schedule-day($e_{current}$, $param$, $firstDay$) erfolgt die eigentliche Planung. Davor wurden die Ressourcen nach Prioritäten verteilt.

schedule-day($e_{current}$, $param$, $firstDay$)
{
    Hole Menge DAYS = $\{day_1, \ldots, day_v, \ldots, day_o\}$

    Schleife der eigentlichen Planung
    while ($firstDay \leq day_o$)
    {
        Ermitte den letzten Tag $dayOfWeek_{last}^{week_j}$ der aktuellen Woche week_j

        Lege Liste mit den Tagen der aktuellen Woche an, ohne diesen die Anzahl der mit der zu planenden identischen Veranstaltungen zu

        Erstelle Liste $DL^{week_j}$
        $dayOfWeek_{last}^{week_j} = dayOfWeek_{first}^{week_j}$
        while ($firstDay \leq dayOfWeek_{last}^{week_j}$ \& $firstDay \leq day_o$)
        {
            Ermitte Variable $e_{j_{firstDay}}^{d_{v_{firstDay}}}$, die die Anzahl der identischen Veranstaltungen an dem Tag wiedergibt
            Füge am Ende der Liste Tupel $dc^{d_{v_{firstDay}}}_{j_{firstDay}} = (firstDay, e_{j_{firstDay}}^{d_{v_{firstDay}}})$
            mit Tag und ermittelter Wert ein
            Nehme nächste Tag: $firstDay = firstDay + 1$
        }

        Sortiere: Tage, die eine bestimmte Anzahl von Veranstaltungen überschritten haben, werden weiter hinten je nach Anzahl aufsteigend sortiert, der Rest wird vorn aufsteigend nach Tag sortiert
        Sortiere-Liste($DL^{week_j}$)

        Gehe nun jeden Tag $v$ in dieser Liste durch und suche Möglichkeit, die Veranstaltung einzuplanen
        $i = 1$
        while ($i \leq |DL^{week_j}|$)
        {
            Nehme-Tupel $dc^{d_{v_{firstDay}}}_{k_{firstDay}} = (day_v, e_{v_{firstDay}}^{d_{v_{firstDay}}})$
            !!freeList($e_v$, $day_v$, $K_{v_{Kurs}}$, $FT^K_{v_{Kurs}}$)!!
            Ermitte Liste $FT^K_{v_{Kurs}}$ mit freien Zeiträumen für den Kurs/-gruppe der Veranstaltung an Tag $day_v$.
            if ($|FT^K_{v_{Kurs}}| \neq 0$)
            {
                Prüfe, ob es Zeitabschnitte gibt, an dem ein Dozent und Raum frei ist
                $x = 1$
                while ($x \leq |FT^K_{v_{Kurs}}|$)
                {
                    if( Zeitabschnitt $ft^{K_{v_{Kurs}}}$ nicht lang genug für Veranstaltung ? )
                        Führe fort mit dem nächsten Zeitabschnitt: $x = x + 1$
                    }
                    Probieren jede Ressource mit hoher Priorität
                }
            }
        }
    }
}

KAPITEL 5. SUCHVERFAHREN FÜR EFFIZIENTE PLANUNGEN
Erster Index von Element mit hoher Priorität ist 1

\[ \text{highRes} = 1 \]

while (\text{highRes} \leq \text{lastUsedResource}_{ed_{current}}^\text{highPrio} )

{  
  
  
  \[ \text{!!freelist}(\{f_{v_{Kurs}}^\text{highPrio}\}_x, p_{v_{highRes}}^\text{highPrio}, d_{v_{Kurs}}, FT_{v_{highPrio}})!! \]
  Erstellt Liste \( FT_{v_{highPrio}} \) mit freien Zeitabschnitten, die die Schnittmenge von Zeitabschnitt \( (f_{v_{Kurs}})_x \) und freien Zeiten der Ressource \( p_{v_{highRes}}^\text{highPrio} \) enthält.

  if (\( |FT_{v_{highPrio}}| \neq 0 \))

  {  
    
    Prüfe jeden Zeitabschnitt \( y \) und versuche, freie Ressource niedriger Priorität zu finden
  
  
  y = 1
  while (\( x \leq |FT_{v_{highPrio}}| \))

  {  
    Probiere jede Ressource mit niedriger Priorität
    Erster Index von Element mit niedriger Priorität ist 1

  
  \[ \text{lowRes} = 1 \]

  while (\text{lowRes} \leq \text{lastUsedResource}_{ed_{current}}^\text{lowPrio} )

  {  
    Der Ort beider Ressourcen muss identisch sein
    if (\text{ort}(p_{v_{highRes}}^\text{highPrio}) = \text{ort}(p_{v_{lowRes}}^\text{lowPrio}) )

    {  
      
      \[ \text{!!freelist}(\{f_{v_{lowPrio}}^\text{lowPrio}\}_y, p_{v_{lowRes}}^\text{lowPrio}, d_{v_{lowPrio}}, FT_{v_{lowPrio}})!! \]
      Erstellt Liste \( FT_{v_{lowPrio}} \) mit freien Zeitabschnitten, die die Schnittmenge von Zeitabschnitt \( (f_{v_{lowPrio}})_y \) und freien Zeiten der Ressource \( p_{v_{lowRes}}^\text{lowPrio} \) enthält.

      if (\( |FT_{v_{lowPrio}}| \neq 0 \))

      {  
        Zeitpunkt zum Einplanen gefunden
        Erster Wert in der List entspricht dem Begin der Veranstaltung
        !\text{einplanen}(\{f_{v_{lowPrio}}^\text{lowPrio}\}_1, e_i)!!
        Plant zum Zeitpunkt \( (f_{v_{lowPrio}})_1 \) ein.
        Lösung gefunden

      
      
      Nehme nächste Ressource: \text{lowRes} = \text{lowRes} + 1
      
      
      Nehme nächsten Zeitabschnitt: \( y = y + 1 \)

    
    
    Nehme nächste Ressource: \text{highRes} = \text{highRes} + 1

    
    
    Nehme nächsten Zeitabschnitt: \( x = x + 1 \)

    
    
    Nehme nächstes Tag/Anzahl-Tupel: \( i = i + 1 \)

    
    
    Erhöhe die aktuelle Woche: \( j = j + 1 \)

  
  
  Automatische Planung für diese Veranstaltung ist fehlgeschlagen
Einplanen einer Veranstaltung zu einem festen Zeitpunkt

Der nachfolgende Algorithmus plant eine Veranstaltung zu einem bestimmten Zeitpunkt unter Berücksichtigung der Ressourcenpräferenzen ein.

```
ScheduleAtTime(e_current, ft, l, param)
{
  Finde-Tag(day_current)
  if (Tag(day_current)nichtgefunden?)
    Automatische Planung für diese Veranstaltung ist fehlgeschlagen
  Prüfe, ob der Kurs zu dem Zeitraum (ft, ft+l) verfügbar ist
  Ermittle Veranstaltungsdefinition ed_current zur Veranstaltung e_current
  Ermittle Kurs von der Veranstaltung ed_current
  if (not (!!isfreeressource(Kurs, (ft, ft+l), Result)))!!
    (Kurs im Zeitraum (ft, ft+l) nicht verfügbar ?)
    Automatische Planung für diese Veranstaltung ist fehlgeschlagen
  Als nächstes sind die Ressourcen nach Priorität zu verteilen
  Die Ressourcenliste sind vorsortiert: erst kommen die Wünsche, dann die
  Bevorratungen und als letztes die Möglichkeiten, ausserdem sind sie indizierbar
  if (param ∩ {PRIORITYOOM} ≠ ∅ ?)
  {  
    hohe Priorität haben Räume, also highPrio = Raum
    niedrige Priorität Dozenten, also lowPrio = Dozent
  }
  else
  {  
    hohe Priorität haben Dozenten, also highPrio = Dozent
    niedrige Priorität Räume, also lowPrio = Raum
  }

  Lege die letztmöglichen Ressourcen fest
  if (param ∩ {USEALL, USEPOSSIBILITIES} = ∅ ?)
  {
    Setze letztmögliche Ressourcen mit dem Index des letzten Wunsches gleich
    lastUsedResourceRessourceedCurrent = lastWishedResourceedCurrent, Ressource ∈ {Dozent, Raum}
  }
  elseif((param ∩ {USEPOSSIBILITIES} ≠ ∅) ∧ (param ∩ {USEALL} = ∅) ?)
  {
    Setze letztmögliche Ressourcen mit dem Index der letzten Bevorratungen gleich
    lastUsedResourceRessourceedCurrent = lastPreferrededCurrent, Ressource ∈ {Dozent, Raum}
  }
  else
  {  
    Setze letztmögliche Ressourcen mit dem Index der letzten Möglichkeiten gleich
    lastUsedResourceRessourceedCurrent = lastPossibleedCurrent, Ressource ∈ {Dozent, Raum}
  }

  Schleife der eigentlichen Planung
  Probiere jede Ressource mit hoher Priorität
```
5.1. PROBLEMZERLEGUNG

Erster Index von Element mit hoher Priorität ist 1
highRes = 1

while(highRes ≤ lastUsedRessource_{highPrio}^c_{edcurrent} ? )
{
    if(!isfreeresource(pl_{highRes}^{highPrio}, (ft, ft + l), Result)!!
    (Ressource pl_{highRes}^{highPrio} im Zeitraum (ft, ft + l) verfügbar ?)
    {
        Probe jede Ressource mit niedriger Priorität
        Erster Index von Element mit niedriger Priorität ist 1
        lowRes = 1
        while(lowRes ≤ lastUsedRessource_{lowPrio}^c_{edcurrent} ? )
        {
            Der Ort beider Ressourcen muss identisch sein
            if(ort(pl_{highRes}^{highPrio}) = ort(pl_{lowRes}^{lowPrio}) ? )
            {
                if(!isfreeresource(pl_{lowRes}^{lowPrio}, (ft, ft + l))!!
                (Ressource pl_{lowRes}^{lowPrio} im Zeitraum (ft, ft + l) verfügbar ?)
                {
                    it Lösung für Zeitpunkt ft zum Einplanen gefunden
                    !einplanen(ft, e_i)!!
                    Plane zum Zeitpunkt ft ein
                    Lösung gefunden
                }
            }
            Nehme nächste Ressource: lowRes = lowRes + 1
        }
    }
    Nehme nächste Ressource: highRes = highRes + 1
    k = 1
}

Automatische Planung für Veranstaltung e_{current} fehlgeschlagen

5.1.4 Heuristiken

Die automatische Planung bietet zahlreiche Möglichkeiten, um den Zeitaufwand, aber auch die Qualität der Lösung zu verbessern. Die verwendeten Techniken zur Verringerung des Zeitaufwandes wurden bereits ausführlich erläutert. Nachfolgend sind Techniken beschrieben, die vor allem die Qualität der Lösung verbessern sollen.

Die Ressourcenlisten

Wie schon erwähnt, werden in der Initialisierungsphase Listen erstellt, die die möglichen Ressourcen enthalten. Diese Liste für die Dozenten und Räume sind sortiert nach Wünschen, bevorzugten Ressourcen und letztendlich den möglich Ressourcen und es bilden sich dabei drei Teillisten innerhalb der Liste. Würden diese Listen in alphabetischer Reihenfolge der Ressourcen geführt, könnte dies den unangenehmen Nebeneffekt bewirken, dass zum Beispiel gerade die Dozenten, deren Namen weit vorn im Alphabet stehen, überdurchschnittlich viel zu tun hätten. Um dem vorzubeugen, werden bei der Initialisierung die Teillisten der bevorzugten und möglichen Ressourcen nach dem Erstellen neu geordnet, und zwar so, dass die Elemente der bevorzugten und möglichen Ressourcen eine zufällige Position in ihrer Teilliste
Die Liste der ungeplanten Veranstaltungen pro Woche

Jede Veranstaltung kann eine unterschiedliche Anzahl von möglichen Ressourcen für Räume, Dozenten und Zeiten haben. Da meistens die Veranstaltungen untereinander nicht priorisiert sind oder die Priorisierung nicht für jede Veranstaltung unterschiedlich ist, zwei oder mehrere Veranstaltungen haben die gleiche Priorität, wird zur Bestimmung der Reihenfolge bei der Sortierung in die Liste der ungeplanten Veranstaltungen pro Woche eine Engpässanalyse durchgeführt. Hierbei soll verhindert werden, dass Veranstaltungen mit vielen möglichen Ressourcen vor solchen mit wenigen möglichen Ressourcen eingeplant werden, da dies dazu führen kann, dass bei den ersteren genau die Ressource zur Anwendung kommt, die als einzig mögliche bei der letzteren zur Verfügung steht, obwohl die erstere durchaus noch andere Alternativen hat könnte. Wird dies verhindert, kann die Anzahl der verplanten Veranstaltung eventuell erhöht werden.

Aus diesem Grund werden ungeplante Veranstaltungen nach Priorität in die Liste der ungeplanten Veranstaltungen eingesortiert. Je niedriger der Wert der Priorität ist, desto weiter vorwärts wird das Element eingesortiert. Die Priorität berechnet sich aus dem Minimum der Anzahl der mögliche Räume und Dozenten zu einer Definition einer Veranstaltung. Sind in der Definition einer ungeplanten Veranstaltung keine Zeitwünsche definiert, so wird die Priorität gleich gesetzt (zum Beispiel um die Hälfte des Wertebereichs der Prioritäten), so dass diese Veranstaltungen höher priorisiert sind als alle anderen. Dies ergibt dann folgendes:

\[
prio_{edl} = \min (lastPossible_{edl}^{dozent}, lastPossible_{edl}^{raum}) + \begin{cases} \sum P_{edl}^{last} & : 30000 \\ \sum P_{edl}^{first} & : 0 \end{cases}
\]

Sortierung der Tage einer Woche innerhalb der Planung


Um dem Problem vorzubeugen, wird in der automatischen Planung wöchentlich vorgegangen und jeder Wochentag anhand der Anzahl der bereits vorhandenen Veranstaltungen bewertet. Dabei werden die Wochentage, oder eine Untermenge davon (kommt dann vor, wenn der zuletzt genutzte Tag genommen werden soll, dieser aber kein Montag war), nach verschiedenen Kriterien sortiert. Je nachdem, ob die Anzahl der Veranstaltungen pro Definition der Veranstaltung und Tag \( c_{edl}^{dayi} \), kleiner oder größer als \( MaxEventsPerDay \) ist, werden die Tage entweder in der Reihenfolge der Tage oder in der Reihenfolge der Anzahl der Veranstaltungen sortiert. Ibs und rhs sind Tupel und definiert als \( lhs_{edl}^{dayi} = \left( dayi, c_{edl}^{current} \right) \) und \( rhs_{edl}^{dayi} = \left( dayj, c_{edl}^{current} \right) \). Dann gilt also:
5.1. PROBLEMZERLEGUNG

\[
\left( \left( \frac{\text{lhs}^d_{i_{\text{current}}}}{\text{rhs}^d_{j_{\text{current}}}} \right) \geq \text{MaxEventsPerDay} \right) \land \left( \left( \frac{\text{lhs}^d_{i_{\text{current}}}}{\text{rhs}^d_{j_{\text{current}}}} \right) < \left( \frac{\text{rhs}^d_{j_{\text{current}}}}{\text{rhs}^d_{i_{\text{current}}}} \right) \right)
\]
\[
\lor
\]
\[
\left( \left( \frac{\text{lhs}^d_{i_{\text{current}}}}{\text{rhs}^d_{i_{\text{current}}}} \right) \geq \text{MaxEventsPerDay} \right) \land \left( \left( \frac{\text{lhs}^d_{i_{\text{current}}}}{\text{rhs}^d_{i_{\text{current}}}} \right) < \left( \text{rhs}^d_{j_{\text{current}}} \right) \right) \lor \left( \left( \frac{\text{lhs}^d_{i_{\text{current}}}}{\text{rhs}^d_{i_{\text{current}}}} \right) < \left( \text{rhs}^d_{j_{\text{current}}} \right) \right)
\]
\[
\implies \text{lhs}^d_{i_{\text{current}}} < \text{rhs}^d_{j_{\text{current}}}
\]

