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 mit getragen hat.
Für die Unterstützung bei der Erstellung dieser Arbeit möchte ich Herrn Dr. Jochen Kramer, dem Geschäftsführer der ASCI Systemhaus GmbH und Frau Dr. Bolemant, der wissenschaftlichen Mitarbeiterin der ASCI Systemhaus GmbH danken, die mir mit vielfältigen Anregungen und Hilfen für meine Forschung zur Seite stand.
Einen großen Anteil am Zustandekommen dieser Arbeit hat meine Familie. Neben meinen Eltern, die mich stets in meinem Werdegang bestärkten und unterstützten, gilt mein besonderer Dank meiner Frau Manuela, die mir in den letzten Monaten viele alltägliche Arbeiten abnahm und somit ermöglichte, dass ich diese Arbeit erstellen konnte.

Kurzfassung

In dieser Arbeit wird ein Constraintlöser für diskrete mehrdimensionale Multiressourcenprobleme vorgestellt. Zu diesen Problemen gehören Planungs- und Platzierungsprobleme aus den Bereichen Produktionsplanung, Personalplanung, Stundenplanung, Kursplanung, Containerpackung, Routenplanung oder Fahrplanerstellung für Verkehrsunternehmen.

Mit zur Zeit verfügbaren kommerziellen Constraintlösern steigt der Lösungsaufwand von Multiressourcenproblemen mit der Problemgröße exponentiell an. Durch den entwickelten und in dieser Arbeit beschriebenen Constraintlöser für Multiressourcenprobleme wird dieser Aufwand für eine wesentlich erweiterte Problemgröße signifikant verringert. In der Arbeit wird an einem realen Anwendungsbeispiel gezeigt, dass die starke Verringerung des Anstiegs des Lösungsaufwands auch für große Probleme realisierbar wird durch die Problemzerlegung des mehrdimensionalen Multiressourcenproblems in mindestens einer Dimension, durch die Einführung einer zentralen Konsistenzsicherung und Bereichsverwaltung in speziellen Wertepräsentationen der Domänenvariablen und durch die Implementation von speziellen globalen Constraints für die Modellierung von Ressourcen und Reihenfolgenbeziehungen.

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.

Zur Konsistenzsicherung der Daten bei der Lösungssuche und der Propagation auf den Wertebereichen der Ressourcen erfolgt, wie in herkömmlichen Constraintlösern üblich, kein Wechsel zur komplexitätsreduzierenden Intervallpropagation mit ausschließlicher Einschränkung der Werte an den Intervallgrenzen, sondern es wird auch bei großen Wertebereichen die wertgenaue Einschränkungsmethode angewendet. Dadurch werden auch kleinste Inkonsistenzen (Löcher) in sonst kontinuierlichen Wertebereichen erkannt.

Der Constraintlöser für Multiressourcenprobleme beinhaltet nur einige einfache Constraints und zwei globale Constraints, um mit einer möglichst geringen Anzahl an Constraints auszukommen. Damit verringert sich der Verwaltungsaufwand der Constraints. Zu den globalen Constraints gehören das Reihenfolgeconstraint RC, womit die Abfolge von Ereignissen festgelegt werden kann und das Constraint diffn2D(), welches die überlappungsfreie Zuordnung zu Ressourcen sicherstellt. Das globale Constraint diffn2D() kann in der derzeitigen Auslegung zweidimensionale Wertebereiche mit einer jeweiligen Ausdehnung von bis zu 10 Millionen Werten je Dimension ohne Intervallverarbeitung bei geringem Speicherverbrauch verwalten. Zur Lösung von Multiressourcenproblemen kann das zweidimensionale globale Constraint diffn2D() mehrfach kombiniert werden. Hierbei wird eine Dimension als Bezugsdimension genutzt.

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 Constraintlösers kann deshalb unmittelbar die Vorteile einer Hardware mit 64 Bit-Architektur nutzen und damit zu einer weiteren Beschleunigung des Planungsprozesses beitragen.

In der vorliegenden Arbeit werden die Eigenschaften des Constraintlöser für mehrdimensionale Multiressourcenprobleme am Problem der mehrjährigen Kursplanung für Weiterbildungseinrichtungen erläutert. Die Implementation des Constraintlösers erfolgte in C^{++}. 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 Eisenbahnnetzen getestet worden.