Damit wurde erreicht, dass die Veranstaltungen so früh wie möglich eingepflanzt werden können (Reduzierung der Freistunden) und keine lokalen Häufungen von gleichen Veranstaltungen entstehen.
KAPITEL 5. SUCHVERFAHREN FÜR EFFIZIENTE PLANUNGEN
Kapitel 6

Modellierung von Stundenplanungsproblemen

6.1 Methoden:


Abb. 6.1: Eindimensionale und dreidimensionale Verschiedenheit beim Diffn - Constraint

Bei den meisten Ressourcenproblemen müssen sich die zu platzierenden Körper zum Beispiel im dreidimensionalen Raum in allen drei Dimensionen voneinander unterscheiden, um den Forderungen zu genügen. So dürfen zwei Veranstaltung in einem Stundenplan in der Regel nicht zur selben Zeit in selben Raum mit dem selben Dozenten stattfinden. Dieser Fall lässt sich nur mit zusätzlichen Ausschlussbedingungen (Constraints) spezifizieren. Wenn auch die Ausnahmen, wie zum Beispiel eine Veranstaltung
(Prüfung) soll zu einer Zeit in einem Raum und mit zwei Dozenten oder eine Veranstaltung (Abschlussprüfung) soll zu einer Zeit in zwei Räumen mit zwei verschiedenen Dozenten stattfinden, abgebildet werden sollen, müssen spezielle Modellierungen mit unter Umständen zurücknehmbaren zusätzlichen Constraint verwendet werden.

In dieser Arbeit soll die Modellierung von n-dimensionalen Körpern in n-dimensionalen Räumen mit Hilfe von zweidimensionalen Räumen vorgestellt werden. Zur Erläuterung des Vorgehens wird zuerst der dreidimensionale Raum gewählt, da dieser vorstellbar und visualisierbar ist. Zur Positionsbestimmung im dreidimensionalen Raum wird in jeder Dimension ein Wert \((x_i, y_i, z_i)\) benötigt, um für ein Objekt im Raum einen eindeutigen Ursprungspunkt festzulegen. Ist der dreidimensionale Raum endlich, sind die Dimensionen abgeschlossen, und man kann sich den Raum als Quader (Würfel) vorstellen. Die Hülle eines Quader (Würfels) umfasst alle Außenflächen des Körpers.


Die Hülle \(H(X,Y,Z)\) wird durch die Menge der zweidimensionalen Flächen \(X \times Y, X \times Z, X \times Y, Y \times Z, X \times Z\) und \(X \times Z\) gebildet, wenn man vom Ursprung im Uhrzeigersinn die Seitenflächen, die Deck- und Bodenfläche betrachtet.

\[
H(X,Y,Z) = \{X \times Y, X \times Y, X \times Z, Y \times Z, X \times Z, X \times Z\}
\]

Die reduzierte Hüle \(HR(X,Y,Z)\) ergibt sich aus der Hüle \(H(X,Y,Z)\), wenn die einander gleichen Flächen entfernt werden.
6.1. **Methoden:**

\[ HR(X,Y,Z) = \{ X \times Y, Y \times Z, X \times Z \} \]

Im Gegensatz zur den \( X, Y \) und \( Z \)-Koordinaten treten in der reduzierten Hüle \( HR(X,Y,Z) \) alle Koordinaten zweimal auf. Eine der zweidimensionalen Flächen der Hüle kann entfallen, ohne das Informationen verloren geben. Die vereinfachte Hüle \( HRV(X,Y,Z) \) besteht dann aus zwei zweidimensionalen Hüllenflächen \( X \times Y \) und \( X \times Z \), die eine gemeinsame Dimension \( X \) haben.

\[ HRV(X,Y,Z) = \{ X \times Y, X \times Z \} \]

6.1.1 **Beweis des Verfahrens zur Darstellung dreidimensionaler Körper in zwei zweidimensionalen Räumen**


Eine der benutzten Projektionen ist die **Normalprojektion** oder **Orthogonalprojektion**. Bei der Abbildung der Normalprojektion verlaufen die zueinander parallelen Projektionsstrahlen senkrecht zur Bildlebene \( E \). Die stark reduzierte Anschaulichkeit wird auf zwei Wegen kompensiert. Bei dem ersten Weg befüllt man markante Punkte oder Linien des dargestellten Objekts mit den Höhennoten über einer horizontalen Bezugsebene. Dies führt auf die Darstellungsweise der **kotierten Eintafelprojektion**. Diese Projektion findet vor allem bei der Projektierung von Erdbauten und Geländedarstellungen Anwendung [80].

Bei dem zweiten Weg ordnet man dem Normalriss im gleichen Zeichenfeld einen zweiten Normalriss zu. Dabei wird die Anordnung so getroffen, dass die Projektionsrichtungen und die Bildleben, die die beiden Normalrisse liefern, senkrecht aufeinander stehen. Das hier angedeutete Verfahren der Zweitafelprojektion oder der zugeordneten Normalrisse findet zum Beispiel im Maschinenbau und im Bauwesen Anwendung [80].

**Definition der Zweitafelprojektion**

Bei der Zweitafelprojektion wird das räumliche Objekt durch Normalprojektion auf zwei aufeinander senkrechten Ebenen \( E_1 \) und \( E_2 \) abgebildet. Diese Ebenen teilen den Raum in vier Quadranten ein, die durchnummeriert werden (siehe Abbildung 6.3 nach [80]). Ein räumliches Objekt im ersten Quadranten wird durch ein ersterprojizierendes Parallelstrahlenbündel \( s_1 \) senkrecht zu \( E_1 \) auf \( E_1 \) und ein zweitprojizierendes Parallelstrahlenbündel \( s_2 \) senkrecht zu \( E_2 \) auf \( E_2 \) abgebildet. Es entstehen auf diese Weise in zwei zunächst aufeinander senkrecht stehenden Ebenen zwei Normalprojektionen von einem Gegenstand. In der Ebene \( E_1 \) liegt der Grundriss und in der Ebene \( E_2 \) liegt der Aufriss. Eine Konvention besteht darin, die Grundrastafel horizontal und die Aufristafel vertikal anzunehmen. Für die weitere Bearbeitung legt man die Aufristafel in das Zeichenfeld und dreht anschließend die Grundrastafel um die waagerechte Rissachse \( x_1 \) in die vorgelegte Ebene. Nach dem Aufdrehen der Bildlebenen in das Zeichenfeld kommt der Aufriss des im ersten Quadranten angebrachten Körpers über und der Grundriss unter der Rissachse zu liegen. Der Grund- und Aufriss eines Punktes \( P \) liegen auf einer zur Rissachse senkrechten Geraden. Diese wird als **Ordnungslinei** oder **Ordnr** bezeichnet. Der Grundriss \( P' \) und der Aufriss \( P'' \) eines Punktes \( P \) befinden sich in **Mongescher Lage**, die auf die wissenschaftlichen Begründungen der darstellenden Geometrie Gaspard Monge (1746-1818) zurückgeht. Den Grundriss \( P' \) von \( P \) liefert ein erster Projektionstrahl \( s_1 \) durch \( P \), und entsprechend ergibt ein zweiter Projektionstrahl \( s_2 \) durch \( P \) den Aufriss \( P'' \) [80].

Beschränkt man sich darauf, die darzustellenden Objekte im ersten Quadranten anzubringen, so erscheinen stets der Aufriss über und der Grundriss unter der Rissachse des Zeichenfelds. Bei der Zweitafelprojektion überdecken sich somit im Zeichenfeld zwei Bildlebenen [80].
Handelt es sich bei dem abzubildenden Körper nun nicht mehr nur um einen Punkt \( P \), sondern um eine zweidimensionale Figur, so wird angenommen, dass das Bild jeder ebenen Figur, die parallel zur Projektionsebene im Raum liegt, bei einer Parallelprojektion mit dem Original kongruent ist.


Zusammenfassung: Liegt nun ein Körper in einem dreiaxigen kartesischen Koordinatensystem (seine Seitenflächen sind natürlich parallel zu den Koordinatenachsen), dann kann man ihn - wenn die Grundriss- die \( xz \)-Ebene und die Aufriss- die \( xy \)-Ebene darstellen - eindeutig mit Hilfe des Zweitafelprojektionsverfahrens in zwei zweidimensionale Koordinatensysteme überführen. Der Körper ist dann eindeutig bestimmt.

Wenn man diese Eigenschaft von dreidimensionalen Körpern auf auf \( n \)-dimensionale Körper überträgt, ergibt sich folgende Regel zur Abbildung \( n \)-dimensionaler Körpern mit \( n-1 \) zweidimensionalen Räumen:

Regel:
Ein \( n \)-dimensionaler Körper lässt sich mit \( n-1 \) zweidimensionalen Räumen abbilden, die eine gemeinsame Dimension haben, wenn \( n \geq 3 \) ist.

Die Regel zur Abbildung \( n \)-dimensionaler Körpern mit \( n-1 \) zweidimensionalen Räumen soll im folgenden Abschnitt bewiesen werden.

### 6.1.2 Beweis des Verfahrens zur Darstellung \( n \)-dimensionaler Körper in (\( n-1 \))- zweidimensionalen Räumen

Beweis zur Darstellung \( n \)-dimensionaler Körper in \( n-1 \) zweidimensionalen Räumen (mittels vollständiger Induktion): Die Richtigkeit der Regel kann mit Hilfe der vollständigen Induktion bewiesen werden.
6.1. Methoden:

Satz: Seien \( c \in \mathbb{Z} \) und \( Z(c) \) ein Zahlenabschnitt beginnend bei \( c \) mit \( N \) den natürlichen Zahlen und \( Z \) den ganzen Zahlen.
Sei \( A(n) \) eine Aussageform, die bei Belegung mit Elementen aus \( Z(c) \) jeweils zu einer Aussage wird. Dann gilt:

\[
(A(c) \land \forall (n \in Z(c)) (A(n) \rightarrow A(n+1))) \rightarrow \forall (n \in Z(c)) A(n).
\]

In Worten:
Gilt die Aussage für \( c \) (Anfangswert des Zahlenabschnitts) und folgt für alle Zahlen des Zahlenabschnitts, dass aus der Aussage für eine beliebige Zahl des Abschnitts die Aussage auch für den Nachfolger gilt, so ist die Aussage allgemein gültig für alle Zahlen aus dem Abschnitt [80].


Induktionsannahme:
Ein \( n \)-dimensionaler Körper lässt sich vollständig durch \( n \)-1 Bildebenen darstellen \((n > 2, n \in \mathbb{N})\).

Induktionsanfang:
Aussage gilt für \( n = 3 \). Wurde schon gezeigt in den Ausführungen zur Zweitafelprojektion.

\[
(n = 3) \ R(X,Y,Z) \rightarrow HRV = \{X \times Y, X \times Z\}, \ AnR(3) \rightarrow AnHRV(2)
\]

Induktionsschritt:
Fügt man nun - ausgehend vom \( n \)-dimensionalen Körper - eine Dimension zum Körper hinzu, so muss sich auch die Anzahl der Bildebenen erhöhen. Diese steigt genauso wie die Anzahl der zum Körper hinzukommenden Dimensionen:
Es wird bei der Dimensionserhöhung des Körpers durch Festhalten der Koordinate \( x \) (wie im Beispiel in der Abbildung 6.4 (könnte natürlich auch jede beliebige andere Koordinate sein)) und Hinzufügen der neuen Dimensionsrichtung jeweils eine neue Bildebene geformt, die in der \( x \) (und-neu-entstandene-Dimension)-Ebene liegt. Diese Ebene liegt dann wieder senkrecht zu allen bisherigen Bildebenen und die Projektionsdaten des Körpers sind mit Aussage zur Zweitafelprojektion vollständig. Da die Anzahl der Bildebenen für den \( n \)-dimensionalen Körper \( n \)-1 entspricht, wird ein \( n+1 \) dimensional der Körper also vollständig durch \( n \)-1 Bildebenen dargestellt.

\[
(n+1 = 4) \ R(V,X,Y,Z) \rightarrow HRV = \{V \times X, V \times Y, V \times Z\}, \ AnR(4) \rightarrow AnHRV(3)
\]

Induktionschluss:
Mit dem Induktionsanfang, dem Induktionsschritt und dem folgenden Satz zur vollständigen Induktion ist somit die Annahme gezeigt.

\[
AnR(n) \rightarrow AnHRV(n - 1)
\]

Beispiel:
Für den vierdimensionalen Raum ergeben sich die Hüllen \( H, HR \) und \( HRV \) gegeben:

\[
H(V,X,Y,Z) = \{V \times X, X \times Y, Y \times Z, V \times Y, V \times Z, V \times X, X \times Y, Y \times Z, V \times Y, V \times Z, X \times Z, X \times Z\} \\
HR(V,X,Y,Z) = \{V \times X, X \times Y, Y \times Z, V \times Y, V \times Z, X \times Z\} \\
HRV(V,X,Y,Z) = \{V \times X, V \times Y, V \times Z\}
\]
Abbildun 6.4: Visualisierung der zweidimensionalen Flächen für den vierdimensionalen Raum

Die Visualisierung der zweidimensionalen Ebenen für den vierdimensionalen Raum ist in Abbildung 6.4 dargestellt. In der Abbildung sind nur die positiv orientierten Ebenen dargestellt.

Im nächsten Abschnitt soll das Verfahren für das vierdimensionale Ressourcenproblem der Erstellung von Stundenplänen bei Weiterbildungseinrichtungen angewendet werden.

6.2 Vierdimensionales Ressourcenplanungsproblem


Bei der kombinierten Ressourcendimension (siehe Abbildung 6.5 rechts) gibt es in einer Ressourcendimension sowohl die Kurse als auch die Gruppen in den einzelnen Kursen. Hingegen ist bei der n-fach aufgelösten Ressourcendimension (siehe Abbildung 6.5 links) für jeden Kurs eine neue Ressourcendimension mit den Gruppen des Kurses angelegt worden. Die n-fach aufgelöste Ressourcendimension setzt sich demzufolge aus n-Kurs-Ressourcendimensionen (RD4 + n) zusammen, mit \( n \in \mathbb{N} \) der Anzahl der Kurse. In der kombinierten Ressourcendimension Kurs (RD4) werden die einzelnen Gruppen der Kurse übereinander in der Ressourcendimension angeordnet und ein bestimmter Teil der Ressourcendimension RD4 wird einem bestimmten Kurs zugeordnet.

Bei der aufgelösten Ressourcendimensionen (RD4 + n) wurden für die Kurse getrennte Ressourcendimensionen eingeführt, um in jeder Kurs-Ressourcendimension die dazugehörigen Gruppen des Kurses abbilden zu können. Hierbei erhöhen sich die Dimensionen des Ressourcenplanungsproblems um die Anzahl jedes einzelnen abgebildeten Kurses. Das Ressourcenplanungsproblem besteht dann aus n-Dimensionen der Kurse mit \( n \) der Anzahl der Kurse, der Dimension der Dozenten, der Dimension der Räume und der Dimension der Zeit. Da bei den aufgelösten Kurs-Ressourcendimensionen (RD4 + n) die Gruppen eines Kurses auf eine eigene Ressourcendimension aufgetragen werden und für jeden weiteren Kurs eine eigene Ressourcendimension benutzt wird, entsteht dann daraus ein \((n_{Kurs} + 3)\)-Ressourcenplanungsproblem. In der Abbildung 6.5 sind die beiden Varianten der kombinierten (gestrichelte Linien mit der Bezeichnung RD4) und aufgelösten Ressourcendimension Kurs (durchgehende Linie mit den Bezeichnungen RD4+n) zu sehen.

Im folgenden Abschnitt wird die Modellierung des Stundenplanungsproblems für Weiterbildungseinrichtungen bei Anwendung der Constraints zur Definition des Constraint-Netzes erläutert.

### 6.3 Modellierung mit Constraints

In diesem Abschnitt wird die Modellierung des Stundenplanungsproblems für Weiterbildungseinrichtungen durch Constraint erläutert. Die Modellierung erfolgt zum einen im kommerziellen constraintlogischen Programmiersystem CHIP und zum anderen mit dem neuen Constraintlösers \( CS_{1-4-MR} \). Die Realisierung der Modellierung ist in beiden Systemen nahezu identisch und im Folgenden einheitlich beschrieben. Es wird auf die vorhandenen Unterschiede an den entsprechenden Stellen speziell eingegangen.

#### 6.3.1 Domäne


Der Planungshorizont erstreckt sich bei Stundenplänen in Schulen über eine Woche und in Weiterbildungseinrichtungen meist über ein bis zwei Jahre. Bei einer minutenbasierten Modellierung werden für die Abbildung einer Woche 60 Minuten * 24 Stunden = 1440 Tagminuten * 7 Tage = 10080 diskrete Werte benötigt und für die Abbildung von zwei Jahren 10080 Wochenminuten * 53 Wochen * 2 Jahre


### 6.3.2 Einschränkung der "Nachtzeiten" - Feiertage, Urlaub, Pausen


| Speicherverbrauch bei Modellierung mit diff()-Constraint und Notin-Constraint |
|---------------------------------|-----------------|-----------------|-----------------|
| Kriterium                       | Constraint      | Einsparung notin() zu diff() |
| Lokaler Speicher durch Constraint belegt | 1221 kB | 976 kB | - 245 kB | - 20 |
| Globaler Speicher durch Constraint belegt | 4489 kB | 332 kB | - 4157 kB | - 93 |
| Trail                           | 485 kB | 433 kB | - 52 kB | - 11 |
| Gesamter Speicher               | 6195 kB | 1714 kB | - 4454 kB | - 72 |
| Prozesszeit                     | 400 ms | 460 ms | - 20 ms | - 4 |

6.3. MODELLIERUNG MIT CONSTRAINTS

gelöst werden. Aufgrund dieser Tatsache wird die Modellierung der Feier- und Urlaubstage im Programmiersystem CHIP mit `notin()`-Constraints umgesetzt.


6.3.3 Überschneidungsfreiheit


6.3.4 Reihenfolgebeziehungen


Das Reihenfolgeconstraint RC1 wurde in der Programmiersprache CHIP mit den in dieser Programmiersprache zur Verfügung stehenden Mitteln neu hinzu gefügt. Hierbei erfolgte die Implementierung direkt in der Programmiersprache ohne Nutzung einer Schnittstelle. Im neuen Constraintlöscher $CS_{NC-MR}$ ist das Reihenfolgeconstraint auch vorhanden.
Kapitel 7

Anwendungsbeispiel

7.1 Stundenplanung für Kurse


Des Weiteren sollen die Veranstaltungen möglichst gleichmäßig und ohne lokale Anhäufungen verteilt werden, wobei der Grad der Anhäufung individuell anpassbar sein muss. Zum Beispiel kann es in Weiterbildungseinrichtungen gewünscht sein, dass an einem Tag 8 Stunden Englisch hintereinander stattfinden, um die Kosten für den Dozenten gering zu halten, in Schulen wäre solche Art von Unterricht aber unterdurchschnittlich. Dort können maximal Doppelstunden gebildet werden. Außerdem sollte die
Anzahl der Freistunden minimal gehalten werden.

Abbildung 7.1: Graphische Interpretation der Abhängigkeiten

Der eigentliche Planungsprozess sollte so schnell wie möglich sein und jederzeit abgebrochen werden können. Außerdem soll er auf üblichen Desktopsystemen und eventuell auch auf mobileren Geräten wie Laptops zum Einsatz kommen können und dort in akzeptabler Zeit Ergebnisse liefern.

Als Problembeispiel wird die Planung von Kursen an einer Weiterbildungseinrichtung vorgestellt. Es ist ein vierdimensionales Planungsproblem mit einer Zeitdimension und drei weiteren Ressourcendimensionen.


Problembeschreibung


- Pausen zwischen zwei Veranstaltungen,
- Wegzeiten zwischen zwei Veranstaltungen an zwei verschiedenen Orten und
- Veranstaltungen für mehrere Kurse zur gleichen Zeit an einem Ort.

Neben den eben genannten Bedingungen, die immer erfüllt sein müssen (harte Constraints), gibt es auch Bedingungen, die möglichst erfüllt sein sollen (weiche Constraints). Zu diesen Bedingungen zählen:

- die Beachtung der Zeitwünsche,
- die Beachtung der Raumwünsche und
- die Beachtung der Dozentenwünsche.
Problembeispiel

Es wird nun das Problem der zeitlichen, räumlichen und personellen Planung von Kursen an einer Weiterbildungseinrichtung beschrieben. Im Planungsproblem gelten die folgenden Rahmenbedingungen des Diskursbereichs:

- **R 1:** Es gibt zwei Typen von Veranstaltungen: Vorlesungen und Praktika.
- **R 2:** Die Vorlesungen werden entweder in einer Gruppe oder in mehreren Gruppen von maximal 15 Kursteilnehmern durchgeführt.
- **R 3:** Jeder Teilnehmer eines Kurses ist einer Gruppe zu geordnet.
- **R 4:** In jedem Kurs gibt es 2 bis 4 einzelne Gruppen (In der Modellierung ist die Anzahl der Gruppen nach oben unbeschränkt.).
- **R 5:** Die Lehrgebäude der Weiterbildungseinrichtung befinden sich an zwei verschiedenen Orten in der Stadt.
- **R 6:** Die Praktika werden im Ausland in verschiedenen Ländern durchgeführt.

Das Problem umfasst die folgende harte Constraints (Bedingungen):

- **C 1:** Die Veranstaltungen können zu jeder Minute beginnen und können unterschiedliche, vorgegebene Zeitspannen dauern.
- **C 1.1:** Die Veranstaltungen dürfen nicht an Feiertagen stattfinden.
- **C 1.2:** Die Pausen zwischen den Veranstaltungen sind einzuhalten.
- **C 2:** Für jede Veranstaltung existieren zeitliche Einschränkungen bezüglich ihrer Durchführbarkeit, die sich aus dem Kursende und der Reihenfolge aufeinander aufbauender Veranstaltungen ergeben.
- **C 3:** Die Vorlesungen und Praktika innerhalb jeder einzelnen Gruppe eines Kurses dürfen sich zeitlich, räumlich und personell nicht überlappen.
- **C 4:** Die Anzahl der Veranstaltungen einer Faches, die zu einer Zeit abgehalten werden können, ist begrenzt durch die Anzahl der zur Verfügung stehenden Spezialkabinette.
- **C 5:** Den einzelnen Veranstaltungen sind im Plan Dozenten und Räumen konfliktfrei zuzuordnen.
- **C 6:** Einige Veranstaltungen können nur in bestimmten Spezialräumen mit besonderer Ausstattung (Projektoren, usw.) stattfinden. In bestimmten Fällen steht nur ein Raum mit der passenden Ausstattung zur Verfügung.
- **C 7:** Viele Veranstaltungen können nur von bestimmten Dozenten mit besonderer fachlicher Qualifikation gehalten werden.
- **C 8:** Die Wege zwischen den verschiedenen Veranstaltungsorten müssen im Plan beachtet werden. Jede Veranstaltung soll pünktlich erreichbar sein.

Das Problem umfasst die folgenden weichen Constraints (Bedingungen):

- **C 9:** Die Wünsche für die Startzeiten der Veranstaltungen sind möglichst zu berücksichtigen.
- **C 10:** Die Wünsche für den Dozenten der Veranstaltungen sind möglichst zu berücksichtigen.
- **C 11:** Die Wünsche für die Räume der Veranstaltungen sind möglichst zu berücksichtigen.

Die harten Constraints müssen erfüllt werden. Die weichen Constraints sollten nach Möglichkeit erfüllt werden, müssen aber nicht erfüllt werden.
Planungssysteme ConTime 1.0 und Kursplan 1.0

Das gegebene Problembeispiel der Planung von Kursen an einer Weiterbildungseinrichtung wurde mit zwei Planungssystemen bearbeitet, um die Vor- und Nachteile der genutzten Constraintlöscher zu zeigen. Das Planungssystem **ConTime 1.0** nutzt die constraintlogische Programmiersprache CHIP der französischen Firma Cosytec. Für die Umsetzung des Problembeispiels werden die vorhandenen Constraints des finite domain - Constraintlösers **CS_FD** (FD-Solvers) in der constraintlogischen Programmiersprache CHIP genutzt. Zusätzlich zu den vorhandenen Basis- und globalen Constraints der Programmiersprache CHIP wurden die Ressourcenconstraints **RC1** und **RC2** integriert. Für die Integration in die constraintlogische Programmiersprache CHIP wurde das Ressourcenconstraint in der constraintlogischen Programmiersprache programmiert, um eine hohe Performance im Lösungsprozess zu erreichen. Die von der constraint-logischen Programmiersprache zur Verfügung gestellten Schnittstellen CLIC und EMC sind nur für Prozesssteuerung und nicht für umfangreiche Datenaustausche ausgelegt. Die Benutzeroberfläche von ConTime 1.0 ist in der von der constraintlogischen Programmiersprache zur Verfügung gestellten Grafikbibliothek realisiert worden.

Das Planungssystem **Kursplan 1.0** nutzt den neuen Constraintlösher für Multiresourcenprobleme **CS_FD-MR** aus dem gleichnamigen Kapitel in dieser Arbeit. Im Planungssystem Kursplan 1.0 wurde die Benutzeroberfläche in der Programmiersprache .NET und der Constraintlösher **CS_FD-MR** in der Programmiersprache C++ implementiert.

Im Planungssystem ConTime 1.0 werden die folgenden Constraints des Constraintlösers **CS_FD** für die Modellierung des oben genannten Planungsproblems genutzt:

- die Constraints = und <,
- die globalen Constraints **diffn()**, **notin()**, 
- die globalen Reihenfolgeconstraints **RC1()** und **RC2()**

Das zweite Planungssystem Kursplan 1.0 nutzt die folgenden Constraints des Constraintlösers **CS_FD-MR** für die Abbildung des gegebenen Planungsproblems:

- die Constraints = und <,
- die globalen Constraints **diffn2D()**, **notin()**, 
- die globalen Reihenfolgeconstraints **RC1()** und **RC2()**

**Modellierung der Bedingungen mit den vorhandenen Constraints**


**Randbedingungen**

Bei der Modellierung der Randbedingungen wird zwischen den unterschiedlichen Typen von Veranstaltungen unterschieden, da zum Beispiel Vorlesungen mit einer Raumzuordnung und die Praktika (im Ausland) ohne eine Raumzuordnung geplant werden (R1, R6). Die Veranstaltungen können eine variable Dauer haben.

Zur Abbildung der Randbedingungen gehört die Definition der Domänenvariable. Entweder beinhaltet die Domänenvariable nur einen Wert, wenn eine eindeutige Wertzuordnung erfolgen soll, zum Beispiel der Zuordnung einer Veranstaltung / eines Teilnehmers eines Kurses zu einer Gruppe (R2, R3) oder es sind in der Domänenvariable nur die benutzbaren Werte nach der Definition in der Randbedingung enthalten. Der letzte Fall ist zum Beispiel bei der Zuordnung einer Veranstaltung / eines Kurses zu
mehreren Gruppen gegeben (R2, R4). Gleiches gilt auch bei der Abbildung von mehreren Orten, an denen die Veranstaltungen alternativ stattfinden können (R5).

### Umsetzung der Randbedingungen

<table>
<thead>
<tr>
<th>Bedingung</th>
<th>Constraint ConTime 1.0</th>
<th>Constraint Kursplan 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Domänenvariable</td>
<td>Domänenvariable</td>
</tr>
<tr>
<td>R2</td>
<td>Domänenvariable</td>
<td>Domänenvariable</td>
</tr>
<tr>
<td>R3</td>
<td>Domänenvariable mit einem Wert</td>
<td>Domänenvariable mit einem Wert</td>
</tr>
<tr>
<td>R4</td>
<td>Domänenvariable</td>
<td>Domänenvariable</td>
</tr>
<tr>
<td>R5</td>
<td>Domänenvariable</td>
<td>Domänenvariable</td>
</tr>
<tr>
<td>R6</td>
<td>Domänenvariable</td>
<td>Domänenvariable</td>
</tr>
</tbody>
</table>

Die obige Tabelle zeigt, dass die Modellierung der Randbedingungen in beiden Systemen gleich erfolgt ist.

### Harte- und weiche Constraints


Zur Modellierung der harten Constraints gehört auch die Festlegung der Domänenlänge, zum Beispiel bei der Abbildung der Genauigkeit (C1) und bei der Abbildung von speziellen Ressourcenausprägungen bei Dozenten und Räumen (C6, C7). Die Modellierung von gespernten Domänenbereichen erfolgt durch die Herausnahme von Werten aus der Domäne mit dem Constraint `notin()`, zum Beispiel bei der Abbildung der Feiertage (C 1.1) und bei der Berücksichtigung von Pausenzeiten (C 1.2). Alle Bedingungen, die eine überlappende Freiheit, eine individuente Zuordnung sicher, werden mit dem globalen Constraint `diffn()` oder `diffn2D()` realisiert (C3 - C5, C8).