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 multiresource 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 diffn2D(), which ensures the non-overlapping allocation of resources. In its current form, the global constraint diffn2D() 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 diffn2D() can be multipally 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

Kapitel 1

Einleitung

Multiressourcenproblemen treten in allen Bereichen auf, in denen mehrere Ressourcen, wie zum Beispiel Zeit, Räume, Maschinen, Züge, Gleise oder menschliche Arbeitskraft nur sehr begrenzt zur Verfügung stehen. Der optimale Einsatz dieser Ressourcen ist aus wirtschaftlichen Gründen meist zwingend erforderlich. Hier ist eine sorgfältige Planung extrem wichtig und wird immer bei der Bearbeitung komplexer Aufgabenstellungen, die vielfältige Randbedingungen (Constraints) umfassen, besonders schwierig. Beispielhaft für solche Planungsprozesse sind die Stundenplanung, Raumbelegungsplanung, Schichtplanung, Werkstattplanung, Erstellung von robusten Fahrplänen durch die Fahrplan- und Betriebssimulation.
Es gibt für derartige Probleme verschiedene Softwarelösungen, doch mit zunehmender Komplexität und Detailliertheit 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.

Für die Lösung von hochkomplexen, kombinatorischen Planungs- und Simulationsproblemen, mit einer Vielzahl nicht zu überschauender Nebenbedingungen, eignen sich Constraint-logische Verfahren, um gute Pläne mit geringem Aufwand zu berechnen oder um überhaupt Lösungen für sehr komplexe Probleme zu finden. Die Nutzung von Constraint-basierten Verfahren liefert im Allgemeinen nach sehr kurzen Abarbeitungszeiten konsistente und gute Lösungen. Es gelingt durch diese Methode, mit Ressourcen - Energie, Material oder auch Personal - schonend umzugehen, um die Umwelt weniger zu belasten, den Forderungen der Verbraucher zu entsprechen und auch wettbewerbsfähig zu sein. An Beispielen der Stundenplanung für Universitäten, Begrenzung der Elektro-Spitzenlast in der Multiressourcenplanung, bei der Konfiguration technischer Geräte mit optimalem Energieverbrauch und am Beispiel der Materialbedarfsplanung wurde die Methodik schon erfolgreich demonstriert [32]. Aus diesen Gründen ist die Verwendung von Constraint-basierten Verfahren und die Entwicklung eines speziellen Constraintlösers für Multiressourcenprobleme sinnvoll und notwendig.

Die Multiressourcenplanung lässt sich gut am Beispiel eines Stundenplanproblems darstellen. In einer Schule, Universität oder Weiterbildungseinrichtung müssen für die Durchführung von Lehrveranstaltungen Ressourcen in Form von Lehrpersonal und Unterrichtsräumen bereitgestellt werden. Die hierfür notwendigen zeitlichen Planungen der Lehrveranstaltungen, des dazugehörigen Lehrpersonals und den hierfür benötigten Unterrichtsräume können als Zuordnungsproblem modellieren werden, wobei bestehende Restriktionen und individuelle Besonderheiten zu beachten sind. Dem Stundenplan werden die Lehrveranstaltungen zu Zeitpunkten unter Einhaltung der bestehenden Restriktionen zugeordnet. Ein Stundenplanungsproblem ist im allgemeinen ein sehr komplexes Problem und gehört zu der Klasse der NP-vollständigen Probleme [20]. Die existierenden Stundenplanungssysteme schränken häufig die Anzahl der Varianten von erzeugbaren Plänen sehr stark ein. Meist wird nur eine Variante eines Stundenplans erzeugt. Zur Lösung dieses Problems bietet sich das neue Paradigma der Constraint-basierten Verfahren an. Die geltenden Restriktionen, die individuellen Besonderheiten und die Ressourcennut-

zung lassen sich mit Constraints modellieren. Das Paradigma der Constraint-basierten Programmierung unterscheidet sich von der klassischen Programmierung durch die Art und Weise der Verarbeitung der Constraints. Diese Verarbeitung der Constraints führt zu einer großen Suchraumeinschränkung, ohne Verlust von gültigen Lösungen. Diese große Einschränkung des Suchraums und die schnelle Erzeugung von Lösungen, die nahe am Optimum liegen, sind die entscheidenden Vorteile der Constraint-basierten Programmierung.