### Modellierung der Constraints

<table>
<thead>
<tr>
<th>Bedingung</th>
<th>verwendetes Constraint in ConTime 1.0</th>
<th>verwendetes Constraint in Kursplan 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Domänengröße</td>
<td>Domänengröße</td>
</tr>
<tr>
<td>C1.1</td>
<td><code>notin()</code></td>
<td><code>notin()</code></td>
</tr>
<tr>
<td>C1.2</td>
<td><code>notin()</code></td>
<td><code>notin()</code></td>
</tr>
<tr>
<td>C2</td>
<td>Domänenvariable / RC1 / RC2 / = / &lt;</td>
<td>Domänenvariable / RC1 / RC2 / = / &lt;</td>
</tr>
<tr>
<td>C3</td>
<td><code>diffn()</code></td>
<td><code>diffn2D()</code></td>
</tr>
<tr>
<td>C4</td>
<td><code>diffn()</code></td>
<td><code>diffn2D()</code></td>
</tr>
<tr>
<td>C5</td>
<td><code>diffn()</code></td>
<td><code>diffn2D()</code></td>
</tr>
<tr>
<td>C6</td>
<td>Domänengröße</td>
<td>Domänengröße</td>
</tr>
<tr>
<td>C7</td>
<td>Domänengröße</td>
<td>Domänengröße</td>
</tr>
<tr>
<td>C8</td>
<td><code>diffn()</code></td>
<td><code>diffn2D()</code></td>
</tr>
<tr>
<td>C9</td>
<td>Suche</td>
<td>Suche</td>
</tr>
<tr>
<td>C10</td>
<td>Suche</td>
<td>Suche</td>
</tr>
<tr>
<td>C11</td>
<td>Suche</td>
<td>Suche</td>
</tr>
</tbody>
</table>
Die Modellierung der Randbedingungen, das heißt der harten und weichen Constraints, ist in beiden Planungssystem ConTime 1.0 und Kursplan 1.0 ähnlich realisiert. Der Unterschied liegt nur in der Nutzung des globalen Constraints diffn() in ConTime 1.0 und der alternativen Anwendung des globalen Constraints diffn2D() im System Kursplan 1.0. Wodurch eine gute Vergleichbarkeit der beiden Planungssysteme gegeben ist. Ein Unterschied in der Planungsqualität und in der Planungsgeschwindigkeit kann somit nur aus der internen Realisierung der Constraintlöser resultieren.

Im folgenden Abschnitt werden die durchgeführten Tests mit den beiden Planungssystem ConTime 1.0 und Kursplan 1.0 erläutert.
Kapitel 8

Ergebnisse und Bewertung

8.1 ConTime 1.0


<table>
<thead>
<tr>
<th>Kurs</th>
<th>Problem</th>
<th>Anzahl</th>
<th>Einheiten</th>
<th>Wochen</th>
<th>Backtracking</th>
<th>Domänenzerlegung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kurse</td>
<td>C.-Asetzen/sec</td>
<td>Suche/sec</td>
<td>LA/RC[sec]</td>
<td>Faktor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>1</td>
<td>72</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>74</td>
<td>6</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>176</td>
<td>10</td>
<td>0</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>1</td>
<td>214</td>
<td>12</td>
<td>15</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
<td>224</td>
<td>10</td>
<td>16</td>
<td>92</td>
</tr>
<tr>
<td>7</td>
<td>1,2</td>
<td>2</td>
<td>400</td>
<td>19</td>
<td>22</td>
<td>174</td>
</tr>
<tr>
<td>8</td>
<td>1,2,9</td>
<td>3</td>
<td>614</td>
<td>22</td>
<td>36</td>
<td>344</td>
</tr>
</tbody>
</table>

Tabelle 8.1: Rechenzeit zur Ermittlung der Lösungen mit den zwei Methoden "Backtracking" und "Domänenzerlegung mit Look ahead und dem Reihenfolgeconstraint RC1/RC2" (1)

Der erste Kurs t gehört nicht mit zur Serie des praktischen Beispiels. Mit dem Kurs t wurde die Zeit ermittelt, die vom Algorithmus zur Domänenzerlegung benötigt wird, um die 75 Wochen zu bearbeiten. Die Zeit zur Bearbeitung der 75 Wochen beträgt 9 Sekunden und stellt den Zusatzaufwand dar, der durch die Benutzung des Algorithmus zur Domänenzerlegung entsteht. Vergleichbar werden die Zeiten erst nach den 9 Sekunden, deshalb sind erst ab diesem Betrag die Faktoren zwischen der

In der zweiten Tabelle sind die größeren Planungsprobleme von 700 bis unter 2500 einzuplanenden Einheiten dargestellt.

Die ermittelten Messergebnisse aus den beiden Tabellen 8.1 und 8.2 wurden zusammengefasst und in

<table>
<thead>
<tr>
<th>Kurse</th>
<th>Problem</th>
<th>Anzahl</th>
<th>Einheiten</th>
<th>Wochen</th>
<th>C.absatz [sec]</th>
<th>Suche [sec]</th>
<th>Domänenzerlegung</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4</td>
<td>1</td>
<td>751</td>
<td>36</td>
<td>216</td>
<td>1750</td>
<td>60</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>1</td>
<td>776</td>
<td>39</td>
<td>190</td>
<td>2470</td>
<td>53</td>
</tr>
<tr>
<td>11</td>
<td>1,4</td>
<td>2</td>
<td>927</td>
<td>39</td>
<td>229</td>
<td>2084</td>
<td>82</td>
</tr>
<tr>
<td>12</td>
<td>2,5</td>
<td>2</td>
<td>1000</td>
<td>44</td>
<td>215</td>
<td>3071</td>
<td>82</td>
</tr>
<tr>
<td>13</td>
<td>4,5</td>
<td>2</td>
<td>1527</td>
<td>55</td>
<td>439</td>
<td>6580</td>
<td>130</td>
</tr>
<tr>
<td>14</td>
<td>1,2,4,5</td>
<td>4</td>
<td>1927</td>
<td>57</td>
<td>500</td>
<td>9367</td>
<td>197</td>
</tr>
<tr>
<td>15</td>
<td>1,2,3,4,5,7,8,9</td>
<td>7</td>
<td>2359</td>
<td>75</td>
<td>–</td>
<td>–</td>
<td>253.322</td>
</tr>
</tbody>
</table>

Tabelle 8.2: Rechenzeit zur Ermittlung der Lösungen mit den zwei Methoden „Backtracking“ und „Domänenzerlegung mit Look ahead und dem Reihenfolgeconstraint RC1/RC2“ (2)

einem Säulendiagramm in Abbildung 8.1 visualisiert. Bei größeren Problemen aus dem praktischen

Abbildung 8.1: Darstellung der Messergebnisse von ConTime 1.0 im Säulendiagramm des Athlon-Prozessors

Beispielen an denen der Kurs 5 ohne Kurs 4 beteiligt ist, liegt der Faktor zwischen ca. 37 bis 47. In dem Beispiel in dem die Kurs 4 und 5 miteinander kombiniert werden, liegt der Faktor bei ca. 50. Der Wert liegt über den beiden anderen Wertepaaren (25-29) und (37-47). Weil beide Kurse gemeinsame Veranstaltungen haben, die mit einander verknüpft sind, wird die Zeit von 4 und von 5 erhöht, denn die Bedingungen für die gemeinsamen Veranstaltungen für beide Kurse müssen in beiden Kursen geprüft werden. Da es nicht möglich war, mehr als 1,6 GB Arbeitsspeicher dem Prozess des Planungssystems in CHIP auf dem Testsystem zur Verfügung zu stellen, konnte der Zeitaufwand bei 2359 Veranstaltungen und unzerlegten Domänen nicht gemessen werden.

Das Säulendiagramm in Abbildung 8.1 verzerrt die Messergebnisse in der Hinsicht, dass es den Eindruck eines exponentiellen steigenden Aufwandes vermittelt. Letztendlich liegt dies an dem ungleichmäßigen Verhältnis der Messergebnisse. Zur Verdeutlichung sollen die Liniendiagramme in Abbildung 8.2 dienen. Die X-Achse wird nun proportional zur Anzahl der Veranstaltungen dargestellt. Zur Bestätigung, dass der Aufwand bei der Domänenzerlegung (zerlegter Domänen) annähernd linear zur Zeit und zur Anzahl der Ressourcen steigt, sind ergänzende Trendlinien der Form \( f(x) = ax^b \) erfasst. Die Trendlinie der Form

\[
y = 0.264005x^{0.848720}
\]

Abbildung 8.2: Darstellung der Messergebnisse von ConTime 1.0 Liniendiagramm

\( f(x) = ax^b \) wird für Backtracking (unzerlegter Domänen) mit der Funktion \( y = 0.000153x^{2.402534} \) und die Trendlinie für die Domänenzerlegung (zerlegter Domänen) mit der Funktion \( y = 0.264005x^{0.848720} \) beschrieben. Bei der Funktion für das Backtracking ist der Wert von \( a \) mit 0,000153 sehr gering. Wohingegen der \( \beta \)-Wert viel stärker mit 2,402534 ins Gewicht fällt. Bei der Funktion für die Domänenzerlegung sind der \( a \)- und der \( \beta \)-Wert funktionsverlaufbestimmend. Der Aufwand für die Berechnung steigt mit Backtracking und unzerlegten Domänen bei zunehmender Anzahl von Veranstaltungen (Problemgröße) stärker an als bei der Berechnung mit Domänenzerlegung und zerlegten Domänen. Im Exponenten ist es ein Unterschied von 1:2,83. Dieser Unterschied im Exponenten von 2,83 führt schon bei kleinen Beispielen (bemessen auf den praktischen Anwendungsfall) zu Unterschieden in der Bearbeitungszeit von 1:5, bei mittleren Problemen von 1:30 und bei großen Problem von 1:50. Somit führt die Zerlegung der Domänen trotz des notwendigen Zusatzaufwandes für die Domänen, der bei kleinen Beispielen stärker ins Gewicht fällt, ab einer bestimmten Domänengröße zu gravierenden Vorteilen in der Bearbeitungszeit.

Die in der Abbildung 8.1 und 8.2 dargestellten Zeiten wurden für die constraint-logische Programmier-
sprache CHIP auf einem Athlon-Prozessor ermittelt. Die zusätzlich ermittelten Zeiten von ConTime 1.0 mit und ohne Ressourcenwünsche auf dem alternativen Pentium-Prozessor können aus der Tabelle 8.3 entnommen werden.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>mit Ressourcenwünschen</th>
<th></th>
<th></th>
<th>unzerlegter Domänen</th>
<th></th>
<th></th>
<th>verplant</th>
<th></th>
<th></th>
<th>verplant</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>72</td>
<td>6,749</td>
<td>72</td>
<td>0,832</td>
<td>1,473</td>
<td>72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>72</td>
<td>6,92</td>
<td>72</td>
<td>1,362</td>
<td>1,983</td>
<td>72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>74</td>
<td>8,582</td>
<td>74</td>
<td>1,673</td>
<td>7,931</td>
<td>74</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>144</td>
<td>11,725</td>
<td>72</td>
<td>2,774</td>
<td>6,49</td>
<td>144</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>176</td>
<td>12,258</td>
<td>176</td>
<td>7,941</td>
<td>42,591</td>
<td>176</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>214</td>
<td>19,327</td>
<td>211</td>
<td>0</td>
<td>7,134</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>224</td>
<td>15,702</td>
<td>224</td>
<td>10,185</td>
<td>71,383</td>
<td>224</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1, 2</td>
<td>400</td>
<td>24,215</td>
<td>400</td>
<td>18,707</td>
<td>136,276</td>
<td>400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1, 2, 9</td>
<td>614</td>
<td>43,272</td>
<td>611</td>
<td>0</td>
<td>127,664</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>751</td>
<td>38,775</td>
<td>751</td>
<td>209,982</td>
<td>164,982</td>
<td>751</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>776</td>
<td>40,048</td>
<td>776</td>
<td>370,652</td>
<td>214,487</td>
<td>776</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1, 4</td>
<td>927</td>
<td>53,006</td>
<td>927</td>
<td>402,3</td>
<td>224,155</td>
<td>927</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>2, 5</td>
<td>1000</td>
<td>56,832</td>
<td>1000</td>
<td>325,969</td>
<td>232,4</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>4, 5</td>
<td>1527</td>
<td>98,612</td>
<td>1527</td>
<td>1190,863</td>
<td>6642</td>
<td>1527</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1, 2, 4, 5</td>
<td>1927</td>
<td>130,668</td>
<td>1927</td>
<td>730</td>
<td>8290</td>
<td>1927</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tabelle 8.3: Messergebnisse von ConTime 1.0 und vom System Pentium 4M, 1,4 Ghz, 256 MB RAM mit und ohne Ressourcenwünsche

Auch hier muss wieder auf den letzten Wert beim Backtracking mit unzerlegten Domänen verzichtet werden, da dem Prozess des Planungssystems nicht mehr als 16 GB Arbeitsspeicher zur Verfügung gestellt werden konnten. Im Säulendiagramm in Abbildung 8.3 werden die Zeiten für den Berechnungsaufwand graphisch visualisiert.

Die Auswertung der Zeiten lässt die Vermutung zu, dass bei der Domänenzerlegung der Aufwand nahezu linear steigt und bei dem Backtracking der Aufwand annähernd exponentiell ansteigt. Zur Bestätigung, dass der Aufwand bei der Domänenzerlegung fast linear zur Zeit und zur Anzahl der Ressourcen steigt, sind ergänzend Trendlinien der Form \( f(x) = ax^3 \) (potentiell) dargestellt. Zusätzlich
werden auch für die Domänenzerlegung Trendlinien der Form \( f(x) = \gamma x^2 + \delta x \) (polynom) erfasst, die für die Vergleiche der Systeme untereinander ergänzend benutzt werden. In dem folgenden Liniendiagramm in der Abbildung 8.4 sind die einzelnen Zeiten für die Lösung und die Trendlinien aufgetragen. Die Trendlinie für das Backtracking (unzerlegter Domänen) mit Wünschen wird mit der Funktion \( y = 0,000003x^{2,58392} \) beschrieben und für das Backtracking (unzerlegter Domänen) ohne Wünsche mit der Funktion \( y = 0,000031x^{2,493377} \). Die Trendlinien für die Domänenzerlegung (zerlegter Domänen) mit Wünschen haben die Funktionen (potentiell) \( y = 0,1778x^{0,8088} \) (polynom) \( y = 0,00024x^2 + 0,034103x \) und für die Domänenzerlegung (zerlegter Domänen) ohne Wünsche die Funktionen (potentiell) \( y = 0,1477x^{0.8816} \) (polynom) \( y = 0,000011x^2 + 0,034506x \). Wenn man die potentielle Funktion für das Backtracking (unzerlegter Domänen) mit Wünschen des Pentium-Prozessors mit der potentiellen Funktion für das Backtracking (unzerlegter Domänen) des Athlon-Prozessors vergleicht, ergibt sich nur ein geringer Unterschied in den \( \alpha \)- und \( \beta \)-Werten der Funktionen und ein nur wenig geringerer Aufwand in den Abarbeitungszeiten. Wenn die potentielle Funktion für die Domänenzerlegung des Pentium-Prozessors mit der potentiellen Funktion für die Domänenzerlegung des Athlon-Prozessors vergleicht, zeigt sich auch hier, dass die Funktionen kaum voneinander abweichen und die Abarbeitungszeiten beim Pentium-Prozessor geringfügig besser sind.

8.2 Kursplan 1.0

Im folgenden Abschnitt werden die Testergebnisse des auf dem neuen Constraintlöser CS/4-MR basierenden Planungssystems Kursplan 1.0 beschrieben.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Kurs 8</td>
<td>72</td>
<td>2</td>
<td>3</td>
<td>72</td>
</tr>
<tr>
<td>Kurs 7</td>
<td>72</td>
<td>2</td>
<td>3</td>
<td>72</td>
</tr>
<tr>
<td>Kurs 3</td>
<td>74</td>
<td>2</td>
<td>2</td>
<td>36</td>
</tr>
<tr>
<td>Kurs 6</td>
<td>144</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Kurs 1</td>
<td>176</td>
<td>3</td>
<td>0</td>
<td>88</td>
</tr>
<tr>
<td>Kurs 9</td>
<td>214</td>
<td>5</td>
<td>6</td>
<td>106</td>
</tr>
<tr>
<td>Kurs 2</td>
<td>224</td>
<td>4</td>
<td>4</td>
<td>99</td>
</tr>
<tr>
<td>Kurs 1, 2, 9</td>
<td>400</td>
<td>6</td>
<td>7</td>
<td>187</td>
</tr>
<tr>
<td>Kurs 1, 2, 4, 5</td>
<td>614</td>
<td>8</td>
<td>10</td>
<td>293</td>
</tr>
<tr>
<td>Kurs 4</td>
<td>751</td>
<td>11</td>
<td>13</td>
<td>489</td>
</tr>
<tr>
<td>Kurs 5</td>
<td>776</td>
<td>13</td>
<td>16</td>
<td>503</td>
</tr>
<tr>
<td>Kurs 1, 4</td>
<td>927</td>
<td>14</td>
<td>18</td>
<td>577</td>
</tr>
<tr>
<td>Kurs 2, 5</td>
<td>1000</td>
<td>14</td>
<td>20</td>
<td>602</td>
</tr>
<tr>
<td>Kurs 4, 5</td>
<td>1527</td>
<td>22</td>
<td>31</td>
<td>992</td>
</tr>
<tr>
<td>Kurs 1, 2, 4, 5</td>
<td>1927</td>
<td>28</td>
<td>38</td>
<td>1179</td>
</tr>
<tr>
<td>Kurs 1, 2, 3, 4, 5, 7, 8, 9, 10</td>
<td>2359</td>
<td>42</td>
<td>54</td>
<td>1465</td>
</tr>
</tbody>
</table>

Tabelle 8.4: Mesergebnisse von Kursplan 1.0 und von System Athlon, 700 Mhz, 256 MB RAM

Es werden drei Fälle der Planung betrachtet:

- das Einplanen von Veranstaltung mit Wünschen ohne weitere Restriktionen,
- das Einplanen von Veranstaltungen mit Wünschen und möglichen (präferierten) Ressourcenausprägungen und
- die Planung von Veranstaltungen unter Beachtung von Wünschen, Präferenzen und allen sonst noch vorgegebenen Bedingungen bezüglich der verfügbaren Ressourcen, wie zum Beispiel Zeitpunkte, Räume und Dozenten.

Bei der Planung mit Wünschen ohne weitere Restriktionen wurden nicht alle Veranstaltungen eingeplant. Dies liegt daran, dass in den Testdaten keine Wünsche angegeben wurden, die bei der Planung nach Wünschen unbedingt benötigt werden, denn bei dieser Planung wird versucht, die Veranstaltungen an ihren Wunschzeiten, in den Wunschräumen und mit den Wunschdozenten einzuplanen. Wenn die Wünsche nicht konsistent sind oder fehlen, wird die Veranstaltung nicht eingeplant. Mit dieser Methode können schon vorhandene oder von Hand erstellte Pläne überprüft werden. In diesem Fall wurden trotzdem die Messwerte ermittelt, um die Effizienz des Algorithmus im Fehlerfall zu zeigen.

Beider Planung mit Wünschen und möglichen (präferierten) Ressourcenausprägungen wurden die Veranstaltungen entsprechend den Angaben in den beiden Kategorien eingeplant, sofern die Angaben vorhanden, korrekt und erfüllbar waren. Wenn eine der Angaben für die Ressourcen nicht erfüllbar oder nicht vorhanden war, wurde die Veranstaltung nicht eingeplant.

Veranstaltungen bleiben ungeplant. In diesen Fällen könnten entweder weniger Sperrzeiten definiert werden, oder der Planungszeitraum wird erweitert, um alle Veranstaltungen einzuplanen. Für die einzelnen Testbeispiele benötigt das Planungssystem **Kursplan 1.0** weniger als 2 Minuten, um eine Lösung zu erzeugen (siehe Abbildung 8.5).

![Zeitaufwand der Planung von Kursplan 1.0 (Pentium-4M, 1,4 G Hz, 256 MB RAM)](image)

Abbildung 8.5: Darstellung der Messergebnisse von Kursplan 1.0 im Säulendiagramm

Das Säulendiagramm in Abbildungen 8.5 gibt die vorherigen Messergebnisse graphisch wieder. Wegen des ungleichmäßigen Abstandes der Messpunkte ist die funktionelle Abhängigkeit nicht erkennbar. Zur Verdeutlichung dienen die Liniendiagramme in Abbildung 8.6. Die X-Achse wird proportional zur Anzahl der Veranstaltungen dargestellt. Zur Bestätigung, dass der Aufwand annähernd linear zur Zeit...
und Anzahl der Ressourcen steigt, sind ergänzende Trendlinien der Form \( f(x) = ax^2 \) und \( f(x) = \gamma x^2 + \delta x \) erfasst worden.

Abbildung 8.6: Darstellung der Messergebnisse von Kursplan 1.0 im Liniendiagramm

Auf Grund des nicht mehr so starken Anstieges der Kurven für die Bearbeitungszeiten, wie es beim Planungssystem ConTime 1.0 mit Backtracking der Fall gewesen war, wurden jetzt zur Approximation der Kurven zusätzlich zu den potentiellen Trendlinien \( f(x) = ax^2 \) auch polynome Trendlinien \( f(x) = \gamma x^2 + \delta x \) benutzt, um eine bessere Annäherung an den tatsächlichen Kurvenverlauf zu erreichen. Die Testreihen wurden wieder, für den Athlon- und für den Pentium-Prozessor durchgeführt, die ermittelten Trendlinien sind im Folgenden angegeben. Betrachtet wird hier nur das Liniendiagramm, auf dessen Basis der nachfolgende Vergleich beider Planungssysteme erfolgt. Die Diagramme des Athlon-Prozessors und die detaillierten Diagramme beider Prozessoren sind im Anhang enthalten.

Die Trendlinien für den Athlon-Prozessor haben bei der Planung mit Ressourcenwünschen die Funktion (potentiell) \( y = 0.03849891x^{0.89205175} \) (polynom) \( y = -0.000002x^2 + 0.021074x \), für die Planung mit Ressourcenwünsche/-präferenzen die Funktion (potentiell) \( y = 0.0442211x^{1.8013974} \) (polynom) \( y = 0.000002x^2 + 0.017077x \) und für die Planung mit all Ressourcenwünsche und -präferenzen zusammen mit den zusätzlichen Ressourcen die Funktion (potentiell) \( y = 0.05350417x^{0.8013974} \) (polynom) \( y = 0.000002x^2 + 0.016203x \).

Die Trendlinien für den Pentium-Prozessor haben bei der Planung mit Ressourcenwünschen die Funktion (potentiell) \( y = 0.04367365x^{0.64988127} \) (polynom) \( y = -0.000001x^2 + 0.005435x \), für die Planung mit Ressourcenwünsche/-präferenzen die Funktion (potentiell) \( y = 0.02624859x^{0.8082558} \) (polynom) \( y = 0x^2 + 0.007100x \) und für die Planung mit all Ressourcenwünsche und -präferenzen zusammen mit den zusätzlichen Ressourcen die Funktion (potentiell) \( y = 0.012406x^{0.078814} \) (polynom) \( y = 0.000003x^2 + 0.007145x \).

Bei den potentiellen Funktionen der Trendlinien sind der \( \alpha \) und \( \beta \)-Wert bestimmd. Der Funktionsverlauf. Hingegen ist bei den polynomialen Funktionen der Trendlinien der \( \gamma \)-Wert sehr klein, kann vernachlässigt werden und somit hat der quadratische Anteil fast keinen Einfluss auf die Funktion.

Der \( \delta \)-Wert des linearen Anteils der Funktion hat einen wesentlichen Einfluss auf den Funktionsverlauf und gibt entscheidend den Anstieg der Funktion vor. Wenn man die potentielle Funktion der Planung mit Ressourcenwünschen mit der Funktion des Planungssystems ConTime 1.0 mit Domänenerzerlegung (zerlegter Domänen) vergleicht, hat man für die Planung mit Wünschen die Funktionen \( y = 0.1778x^{0.0889} \) und \( y = 0.04367365x^{0.64988127} \).

Beim Vergleich der Ergebnisse der beiden Planungssysteme Kursplanung 1.0 und ConTime 1.0, der im folgenden Kapitel vertieft wird, ergibt sich ein Unterschied im \( \alpha \)-Wert der potentiellen Funk-
tionen, der beim Planungssystem Kursplanung 1.0 ungefähr um den Faktor 4 geringer ist als beim Planungssystem ConTime 1.0. Des Weiteren fällt auch der $\beta$-Wert beim Planungssystem Kursplanung 1.0 ungefähr um den Faktor 1,24 geringer aus als beim Planungssystem ConTime 1.0. Somit ist das Planungssystem Kursplan 1.0 bei vergleichbaren Bedingungen schneller als das Planungssystem ConTime 1.0.

Bei den polynomen Funktionen gibt es den gleichen Zusammenhang, wenn man die polynome Funktion der Planung mit Ressourcenwünschen mit der Funktion des Planungssystems ConTime 1.0 mit Domänenzerlegung (zerlegter Domänen) vergleicht, hat man für die Planung mit Wünschen die Funktionen $y = 0,00021x^2 + 0,034103x$ und $y = -0,000001x^2 + 0,005435x$. Hier zeigt sich derselbe Trend: beide Planungssysteme haben einen geringen $\gamma$-Wert, der sich ungefähr um den Faktor 240 voneinander unterscheidet und beim $\beta$-Wert hat das Planungssystem Kursplan 1.0 einen ungefähr um den Faktor 6,274 geringeren Anstieg des linearen Anteils als das Planungssystem ConTime 1.0.

Der neue Constraint solver $CS_{\text{d-MR}}$ ist in dieser Problemklasse effizienter in der Problemklasse als der Constraint löser in der kommerziellen constraintlogischen Programmiersprache CHIP.

### 8.3 Vergleich von ConTime und Kursplan

In diesem Abschnitt sollen die ermittelten Werte des Planungssystems Kursplan 1.0 mit den Ergebnissen des Planungssystems ConTime 1.0 zusammenfassend verglichen werden. Das Planungssystem ConTime 1.0 ist in der constraintlogischen Programmiersprache CHIP implementiert. Das Planungssystem Kursplan 1.0 ist in der Programmiersprache .Net und mit dem neuen Constraintlöser $CS_{\text{d-MR}}$ in C++ implementiert. Interessant ist hierbei das Zeitverhalten beider Planungssysteme (siehe Abbildung 8.7).

![Vergleich des Zeitaufwandes (Pentium-4M, 1,4 Ghz, 256MB RAM)](image)

**Abbildung 8.7: Vergleich ConTime 1.0 und Kursplan 1.0 mit uneingeschränkter Y-Achse**

In der Abbildung 8.7 sind die Messergebnisse des Planungssystems ConTime 1.0 mit den oberen Linien für die Suche mit Backtracking mit Wünschen und unzerlegten Domänen und für die Suche mit Look ahead-Check und Domänenzerlegung mit Wünschen und zerlegten Domänen mit den mittleren Linien dargestellt. Für das Planungssystem Kursplan 1.0 sind die Messergebnisse für die Planung für
alle Ressourcenwünsche und -präferenzen zusammen mit den zusätzlichen Ressourcen mit den unteren Linien angegeben.

Die Linien des Planungssystems ConTime 1.0 steigen bei der Suche über unzerlegten Domänen mit der Anzahl der zu planenden Veranstaltungen stark an und lassen einen exponentiellen Verlauf der Trendlinien vermuten. Die Trendlinien vom Planungssystem ConTime 1.0 liegen bei der Suche mit zerlegten Domänen nahe den Trendlinien des Planungssystems Kursplan 1.0.

Da es nicht möglich war, mehr als 1,6 GB Arbeitsspeicher von kommerziellen Programmiereinrichtungen zu nutzen, konnte der Zeitaufwand bei 2359 Veranstaltungen und unzerlegten Domänen nicht gemessen werden. Aus diesem Grund fehlen in der Linie der Messergebnisse die letzten Werte für mehr als 1297 Veranstaltungen. Bei der Planung mit unzerlegten Domänen stellt der enorme Speicherverbrauch und der große Zeitaufwand der Planungssystemen ConTime 1.0 ein Problem dar.

Für den direkten Vergleich der Planungssysteme ConTime 1.0 mit zerlegten Domänen und Kursplan 1.0 eignet sich die Planungsvarianten “alles” bei Kursplan 1.0 und “mit Wünschen” bei ConTime 1.0. Der gemessene Zeitaufwand wurde hier ausschließlich auf dem Laptop mit Pentium 4M, 1,4 GHz, 256 RAM ermittelt. In Abbildung 8.8 sind diese zusammenfassend in Liniendiagrammen dargestellt.

Die Messergebnisse von ConTime 1.0 mit den oberen und mittleren Linien liegen deutlich über den Messergebnissen von Kursplan 1.0, dargestellt mit den unteren Linien. Deutlich wird dabei, dass der Zeitaufwand bei unzerlegten Domänen von ConTime 1.0 mit zunehmender Anzahl der Veranstaltungen stark ansteigt. Während die Suche mit zerlegten Domänen (mittlere Linien) noch relativ schnell Lösungen liefert, entwickelt sich der Zeitaufwand bei der Suche auf unzerlegten Domänen (obere Linien) exponentiell. Das Liniendiagramm zeigt weiterhin, dass der Zeitaufwand in Kursplan 1.0 geringer und in Bezug auf die Anzahl der Veranstaltungen nahezu linear steigt, während es bei ConTime 1.0 einen exponentiellen steigenden Zeitaufwand bei der Suche mit unzerlegten Domänen und einen stellaren Anstieg bei der Suche mit zerlegten Domänen gibt.