Das Paradigma der Constraint-basierten Programmierung wurde in kommerziellen Programmiersystem verfügbar gemacht. Die bekanntesten Programmiersysteme sind ILOG, CHIP5, SICSTus-Prolog und ECL^iPS^e. Diese kommerziellen Programmiersysteme beinhalten eine Vielzahl von einzelnen Constraints für verschiedene Anwendungen. In der großen Anzahl der verfügbaren Constraints liegt auch das Defizit der Programmiersysteme begründet. Die einzelnen Varianten der Constraints erfordern eine enorme Verwaltungsstruktur, welche die Effizenz der Abarbeitung und die verarbeitbare Problemgröße einschränkt. Zum Beispiel gibt es im Programmiersystem CHIP5 für das globale Constraint diffn() neun Varianten. Des Weiteren sind diese Systeme in der Nutzung des Speichers begrenzt. Das Programmiersystem SICSTus kann 256 MB und das Programmiersystem CHIP5 kann maximal 1,8 Gbyte als Arbeitsspeicher zur Problemlösung benutzen. Hinzukommt, dass die Anzahl der nutzbaren Variablen, Atome und Terme begrenzt ist. Bei den meisten Programmiersystemen geht die Verarbeitung von einzelnen Werten in eine Intervallverarbeitung über, bei der nur noch die Intervallgrenzen und nicht die einzelnen Werte selbst betrachtet werden. Im Programmiersystem CHIP5 erfolgt dieser Übergang bei einer Domänengröße von über 100.000 Werten in den Variablen.

Ziel der Arbeit ist es, die drei Defizite der begrenzten Speichernutzung, der eingeschränkten Variablenanzahl und der automatischen Umschaltung auf Intervallbetrachtung bei den Werten der Variablen durch einen neu entwickelten Constraintlöser aufzuheben. Der Constraintlöser für Multiressourcenprobleme beinhaltet nur einige einfache Constraints und zwei globale Constraints, um mit einer möglichst geringen Anzahl an Constraints auszukommen. Damit verringert sich der Verwaltungsaufwand der Constraints. Zu den globalen Constraints gehören das Reihenfolgeconstraint RC, womit die Abfolge von Ereignissen festgelegt werden kann und das Constraint diffn2D(), welches die überlappungsfreie Zuordnung zu Ressourcen sicherstellt. Das globale Constraint diffn2D() kann in der derzeitigen Auslegung zweidimensionale Wertebereiche mit einer jeweiligen Ausdehnung von bis zu 10 Millionen Werten je Dimension ohne Intervallverarbeitung bei geringem Speicherverbrauch verwalten. Zur Lösung von Multiressourcenproblemen kann das zweidimensionale globale Constraint diffn2D() mehrfach kombiniert werden. Hierbei wird eine Dimension als Bezugsdimension genutzt.

Um die Zeit zur Lösung des Multiressourcenproblems gering zu halten, wurden zwei neue Methoden zur Problemmodellierung entwickelt und alternative Propagationstechniken verwendet. In dieser Arbeit werden die Modellierungsmethode der hierarchisch geordneten Nichtzeiten und der Problemzerlegung zur Problemreduktion vorgestellt. Bei der Nichtzeitenhierarchie wird die Eigenschaft der Kontinuität der Nichtzeiten ausgenutzt. Die Nichtzeiten sind meist in einer Hierarchie abbildbar und gelten in wiederkehrenden Zeiträumen. Durch die Kontinuität von Nichtzeiten, wie zum Beispiel an Wochenenden, bei Arbeit- und Freizeit entsteht eine Strukturierung des Suchraums des Planungsproblems.
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.
Für die Constraintpropagation werden Forward- und Look-ahead-Algorithmen als Alternative zum Backtracking diskutiert. Die Forward- und Look-ahead-Algorithmen betrachten Auswirkungen der derzeitigen Aktion in Bezug auf die Lösbarkeit des Gesamtproblems in der Zukunft. Beim Backtracking hingegen werden die Entscheidungen zurückgenommen und andere Alternativen ausprobiert.