Die Trendlinien können für das System ConTime 1.0 mit Wünschen und unzerlegten Domänen mit der Funktion $y = 0,000033x^{2,885826}$ beschrieben werden, für das System ConTime 1.0 mit Wünschen und zerlegten Domänen mit der Funktion $y = 0,147656x^{5,881577}$ und für das System Kursplan 1.0
mit allen Wünschen mit der Funktion $y = 0,0139x^{0.9023}$.
Beide Systeme unterscheiden sich nicht nur in Bezug auf den Zeitaufwand, sondern auch im benötigten Speicherplatz. Während **Kursplan 1.0** nicht mehr als 100MB beanspruchte, benötigte das System **ConTime 1.0** für seinen größtmöglichen lösbaren Plan mit 1927 Veranstaltungen das 16-fache an Speicherplatz zur Planung der Veranstaltungen. Der nächst größere Plan mit 2359 Veranstaltungen ließ sich auf Grund des ausgeschöpften Speicherplatzes nicht mehr planen.

Abschließend kann festgestellt werden, dass die Bearbeitung einer Planungsaufgabe mit **ConTime 1.0**, das in der kommerziellen constraintlogischen Programmiersprache CHIP programmiert wurde, vergleichsweise viel Zeit und einen verhältnismäßig großen Speicher bei der Planung mit unzerlegten und zerlegten Domänen benötigt. Das Planungssystem **ConTime 1.0** eignet sich aus diesen Gründen für Planungsprobleme mit einer geringen Anzahl von Veranstaltungen, bei denen innerhalb eines kurzen Horizonts geplant werden muss. Das Planungssystem **Kursplan 1.0** mit dem Constraintlöser $CS_{1d-MR}$ in der Programmiersprache $C^{++}$ benötigt einen geringen Zeitaufwand für die Planung und wenig Speicher. Dieses Planungssystem ist aufgrund der daraus resultierenden kurzen Antwortzeiten und des geringeren Speicherbedarfs für die interaktive Planung auch größerer Probleme geeignet.
Kapitel 9

Ausblick:


Abbildung 9.1: Erweiterte Komponenten des Constraint solvers $CS_{fd-MR}$

In Anwendungen, die mit großen Datenmengen unter Echtzeitbedingungen arbeiten, wie zum Beispiel bei der Simulation in Eisenbahnnetzten, könnte die Integration des Constraint solvers vorgenommen werden, um keine Effizienzverluste durch zusätzliche verallgemeinerte Datenformate für die Ein- und Ausgabe und die Aufrufformate zu erhalten. Die Realisierung direkter Aufrufe des in C++ geschriebenen Codes des Constraint solvers würde kurze Rechenzeiten und ein effizientes Management der Speichervernutzung ermöglichen.

Um den Constraint solvers $CS_{fd-MR}$ eigenständig einsetzen zu können, sind verallgemeinerte Datenformate für die Ein- und Ausgabe sowie für die Aufrufformen zu definieren.
9.1 Erweiterung des Constraintsolvers $CS_{fd-MR}$ um weitere Constraints und eine Optimierungs methode


Durch die Erweiterung um die Branch and bound-Methode wäre die Möglichkeit gegeben, nicht nur gute Lösungen, sondern die optimale Lösung für ein gegebenes Problem zu finden. Hierbei wird durch das mehrmalige Lösen des Problems bei gezielt veränderten Bedingungen das Optimum des Problems ermittelt.


9.2 Simulation in Eisenbahnnetzen

Der Constraintlöscher $CS_{fd-MR}$ könnte für die Simulation in Eisenbahnnetzen eingesetzt werden. Hierfür stünden die Vorschläge für die Modellierung aus den Veröffentlichungen [13], [37], [58] zum Förderprojekt¹ "Simulation von Trassen-Slots und Zuglagen in Eisenbahnnetzen" (SIMONE) zur Verfügung.

Die Grundlage für die Modellierung im Eisenbahnverkehr bildet die Infrastruktur und die Züge, die zu einer bestimmten Zeit an einem bestimmten Punkt der Infrastruktur fahren oder stehen. Die Infrastruktur, auch Topologie genannt, liegt zweidimensional vor. In der ersten Dimension werden parallele Gleise und in der zweiten Dimension die Länge der einzelnen Gleise abgetragen. Über dieser Infrastruktur befindet sich eine dritte Dimension, die Zeit, auf der sich der zeitliche Verlauf der Zugfahrten auf dieser vorhandenen Infrastruktur abbilden lässt. Die drei Dimensionen werden dann als dreidimensionales Koordinatensystem betrachtet. In der x-Achse wird die Länge der Gleise, in der y-Achse die parallel liegenden Gleise und in der z-Achse die Zeit abgetragen.

Die Infrastruktur muss zur Reduzierung der Datenmenge und zur Behandlung mit Constraint-Techniken auf endlichen Domänen und den entsprechenden Solvoren $CS_{fd}$ diskretisiert werden. Zu diesem Zweck wird über die Infrastruktur ein Netz gelegt und die entstehenden Rechtecke als kleinste Einheit betrachtet (siehe Abbildung 9.2).

¹Förderprojekt des BMBF Förderkennzeichen: 19 P 0021
Infrastruktur/Topologie


Züge


Kapitel 10

Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Constraintlöser für diskrete mehrdimensionale Multiressourcenprobleme beschrieben. Als exemplarisches mehrdimensionales Multiressourcenproblem wurde die Kursplanung für eine Weiterbildungseinrichtung ausgewählt. Für das Problem der mehrjährigen Kursplanung existierten reale Daten einer Weiterbildungseinrichtung für einen Kursplan mit einem Planungszeitraum von 75 Wochen, für zwei Standorte mit unterschiedlichen Räumen und spezialisierteren Dozenten. Die Daten umfassten nahezu alle bei einer Kursplanung vorkommenden Bedingungen.


In der vorliegenden Arbeit wurden die Eigenschaften des Constraintlösers für mehrdimensionale Multiressourcenprobleme am Problem der mehrjährigen Kursplanung für Weiterbildungseinrichtungen vorgestellt. Die Implementierung des Constraintlöser erfolgte in C++ und der Constraintlöser ist außer an der Kursplanung auch, in Zusammenarbeit mit der Deutschen Bahn AG, für das sehr komplexe Problem der Fahrplan- und Betriebssimulation in Eisenbahnen erfolgreich getestet worden.
<table>
<thead>
<tr>
<th>Abschnitt</th>
<th>Seitenzahl</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Lösungsaufwand in Abhängigkeit von der Problemdimension</td>
<td>17</td>
</tr>
<tr>
<td>3.1 Geometrische Interpretation von $y \leq 5x \wedge x \leq 5y$</td>
<td>24</td>
</tr>
<tr>
<td>3.2 Geometrische Interpretation von $5y &lt; x \wedge 5x &lt; y$</td>
<td>25</td>
</tr>
<tr>
<td>3.3 Geometrische Projektion</td>
<td>26</td>
</tr>
<tr>
<td>3.4 Übersichtsbild (Backtracking, Forward checking, Look ahead)</td>
<td>35</td>
</tr>
<tr>
<td>3.5 Beispiel 1 - GC1: Cumulative - Constraint</td>
<td>41</td>
</tr>
<tr>
<td>3.6 Beispiel 2 - GC1: Cumulative - Constraint</td>
<td>42</td>
</tr>
<tr>
<td>3.7 Beispiel 1 - GC2: Diffn - Constraint</td>
<td>44</td>
</tr>
<tr>
<td>4.1 Komponenten des Constraintsolvers CS/fd, MR</td>
<td>48</td>
</tr>
<tr>
<td>4.2 N - Ressourcendimension</td>
<td>49</td>
</tr>
<tr>
<td>4.3 Suchraum $S$, Teilsuchraum $ST$ und zerlegter Teilsuchraum $zST$</td>
<td>50</td>
</tr>
<tr>
<td>4.4 Beispiel GCR1: diffn2D - Constraint</td>
<td>59</td>
</tr>
<tr>
<td>4.5 Reihenfolgebeziehung von drei Ereignissen mit Domänen</td>
<td>60</td>
</tr>
<tr>
<td>5.1 Sortierung der Domänen und Bildung einer Treppenstruktur</td>
<td>66</td>
</tr>
<tr>
<td>5.2 Abschätzung der oberen Grenze des Suchaufwandes</td>
<td>66</td>
</tr>
<tr>
<td>5.3 Unterteilung der Suchräume</td>
<td>67</td>
</tr>
<tr>
<td>5.4 Bestimmung der Zerlegungpositionen</td>
<td>68</td>
</tr>
<tr>
<td>5.5 Zerlegung des Teilsuchraums an den Zerlegungspositionen</td>
<td>69</td>
</tr>
<tr>
<td>5.6 Bestimmung der Zerlegungpositionen</td>
<td>69</td>
</tr>
<tr>
<td>5.7 Zerlegung des Teilsuchraums an den Zerlegungspositionen</td>
<td>70</td>
</tr>
<tr>
<td>5.8 Kapazitäten der Ressourcendimension $RD_2$ zur Ressourcendimension $RD_1$</td>
<td>71</td>
</tr>
<tr>
<td>5.9 Aufbau der Planungskomponente für das Beispiel der Kursplanung an Weiterbildungseinrichtungen</td>
<td>72</td>
</tr>
<tr>
<td>5.10 Ablauf des Einplanens einer Veranstaltung mit Zeit, Dozent und Raum</td>
<td>78</td>
</tr>
<tr>
<td>6.1 Eindimensionale und dreidimensionale Verschiedenheit beim Diffn - Constraint</td>
<td>91</td>
</tr>
<tr>
<td>6.2 Ressourcendimensionen $X$, $Y$ und $Z$</td>
<td>92</td>
</tr>
<tr>
<td>6.3 Zweitafelprojektion eines räumlichen Objekts in den zugeordneten Normalrissen</td>
<td>94</td>
</tr>
<tr>
<td>6.4 Visualisierung der zweidimensionalen Flächen für den vierdimensionalen Raum</td>
<td>96</td>
</tr>
<tr>
<td>6.5 Ressourcendimensionen $RD_1$, $RD_2$, $RD_3$, $RD_4$ und $RD_4+n$</td>
<td>97</td>
</tr>
<tr>
<td>7.1 Graphische Interpretation der Abhängigkeiten</td>
<td>102</td>
</tr>
<tr>
<td>8.1 Darstellung der Messergebnisse von ConTime 1.0 im Säulendiagramm des Athlon - Prozessors</td>
<td>108</td>
</tr>
<tr>
<td>8.2 Darstellung der Messergebnisse von ConTime 1.0 Liniendiagramm</td>
<td>109</td>
</tr>
<tr>
<td>8.3 Darstellung der Messergebnisse von ConTime 1.0 im Säulendiagramm des Pentium - Prozessors</td>
<td>111</td>
</tr>
<tr>
<td>8.4 Darstellung der Messergebnisse von ConTime 1.0 im Liniendiagramm</td>
<td>112</td>
</tr>
<tr>
<td>8.5 Darstellung der Messergebnisse von Kursplan 1.0 im Säulendiagramm</td>
<td>114</td>
</tr>
<tr>
<td>8.6 Darstellung der Messergebnisse von Kursplan 1.0 im Liniendiagramm</td>
<td>115</td>
</tr>
</tbody>
</table>
ABBLDUNGSVERZEICHNIS

8.7 Vergleich ConTime 1.0 und Kursplan 1.0 mit uneingeschränkter Y-Achse .......... 116
8.8 Vergleich ConTime 1.0 und Kursplan 1.0 mit eingeschränkter Y-Achse ............... 117

9.1 Erweiterte Komponenten des Constraintolvers $CS_{fd-MR}$ .................................. 119
9.2 Diskretisierung der Infrastruktur in Spurplangenaigkeit .................................. 121

D.1 Komponenten der beiden Stundenplanungssysteme mit CHIP und dem Constraintlös er
$CS_{fd-MR}$ .................................................................................................................. 152

E.1 Darstellung der Messergebnisse von ConTime 1.0 im Säulendiagramm mit eingeschränkter (oben) und maximaler (unten) Y-Achse .......................................................... 155
E.2 Darstellung der Messergebnisse von ConTime 1.0 im Liniendiagramm mit eingeschränkter (oben) und maximaler (unten) Y-Achse .......................................................... 156
E.3 Darstellung der Messergebnisse von ConTime 1.0 im Säulendiagramm mit eingeschränkter (oben) und maximaler (unten) Y-Achse .......................................................... 157
E.4 Darstellung der Messergebnisse von ConTime 1.0 im Liniendiagramm mit eingeschränkter (oben) und maximaler (unten) Y-Achse .......................................................... 158
E.5 Darstellung der Messergebnisse von Kursplan 1.0 (Athlon - Prozessor 700 Mhz, 256 MB RAM) ................................................................................................. 159
E.6 Darstellung der Messergebnisse von Kursplan 1.0 (Pentium-4M - Prozessor, 1,4 Ghz, 256 MB RAM) ................................................................. 160
E.7 Vergleich ConTime 1.0 und Kursplan 1.0 mit eingeschränkter (oben) und maximaler (unten) Y-Achse ................................................................. 161
Tabellenverzeichnis

<table>
<thead>
<tr>
<th>Nummer</th>
<th>Titel</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Rechenzeit zur Ermittlung der Lösungen mit den zwei Methoden &quot;Backtracking&quot; und &quot;Domänenzerlegung mit Look ahead und dem Reihenfolgeconstraint RC1/RC2&quot; (1)</td>
<td>107</td>
</tr>
<tr>
<td>8.2</td>
<td>Rechenzeit zur Ermittlung der Lösungen mit den zwei Methoden &quot;Backtracking&quot; und &quot;Domänenzerlegung mit Look ahead und dem Reihenfolgeconstraint RC1/RC2&quot; (2)</td>
<td>108</td>
</tr>
<tr>
<td>8.3</td>
<td>Messergebnisse von ConTime 1.0 und vom System Pentium 4M, 1.4 Ghz, 256 MB RAM mit und ohne Ressourcenwünsche</td>
<td>110</td>
</tr>
<tr>
<td>8.4</td>
<td>Messergebnisse von Kursplan 1.0 und vom System Athlon, 700 Mhz, 256 MB RAM</td>
<td>113</td>
</tr>
<tr>
<td>8.5</td>
<td>Messergebnisse von Kursplan 1.0 und vom System Pentium 4M, 1.4 Ghz, 256 MB RAM</td>
<td>114</td>
</tr>
</tbody>
</table>
Index

backtracking, 32
build-in-Prädikate, 19

Constraint, 21
Constraint satisfaction Problem, 16
globale, 38
diffl, 39
among, 120
cumulative, 40
cycle, 120
diffn, 44
diffn2D, 105
diskjunktn Serialisierung, 39
RC, 105
harte Constraints, 12, 105
Programmierung, 19
weiche Constraints, 12, 105
Constraint-Satisfaction-Problem, 27
Constraintlöser, 16, 25
Determinationsdetektion, 26
Elimination, 26
endliche Domänen/ld, 27
Erfüllbarkeit, 23
Erfüllbarkeit, 25
Folgerbarkeit, 25
multiressourcen/ld-MR, 47
Domäne, 51
Domänenzerlegung, 53
globales Reihenfolgeconstraint RC, 60
globales Ressourcenconstraint ddiffn2D, 56
Nichtzeitenhierarchie, 53
notin, 56
Problemerzeugung, 65
Projektion, 26
Simplifikation, 26
Wohlderhalten, 27