Zusammengefasst werden die neuen Modellierungsmethoden in einem Constraintlöser für spezielle Mul-

tiressourcenprobleme, der eine globale Konsistenztechnik und globale Constraints für die Modellierung von Ressourcen- und Reihenfolgebeziehungen enthält. Mit diesem Constraintlöser kann der Aufwand beim Lösen großer praktischer Probleme verringert werden. Somit können spezielle Multiressourcenprobleme mit vertretbarem Zeitaufwand berechnet werden.
Aus dieser Art der Problemmodellierung und -lösung lässt sich ein System mit entscheidungsunterstützender Funktion realisieren. Die sofortige Domänreduktion und Constraintpropagation erlauben die Planung im Bereich gültiger Lösungen und die ständige Kontrolle der noch vorhandenen gültigen Varianten der Planung.
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 Detailierungsgrad, die über einen großen Zeitraum geplant werden sollen. Zu diesen Problemen zählen zum Beispiel: Jahresproduktionspläne in Minutengenauigkeit, Jahresfahrpläne für den offfentlichen 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.

Ein Beispiel für das Stundenplanungsproblem ist die Erstellung von Stundenplänen in Weiterbildungseinrichtungen, die heute meist noch von Hand geschieht. Hierbei werden auf einer Tafel Kärtchen mit Lehrer-, Fach-, Raum- und Stundennamen so lange verschoben, bis ein Stundenplan gefunden wurde, bei dem jede Unterrichtsstunde in einem Raum, bei einem Lehrer und zu einer Zeit zu sein scheint. Die vielen Randbedingungen, so genannte Constraints, die die Stundenplanung zu einer komplizierten und langwierigen Arbeit machen, beinhalten auch eine große Anzahl von Fehlermöglichkeiten. Die Erstellung des Stundenplans nimmt Tage oder Wochen in Anspruch. Mit der Entwicklung eines Planungssystems, das die manuelle Planung ablöst und die Erstellungszeit widerspruchsfreier Stundenpläne drastisch reduziert, kann der Planer entlastet werden. Dann ist es auch möglich, auf kurzfristig eingetretene Änderungen mit zeitnah erstellten Vertretungsplänen zu reagieren. Zur Zeit gibt es am Markt keine Planungsysteme für Stundenpläne, die alle Randbedingungen und die gesetzlich vorgeschriebenen minimalen und maximalen Personalauslastungen beachten sowie in kurzer Zeit einen optimierten Stundenplan erstellen. Bei der Erstellung der Stundenpläne sind Freistunden der Schüler und Lehrer, die Raumkapazitäten, die Fachzuordnungen der Lehrer und die Arbeitszeitvorschriften gleichzeitig zu beachten.