Constraintsysteme, 16, 22
bool, 22
domäne/ld, 23

Domäne, 22

Ereignis, 49
typ, 61
typenliste, 62

Hoch- und Fachschulen, 14

Konsistenz
Grenzen-Konsistenz, 30
Kanten-Konsistenz, 28
Knoten-Konsistenz, 28
locale, 29
Krankenpflegeschulen, 14

Löschungsverfahren
Divide-and-Conquer, 65

Modellierung
Aufbrechen von Symmetrien, 37
Constraints, 97
Redundante Constraints, 37
Redundante Constraints und Aufbrechen von Symmetrien, 38
Stundenplanprobleme, 91
Multiressourcenproblem, 13

Planung, 72
Ausplanen, 74
Einplanen, 73
er aller Veranstaltungen, 81
er einer Veranstaltung, 82
fester Zeitpunkt, 86
Heuristiken, 87
Initialisierung, 72
private Weiterbildungsträger, 14
Prolog, 19
Horn-Klausel, 20
Klausel, 20
Propagation, 16, 33
Forward checking, 34
Look ahead, 34

Ressource
Dimension, 49
Kapazität, 50

Schulen, 14
SLD-Resolution, 20
Stundenplan, 11
Software, 15
Stundenplanungsproblem, 11

Suche
Backtracking, 21
backtracking, 32
Breitensuche, 21
Domänenreduzierung, 32
Heuristische Suche, 35
  Variablenreihenfolge, 35
  Wertereihenfolge, 36
Labeling, 32
Suchbaum, 21
Suchraum, 50
Teilsuchraum, 50
Tiefensuche, 21
  zerlegter Teilsuchraum, 50
Systeme
  Stundenplanung, 151

Unifikator, 20
Universitäten, 14

Variable, 22

Zuordnungsproblem, 11
Literaturverzeichnis


LITERATURVERZEICHNIS


[80] Rainer Wüst. Mathematik für Physiker und Mathematiker Band 1. WILEY VCH.
Anhang A

Funktionen des globalen Ressourcen-Constraints diffn2D

Konstanten

- \textit{CS – RESOURSE – STATE – FREE} entspricht der -1, kennzeichnet einen Wert als unbenutzt
- \textit{CS – RESOURSE – STATE – NOT – AVAILABLE} entspricht der -2, kennzeichnet einen Wert als nicht verfügbar

Konstanten zum Laufzeit-Status

- \textit{CS – STATUS – IS – LOCKING} gibt an, dass das Objekt gerade die Lock-Funktion ausführt
- \textit{CS – STATUS – IS – CHANGING} gibt an, dass das Objekt gerade die ChangeX- oder ChangeY-Funktion ausführt
- \textit{CS – STATUS – IS – CROSSING} gibt an, dass das Objekt gerade die Cross-Funktion ausführt
- \textit{CS – STATUS – IS – UNLOCKING} gibt an, dass das Objekt gerade die Unlock-Funktion ausführt

Definition und Initialisierung

- \texttt{diffn2D-K(void)}:
  Der Defaultkonstruktor setzt die interne Datenrepräsentation auf die Ausdehnung \(x=0, y=0\).

- \texttt{diffn2D(DWORD x, DWORD y)}:
  Der Konstruktor initialisiert die interne Datenrepräsentation mit den übergebenen Ausdehnungen.

- \texttt{diffn2D(list< list<DWORD> >* LRe, DWORD x, DWORD y, list< list<DWORD> >* LSp)}:
  Der Hauptkonstruktor initialisiert die interne Datenrepräsentation mit den übergebenen Ausdehnungen, setzt die Liste der Sperrflächen in die interne Datenrepräsentation ein und platziert die Liste der übergebenen Rechtecke in der internen Datenrepräsentation.

- \texttt{diffn2D-D(void)}:
  Der Destruktor gibt benötigten Speicher wieder frei.
• WORD getCurrentStatus(void) const:
  Gibt den aktuellen Status des Objektes mit folgenden möglichen Rückgabewerten an:
  - CS = STATUS = IS = LOCKING
  - CS = STATUS = IS = CHANGING
  - CS = STATUS = IS = CROSSING
  - CS = STATUS = IS = UNLOCKING

• void setDimensions(DWORD x, DWORD y):
  Setzt die Ausdehnungen des Feldes neu und löscht damit die alten Daten.

• void setXDimension(DWORD x):
  Setzt die x-Ausdehnung des Feldes neu und löscht damit die alten Daten.

• DWORD getXDimension(void) const:
  Gibt die x-Ausdehnung des Feldes zurück.

• void setYDimension(DWORD y):
  Setzt die y-Ausdehnung des Feldes neu und löscht damit die alten Daten.

• DWORD getYDimension(void) const:
  Gibt die y-Ausdehnung des Feldes zurück.

**Distanz**

• void setXDistance(DWORD x):
  Setzt den Wert für den minimalen Abstand zweier Rechtecke in der X-Dimension zueinander. Dabei muss sich ein als frei gekennzeichneter Bereich zwischen den Rechtecken auf der gleichen Spalte/Zeile befinden. Es können aber auch andere Statuswerte definiert werden, die als frei in Bezug auf die Distanz interpretiert werden. Dazu dient die Funktion AddDistanceConstant.

• WORD getXDistance(void):
  Gibt die Länge der einzuhaltenenden Distanz in der X-Dimension zurück.

• void setYDistance(DWORD y):
  Setzt den Wert für den minimalen Abstand zweier Rechtecke in der Y-Dimension zueinander. Dabei muss sich ein als frei gekennzeichneter Bereich zwischen den Rechtecken auf der gleichen Spalte/Zeile befinden. Es können aber auch andere Statuswerte definiert werden, die als frei in Bezug auf die Distanz interpretiert werden. Dazu dient die Funktion AddDistanceConstant.

• DWORD getYDistance(void):
  Gibt die Länge der einzuhaltenenden Distanz in der Y-Dimension zurück.

• Bool AddDistanceConstant(short neutral):
  Mit Hilfe der Funktion können zusätzliche Statuswerte festgelegt werden, die bei der Berücksichtigung von Distanzen als frei interpretiert werden. Zu beachten ist, dass diesbezüglich nur negative Werte kleiner -1 verwendet werden dürfen.
• void DeleteDistanceConstant(short neutral):

Passage
• bool MakePassageList(const ValueList* values, PassageList* passages):

Nutzung bekannt geben(Lock)
• bool Lock(bool canOverwrite=false):

• bool Lock(short state, bool canOverwrite=false):
  Sie kennzeichnet alle Werte des Feldes mit dem übergebenen Wert.

• bool Lock(DWORD firstX, DWORD firstY, DWORD width, DWORD height, bool canOverwrite=false):

• bool Lock(DWORD firstX, DWORD firstY, DWORD width, DWORD height, short state, bool canOverwrite=false):

Verschieben
In X-Richtung.
• bool ChangeX(DWORD rowXOld, DWORD rowXNew, short state):

• bool ChangeX(DWORD rowXOld, DWORD rowXNew, DWORD firstY, short state):
  Verschiebt die mit dem Wert gekennzeichneten Werte aus der Reihe rowXOld in die Reihe rowXNew ab der y-Koordinaten firstY, falls möglich. In der Reihe rowXOld wird für jeden Wert der
Wert \( CS = \texttt{RESSOURCE} - \texttt{STATE} - \texttt{FREE} \) gesetzt.

- \texttt{bool \: ChangeX(DWORD \: \texttt{rowXOld}, DWORD \: \texttt{rowXNew}, DWORD \: \texttt{firstYOld}, DWORD \: \texttt{firstYNew}, short \: \texttt{state})}: Verschiebt die mit dem Wert gekennzeichneten Werte aus der Reihe \texttt{rowXOld} in die Reihe \texttt{rowXNew} ab der \( y \)-Koordinate \texttt{firstYOld} nach \texttt{firstYNew}, falls möglich. In der Reihe \texttt{rowXOld} wird für jeden Wert der Wert \( CS = \texttt{RESSOURCE} - \texttt{STATE} - \texttt{FREE} \) gesetzt.

In \( Y \)-Richtung.

- \texttt{bool \: ChangeY(DWORD \: \texttt{rowYOld}, DWORD \: \texttt{rowYNew}, short \: \texttt{state})}: Verschiebt die mit dem Wert gekennzeichneten Werte aus der Reihe \texttt{rowYOld} in die Reihe \texttt{rowYNew}, falls möglich. In der Reihe \texttt{rowYOld} wird für jeden Wert der Wert \( CS = \texttt{RESSOURCE} - \texttt{STATE} - \texttt{FREE} \) gesetzt.

- \texttt{bool \: ChangeY(DWORD \: \texttt{rowYOld}, DWORD \: \texttt{rowYNew}, DWORD \: \texttt{firstX}, short \: \texttt{state})}: Verschiebt die mit dem Wert gekennzeichneten Werte aus der Reihe \texttt{rowYOld} in die Reihe \texttt{rowYNew} ab der \( x \)-Koordinate \texttt{firstX}, falls möglich. In der Reihe \texttt{rowYOld} wird für jeden Wert der Wert \( CS = \texttt{RESSOURCE} - \texttt{STATE} - \texttt{FREE} \) gesetzt.

- \texttt{bool \: ChangeY(DWORD \: \texttt{rowYOld}, DWORD \: \texttt{rowYNew}, DWORD \: \texttt{firstXOld}, DWORD \: \texttt{firstYNew}, short \: \texttt{state})}: Verschiebt die mit dem Wert gekennzeichneten Werte aus der Reihe \texttt{rowYOld} in die Reihe \texttt{rowYNew} ab der \( x \)-Koordinate \texttt{firstX} nach der \( x \)-Koordinate \texttt{firstXNew}, falls möglich. In der Reihe \texttt{rowYOld} wird für jeden Wert der Wert \( CS = \texttt{RESSOURCE} - \texttt{STATE} - \texttt{FREE} \) gesetzt.

**Vertauschen**

- \texttt{bool \: Cross(short \: \texttt{state1}, short \: \texttt{state2})}: Vertauscht zwei Werte miteinander.

**Nutzung aufheben (UnLock)**

- \texttt{bool \: UnLock()}: Gibt alle Werte mit \( f(x, y) \geq 0 \) des gesamten Lösungsbereichs frei und kennzeichnet sie mit \( CS = \texttt{RESSOURCE} - \texttt{STATE} - \texttt{FREE} \).

- \texttt{bool \: UnLock(short \: \texttt{state})}: Gibt alle Werte mit \( f(x, y) = \text{state} \) des gesamten Lösungsbereichs frei und kennzeichnet sie mit \( CS = \texttt{RESSOURCE} - \texttt{STATE} - \texttt{FREE} \).

- \texttt{bool \: UnLock(DWORD \: \texttt{firstX}, DWORD \: \texttt{firstY}, DWORD \: \texttt{width}, DWORD \: \texttt{height})}: Gibt alle Werte mit \( f(x, Y) \geq 0 \) innerhalb des übergebenen Rechteckes frei und kennzeichnet sie mit \( CS = \texttt{RESSOURCE} - \texttt{STATE} - \texttt{FREE} \).
Freiräume ermitteln

- void getFreeY(DWORD x, list<DWORD>* value):
  Gibt alle y-Argumente, die mit CS – RESsource – State – FREE gekennzeichnet sind, an der x-Koordinate zurück.

- void getFreeY(DWORD x, list<DWORD>* value, const list<short>* neutrals):
  Gibt alle y-Argumente zurück, die mit einem der Werte aus der Liste neutrals gekennzeichnet sind.

- void getFreeY(DWORD x, DWORD length, list<DWORD>* value):
  Gibt alle y-Argumente zurück, die mit einem der Werte aus der Liste neutrals gekennzeichnet sind, im Bereich von x bis x+length zurück.

- void getFreeX(DWORD y, list<DWORD>* value):
  Gibt alle x-Argumente, die mit CS – RESsource – State – FREE gekennzeichnet sind, an der y-Koordinate zurück.

- void getFreeX(DWORD y, list<DWORD>* value, const list<short>* neutrals):
  Gibt alle x-Argumente, die mit einem Wert aus der Liste neutrals gekennzeichnet sind, an der y-Koordinate zurück.

- void getFreeX(DWORD y, DWORD length, list<DWORD>* value):
  Gibt alle x-Argumente im Bereich von x bis x+length zurück, die mit CS – RESsource – State – FREE gekennzeichnet sind.

- void getFreeX(DWORD y, DWORD length, list<DWORD>* value, list<short>* neutrals):
  Gibt alle x-Argumente im Bereich von x bis x+length zurück, die mit einem Wert aus der Liste neutrals gekennzeichnet sind.

- bool isFree(DWORD firstX, DWORD firstY, DWORD width, DWORD height):

- bool isFree(DWORD firstX, DWORD firstY, DWORD width, DWORD height, const list<short>* neutrals):
  Prüft, ob das angegebene Rechteck mit einem der Werte aus der Liste neutrals gekennzeichnet ist.

Übergreifende Freiräume (Intersection)

- void IntersectX(DWORD firstX, DWORD firstY, DWORD width, DWORD height, list<CS – Rectangle>* value):
1 und width entspricht der Länge des freien Bereiches.

- void IntersectX(DWORD firstX, DWORD firstY, DWORD width, DWORD height, list<CS - Rectangle>* value, const list<short>* neutrals):

- void IntersectX(DWORD firstX, DWORD firstY, DWORD width, DWORD height, list<CS - Rectangle>* value, DWORD minLength):

- void IntersectY(DWORD firstX, DWORD firstY, DWORD width, DWORD height, list<CS - Rectangle>* value, const list<short>* neutrals): 

- void IntersectY(DWORD firstX, DWORD firstY, DWORD width, DWORD height, list<CS - Rectangle>* value, DWORD minLength):

- void IntersectY(DWORD firstX, DWORD firstY, DWORD width, DWORD height, list<CS - Rectangle>* value, const list<short>* neutrals, DWORD minLength):
  Sie teilt den übergebenen Bereich in als mit Werten aus der Liste neutrals gekennzeichnete Rechtecke ein und gibt diese zurück. Dabei wird jeweils jede Spalte für sich betrachtet und pro Spalte Rechtecke generiert, wenn freie Werte vorhanden sind. Die freien Bereiche müssen dabei wenigstens die Länge minLength aufweisen.
Rechtecke generiert, wenn freie Werte vorhanden sind. Die freien Bereiche müssen dabei mindestens die Länge minLength aufweisen. Dadurch ist width immer 1 und height entspricht der Länge des freien Bereiches.
Anhang B

Parameter bei der Planung

USELASTINSERTION

Ist bereits eine Veranstaltung eingeplant worden, die mit der zu planenden Veranstaltung eine identische Veranstaltungsdefinition hat, so kann man davon ausgehen, dass der Planungsalgorithmus bereits alle Tage vor dem Tag, an dem die Veranstaltung eingeplant wurde, auf freie Zeiträume mit allen zur Planung vorgesehenen Ressourcen getestet hat. Wird dieses Flag mit angegeben, so wird derjenige Tag als erster nach Freiräumen zu durchsuchender Tag genommen, an dem zuletzt eine Veranstaltung der gleichen Veranstaltungsdefinition eingeplant wurde. Wird die Anzahl der möglichen Ressourcen vergrößert, so macht das Setzen dieses Flags keinen Sinn.