2.2 Das Stundenplanungsproblem als Zuordnungsproblem

Bevor das Stundenplanungsproblem als Zuordnungsproblem ausführlich betrachtet wird, soll beispielhaft das Zuordnungsproblem der Ressourcenbelegungsplanung eines Unternehmens betrachtet werden. Die Ressourcenbelegungsplanung eines Unternehmens wird als ein kontinuierliches Zuordnungsproblem angesehen (vgl. Puppe [66]), bei dem beliebiger Bedarf passendem Bestand zuzuordnen ist .
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 Nachfrageobjekten 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 Fertigungsschritte 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 zu verarbeitenden Daten bestehen aus den Grunddaten und den dispositiven Daten, die das für die Planung notwendige Wissen der Disponenten enthalten, welches in Form von Restriktionen und Wünschen vorliegt. Durch die Restriktionen werden die Bedingungen formuliert, die im Lauf der Planung eingehalten werden müssen (harte Constraints) oder sollten (weiche Constraints) . Die harten Constraints beschreiben z.B. Anforderungen der Fertigung und dürfen im Verlauf der Planung nicht verändert werden. Im Unterschied dazu beschreiben die weichen Constraints Bedingungen oder Wünsche, die beachtet werden können. Diese weichen Bedingungen (Wünsche) können mit einer Priorität versehen werden, die das Maß der Verletzung der weichen Constraints angibt. Die Summe der Prioritäten der verletzten Constraints soll während der Planungsprozesse minimiert werden.
Das Paradigma der Constraint-basierten Programmierung unterscheidet sich wesentlich von dem der klassischen Programmierung. Die Probleme werden in der Constraintprogrammierung durch eine Menge von Variablen, Funktionen und Relationen zwischen den Variablen dargestellt. Die Relationen beschreiben die Restriktionen und Bedingungen (Constraints), die an eine zulässige Variablenbelegung mit Werten gestellt wird. In klassischen Systemen werden die Bedingungen erst nach dem Belegen der Variablen berechnet. Dieses ist das bekannte Trial-and-Error-Verfahren, bei dem die Einhaltung der Bedingungen am Ende des Planungsvorgangs geprüft werden. In der Constraintprogrammierung erfolgt erst die Behandlung der Constraints und danach die Ermittlung der Variablenbelegungen. Die Constraints mit den teilweise gemeinsamen Variablen, bilden ein Constraintnetz. In diesem Constraintnetz sind die Variablen als Knoten und die relationalen Beziehungen als Kanten dargestellt. Bei der Einschränkung der Domäne einer Variablen wird durch den Constraintlöser sofort die Auswirkungen für andere Variablen berechnet. Die betroffenen Variablen stehen zueinander über Constraints in Verbindung. Dieser Vorgang nennt sich Constraint-Propagation und löscht aus den Domänen der Variablen die Werte, die das Constraintnetz nicht erfüllen können. Eine Sackgasse liegt vor, wenn eine Domäne einer Variable keinen Wert mehr enthält. In diesem Fall wird durch Rücksetzen (Backtracking) zu einer alternativen Einschränkung, in Verbindung mit Suche, die Sackgasse umgangen. Vorteilhaft ist, dass diese Situation frühzeitig erkannt wird, bevor alle Variablen mit Werten belegt wurden. Hieraus resultiert die effiziente Abarbeitung bei vielen zu berücksichtigen Constraints. Das Verfahren der Constraint-Propagation verringert den Suchraum stark, ohne gültige Lösungen abzuschneiden. Nur durch die Propagtion der Constraints selbst kann in einzelnen Fällen schon eine Lösung erstellt werden. Eine Lösung liegt vor, wenn alle Domänen der Variablen genau einen Wert haben. Diese eindeutige Lösung kann, bei einem in der Regeln durch die Propagation eingeschränkten Suchraum, im allgemeinen durch einen Suchprozess (Labeling) unter zu Hilfenahme von Heuristiken erzeugt werden.

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 Gesamtlösung durch Vertauschung mit Zeitlimit

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).

Nach jedem Zuordnungsschritt im Schritt 4 können weitere weiche Constraints überprüft werden. Für den Fall, dass wichtige weiche Constraints verletzt sind, kann mit Vertauschungsprozeduren versucht werden, ob sie die Summe der verletzten weichen Constraints verringern würden. Hierfür können die bekannten Suchverfahren eingesetzt werden: Breitensuche, Tiefensuche, Iterative Tiefensuche, Hill-Climbing, Simulated Annealing, Beam-Search, Bestensuche, A* usw. Da vollständige Suchverfahren wegen der Größe des Suchraumes ausscheiden, muss man sich auf heuristische Suchverfahren beschränken, die zur Effizienzsteigerung durch gute Heuristiken geleitet werden müssen. Dieser Schritt ist in der Zeit begrenzt und behält bei nicht verbesserter Lösung die alte Lösung bei.
Danach wird mit dem nächsten Objekt fortgesetzt. Dadurch wird erreicht, dass der Algorithmus auch terminiert. Man trägt damit dem Umstand Rechnung, dass es bestimmte weiche Constraints gibt, für die in sinnvoller Zeit keine gute Auflösung gefunden werden kann, was in der Praxis ein bekanntes Phänomen ist, wenn nicht ein optimaler Plan gesucht wird, sondern im Normalfall auch ein weniger guter Plan ausreicht. Je nach zur Verfügung stehender Zeit kann man die Zeitschranken mehr oder weniger grosszügig gestalten und damit die Qualität des Planes beeinflussen, der umso besser wird, je mehr Suchzeit zugelassen wird.