SCHEDULETIME

Neben Wunschzeiten können bei der Planung auch alle anderen möglichen Zeitpunkte, nach dem erfolgreichen Test der Wunschzeiten, berücksichtigt werden. Ist dieses gewünscht, so ist dieser Parameter zu setzen. Ist er nicht gesetzt, so werden nur die Wunschzeiten genommen, unabhängig davon, ob PRIORITY.TIME gesetzt ist.

USEWISHES

Ist dieser Parameter gesetzt, so werden für die Planung nur die Ressourcen gewählt, die als Wunsch angegeben wurden.

USEPOSSIBILITIES

Ist dieser Parameter gesetzt, so werden für die Planung nur die Ressourcen gewählt, die als Wünsche und bevorzugte Ressourcen angegeben wurden. Es werden zuerst die Wünsche und dann die bevorzugten Ressourcen gewählt.

USEALL

Ist dieser Parameter gesetzt, so werden für die Planung alle Ressourcen gewählt, die alle Restriktionen der Veranstaltungen erfüllen. Es werden zuerst die Wünsche, dann die bevorzugten und zuletzt die möglichen Ressourcen gewählt.

PRIORITY.TIME

Sollen die Zeitwünsche bei der Planung berücksichtigt werden, so ist dieser Parameter zu setzen. Dabei können auch Zeitwünsche definiert worden sein, die außerhalb der vorher festgelegten, bevorzugten Woche liegen.

PRIORITY.ROOM

Ist dieser Parameter gesetzt, so bekommen die Räume eine höhere Priorität als die Dozenten. Er sollte gesetzt werden, wenn die Räume die knappe Ressource sind.
**Priority Teacher**

Ist dieser Parameter gesetzt, so bekommen die Dozenten eine höhere Priorität als die Räume. Er sollte gesetzt werden, wenn die Dozenten die knappe Ressource sind. Ist nicht der Parameter **Priority Room** angegeben, so werden standardmäßig die Dozenten als bevorzugte Ressource gewählt.

**Log Standard**

Ist dieser Parameter gesetzt, so werden Nachrichten über den aktuellen Status der Planung generiert. Allerdings muss dazu noch eine Funktion angegeben werden, über die die Anwendung über neue Nachrichten informiert wird.

**Log Detailed**

Für diesen Parameter gilt das gleiche wie für **Log Standard**. Die generierten Nachrichten geben detailliertere Auskünfte über den aktuellen Planungsprozess. Wenn der Parameter gesetzt ist, muss nicht mehr **Log Standard** gesetzt werden.

**Default**

Alle vorangegangenen Parameter werden gesetzt, wenn dieser Parameter gesetzt wird. Nähere Informationen zu den einzelnen Parametern stehen bei den entsprechenden Parametern.
Anhang C

Schnittstelle

Folgende Funktionen sind bei der Planung von Bedeutung:

C.1 Funktionen

```cpp
bool Schedule(void)
```

Plant alle Veranstaltungen mit den Default Flags ein. Wurden alle Veranstaltungen erfolgreich verplant, wird true zurückgegeben, ansonsten false.

```cpp
bool Schedule(WORD flags)
```

Plant alle Veranstaltungen unter der Berücksichtigung der übergebenen Flags ein. Wurden alle Veranstaltungen erfolgreich verplant, wird true zurückgegeben, ansonsten false.

```cpp
bool Schedule(const string& event)
```

Sucht die zur Id gehörende Veranstaltung und versucht, sie unter Berücksichtigung der Default Flags einzuplanen. Wurde die Veranstaltung erfolgreich eingeplant, dann gibt die Funktion true zurück, ansonsten false.

```cpp
bool Schedule(const string& event, WORD flags)
```

Sucht die zur Id gehörende Veranstaltung und versucht, sie unter Berücksichtigung der übergebenen Flags einzuplanen. Wurde die Veranstaltung erfolgreich eingeplant, dann gibt die Funktion true zurück, ansonsten false.

```cpp
bool Schedule(kpmValues* event)
```

Plant die übergebene Veranstaltung unter der Berücksichtigung der Default Flags ein. Wurde die Veranstaltung erfolgreich eingeplant, dann gibt die Funktion true zurück, ansonsten false.

```cpp
bool Schedule(kpmValues* event, WORD flags)
```

Plant die übergebene Veranstaltung unter der Berücksichtigung der übergebenen Flags ein. Wurde die Veranstaltung erfolgreich eingeplant, dann gibt die Funktion true zurück, ansonsten false.

```cpp
bool UnSchedule()
```

Plant alle eingeplanten Veranstaltungen aus. Gibt true zurück, falls alle ausgeplant werden konnten, ansonsten false.

```cpp
bool UnSchedule(const string& event)
```

Sucht die zur Id gehörende Veranstaltung und plant diese aus. Gibt true zurück, falls die Veranstaltung erfolgreich ausgeplant werden konnte, ansonsten false.
C.2 Eigenschaften und Konstanten

Damit die Planung viel Flexibilität bietet, lässt sich der Planungsprozess leicht mit Hilfe von Parametern und variablen Größen variieren. Folgende Parameter können der Funktion übergeben werden:

**SCHEDULER_FLAG_USELASTINSERTION (0×1)**

Ist bereits eine Veranstaltung eingeplant worden, die mit dieser identisch ist, so kann man davon ausgehen, dass der Planungsalgorithmus bereits alle Tage vor dem Tag, an dem die Veranstaltung eingeplant wurde, auf freie Zeiträume mit allen zur Planung vorgesehenen Ressourcen getestet hat. Wird dieses Flag mit angegeben, so wird der Tag als erster nach Freiräumen zu durchsuchender Tag genommen, an dem zuletzt eine Veranstaltung der gleichen Veranstaltungsdefinition eingeplant wurde. Wird die Anzahl der möglichen Ressourcen vergrößert, so macht das Setzen dieses Flags keinen Sinn.

**SCHEDULER_FLAG_SCHEDULEETIME (0×2)**

Neben Wunschzeiten können bei der Planung auch alle anderen möglichen Zeitpunkte berücksichtigt werden. Ist dieses gewünscht, so ist dieses Flag zu setzen. Ist es nicht gesetzt, so werden nur die Wunschzeiten genommen, unabhängig davon, ob SCHEDULER_FLAG_PRIORITY_TIME gesetzt ist.

**SCHEDULER_FLAG_USEWISHES (0×4)**

Ist dieses Flag gesetzt, so werden für die Planung nur die Ressourcen gewählt, die als Wunsch angegeben wurden.

**SCHEDULER_FLAG_USEPOSSIBILITIES (0×8)**

Ist dieses Flag gesetzt, so werden für die Planung nur die Ressourcen gewählt, die als Wünsche und bevorzugte Ressourcen angegeben wurden. Es werden zuerst die Wünsche und dann die bevorzugten Ressourcen gewählt.

**SCHEDULER_FLAG_USEALL (0×10)**

Ist dieses Flag gesetzt, so werden für die Planung alle Ressourcen gewählt, die alle Restriktionen der Veranstaltungen erfüllen. Es werden zuerst die Wünsche, dann die bevorzugten und zuletzt die möglichen Ressourcen gewählt.

**SCHEDULER_FLAG_PRIORITY_TIME (0×20)**

Sollen die Zeitwünsche bei der Planung berücksichtigt werden, so ist dieses Flag zu setzen. Dabei können auch Zeitwünsche definiert worden sein, die außerhalb der vorher festgelegten, bevorzugten Woche liegen.

**SCHEDULER_FLAG_PRIORITY_ROOM (0×40)**

Ist das Flag gesetzt, so bekommen die Räume eine höhere Priorität als die Dozenten. Es sollte gesetzt werden, wenn Räume die knappe Ressource sind.

**SCHEDULER_FLAG_PRIORITY_TEACHER (0×80)**

Ist das Flag gesetzt, so bekommen die Dozenten eine höhere Priorität als die Räume. Es sollte gesetzt werden, wenn die Dozenten die knappe Ressource sind.

Ist nicht das Flag SCHEDULER_FLAG_PRIORITY_ROOM angegeben, so werden standardmäßig die Dozenten als bevorzugte Ressourcen gewählt.

**SCHEDULER_FLAG_LOG_STANDARD (0×100)**

Ist das Flag gesetzt, so werden Nachrichten über den aktuellen Status der Planung generiert. Allerdings muss dazu noch eine Funktion mit void setLogEvent(SCHEDULER_LOG_FUNC value) angegeben werden, über die die Anwendung über neue Nachrichten informiert wird.
C.2. EIGENSCHAFTEN UND KONSTANTEN

**schedulerFlagLogDetailed (0x200)**


**schedulerFlagDefault (MAXWORD)**


```c
schedulerLogFunc getLogEvent (void)
void setLogEvent (schedulerLogFunc value)
```

Mit der Eigenschaft kann eine Funktion bestimmt werden, die im Falle einer neu generierten Nachricht aufgerufen wird.

```c
 scheduelerEventProcessedListFunc -
 void setschedulerProcessedListEvent (void)
```

Mit der Eigenschaft kann eine Funktion definiert werden, die immer dann aufgerufen wird, wenn die automatische Planung für eine Veranstaltung abgeschlossen ist.

```c
 WORD getMaxEventsPerDay (void)
void setMaxEventsPerDay (WORD value)
```

Mit der Eigenschaft kann festgelegt werden, wie viele Veranstaltungen pro Tag maximal eingeplant werden sollten. Der Wert dient nur als Richtwert und kann mangels Alternativen auch missachtet werden.

```c
 WORD getScheduleAs (void)
void setScheduleAs (WORD value)
```

Anhang D

Stundenplanungssysteme

Die beiden Stundenplanungssysteme ConTime 1.0 in der Programmiersprache CHIP und Kursplan 1.0 auf Basis des Constraintlösers CS/IλMR wurden ähnlich realisiert und bestehen aus einer Datenhaltungs-, Visualisierungs- und Planungskomponente.


In der Planungskomponente wird die Einhaltung der definierten Randbedingungen bei der Erzeugung eines korrekten Stundenplanes sichergestellt. Die definierten Randbedingungen werden durch Constraints repräsentiert. Auf diesen Constraints werden dann die Suchmethoden zur Stundenplanerzeugung ausgeführt.

Bei der Realisierung des Systems ConTime 1.0 wird die kommerzielle Programmiersprache CHIP benutzt, welche eine graphische Oberfläche zur Visualisierung und eine interne Objektstruktur zur Datenhaltung beinhaltet. Die Planungskomponente verwendet die in der kommerziellen Programmiersprache CHIP vorhandenen einfachen und globalen Constraints sowie die vorhandenen Methoden zur Unterstützung der Stundenplanerzeugung. Bei der Implementierung des Systems wurden die vorhandenen Komponenten zur Visualisierung, Datenhaltung und Planung genutzt, womit das System innerhalb einer Programmiersprache realisiert werden konnte.


In der Datenverwaltung werden die Daten beim Einlesen und Bearbeiten auf ihre Konsistenz geprüft. Hierfür wird bei jeder Veranstaltung die Verfügbarkeit der gewünschten und nutzbaren Dozenten, Räume und Zusatzbedingungen in der Problemdefinition kontrolliert.

In der nachfolgenden Abbildung D.1 sind die Hauptkomponenten der beiden Systeme für die Stundenplanung auf der Basis der kommerziellen Programmiersystems CHIP und des neuen Constraintlöasers $CS_{ld-MR}$ dargestellt.

Abbildung D.1: Komponenten der beiden Stundenplanungssysteme mit CHIP und dem Constraintlöser $CS_{ld-MR}$

Die Implementation der Systeme **ConTime 1.0** und **Kursplan 1.0** in den entsprechenden Programmiersprachen ist so realisiert, dass die maximale Verarbeitungsgeschwindigkeit bei geringst möglicher Speichernutzung gewährleistet wird. Dadurch konnten die zu behandelnden Planungsprobleme sehr groß werden, sich über mehrere Jahre erstrecken und in jeder einzelnen Woche verschiedene Stundenpläne besitzen.

Der Implementation des Systems **ConTime 1.0** in der kommerziellen Programmiersprache CHIP war nahelegend, um zu dem Constraintlöser in der Programmiersprache CHIP auch eine Visualisierungs- und Datenhaltungskomponente bereitstellen zu können. Bei der Nutzung einer Visualisierung und Datenhaltung außerhalb der Programmiersprache CHIP wäre der Austausch der Daten über die C-Schnittstelle der Programmiersprache CHIP notwendig. Die C-Schnittstelle stellt aber bei großen Datennmen einen Engpass dar und ist in der Nutzung des verfügbaren Speichers begrenzt.

Die Implementation des Systems **Kursplan 1.0** mit den beiden Programmiersprachen .NET und C++ bietet den Vorteil der schnellen Verarbeitung großer Datennmen bei einer einfachen Realisierung einer Visualisierung durch die vorhandenen Basisfunktionen in der Programmiersprache .NET.
Anhang E

Diagramme von ConTime 1.0 und Kursplan 1.0


Begonnen wird mit den Diagrammen des Planungssystems ConTime 1.0, welches die constraintlogische Programmiersprache CHIP der französischen Firma Cosytec nutzt.


E.1 ConTime 1.0

AMD-Prozessor

Abbildung E.1: Darstellung der Messergebnisse von ConTime 1.0 im Säulendiagramm mit eingeschränkter (oben) und maximaler (unten) Y-Achse
Abbildung E.2: Darstellung der Messergebnisse von ConTime 1.0 im Linienendiagramm mit eingeschränkter (oben) und maximaler (unten) Y-Achse
E.1. CONTIME 1.0

Pentium-Prozessor

Abbildung E.3: Darstellung der Messergebnisse von ConTime 1.0 im Säulendiagramm mit eingeschränkter (oben) und maximaler (unten) Y-Achse
Abbildung E.4: Darstellung der Messergebnisse von ConTime 1.0 im Linien Diagramm mit eingeschränkter (oben) und maximaler (unten) Y-Achse
E.2 Kursplan 1.0

AMD-Prozessor

Abbildung E.5: Darstellung der Messergebnisse von Kursplan 1.0 (Athlon - Prozessor 700 Mhz, 256 MB RAM)
Pentium-Prozessor

Abbildung E.6: Darstellung der Messergebnisse von Kursplan 1.0 (Pentium-4M - Prozessor, 1,4 Ghz, 256 MB RAM)
E.3 Vergleich von ConTime 1.0 und Kursplan 1.0

Pentium-Prozessor

Abbildung E.7: Vergleich ConTime 1.0 und Kursplan 1.0 mit eingeschränkter (oben) und maximaler (unten) Y-Achse