Die Stundenplanung in Weiterbildungseinrichtungen und die Fahrplanerstellung im Eisenbahnverkehr können als Zuordnungsproblem abgebildet werden. Bei beiden Planungsproblemen liegen begrenzte Ressourcen in Form von Räumen, Lehrern, Zeiten oder Schienen, Blockabschnitten, Zügen und Zeiten vor. In dieser Arbeit wird das Stundenplanungsproblem als Multiressorcenproblem betrachtet und es werden Untersuchungen für die Umsetzung mit einem speziellen Constraintlöser für Multiressourcenprobleme durchgeführt. Die Anwendbarkeit für das Verkehrsproblem wird im Ausblick der Arbeit umrissen.

Bei der Stundenplanung soll nicht nur eine Zuordnung der Stunden zu Zeitpunkten erfolgen, sondern auch die Berücksichtigung der Ressourcenwünsche für die Zeitpunkte der Stunden, die Räume und die Lehrer zu einer Unterrrichtseinheit. Zusätzlich müssen noch die vorgenannten Bedingungen eingehalten werden. Deshalb liegt hier ein Multiressourcenproblem vor. Dieses Multiressourcenproblem wird durch die Vielzahl von verschiedenen Stunden über einen langen Zeitraum (2 Jahre) bei Weiterbildungseinrichtungen noch wesentlich schwieriger für den Planer handhabbar. Durch die Verwendung des Constraint-Paradigmas kann dieses Planungsproblem gelöst werden. Anders als in konventionellen Programmen wird nicht erst nach der Zuweisung von Werten zu Variablen, sondern bereits während der Zuweisungen von Werten die Überprüfung der Randbedingungen vorgenommen.

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 Lösungssuche erfordert einen zu langen Zeitraum, was eine operative Planung unmöglich macht. In dieser Arbeit wird ein Modell für einen Constraintlöser auf mehrdimensionalen, begrenzten Ressourcen definiert, der große Planungs- und Simulationenprobleme mit kurzen Antwortzeiten löst. Das Modell ist in verschiedenen Constraintdomänen anwendbar, weil die verwendeten Constraints kein domänspezifischen Wissen enthalten, sondern nur allgemeine mathematische Bedingungen effektiv umsetzen. Die Variablen der Constraints werden durch Propagationsmethoden schrittweise eingeschränkt. Zu diesen Propagationsmethoden gehören vorausschauende

Methoden wie `Forward checking` und `Look ahead` als auch Konsistenzsicherungsmethoden, die in globalen Constraints `diffn` realisiert sind. In diesem neuen Modell des Constraintlösers ist ein Algorithmus zur Problemzerlegung integriert, der die Struktur und Vernetzung der Problemdomänen auswertet. Die sich daraus ergebenen Teilprobleme werden dann mit dem Constraintlöser in kurzer Zeit gelöst.

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

Bei der verfügbaren Stundenplansoftware existiert zur Zeit eine starke Spezialisierung der einzelnen Produkte am Markt auf Zielgruppen. Die einzelnen Produkte sind meist nur von einer Zielgruppe anwendbar, weil die Oberfläche und damit die Eingabemasken für die spezielle Zielgruppe zugeschnitten wurden. Am Markt kann man vier Zielgruppen ausmachen.

Zur ersten Zielgruppe der **Schulen** gehören die Grundschulen, Realschulen, Hauptschulen, Gesamtschulen, Sonderschulen und Gymnasien. Die Stundenplanung ist ein einheitliches Problem für alle Schultypen. Diese Zielgruppe benötigt neben der allgemein gebräuchlichen Stundenplansoftware als Stundentafel, oft bundeslandspezifische Tools für die Statistik, Kursplanung, Notenverwaltung, Personal- und Schülerverwaltung. Der Umfang der Anforderungen variiert sehr stark in Abhängigkeit von Schüleranzahl, Schultyp und Standort der Schule.
Die zweite Zielgruppe den **Krankenpflegeschulen** benötigt zur allgemein gebräuchlichen Stundenplan- und Verwaltungssoftware, die Möglichkeit der Einbindung sehr vieler Praktikumsstandorte und Kurse sowie eine leistungsfähige Textverarbeitung mit vorgefertigten Formularen für die Erstellung von Zeugnissen und Zertifikaten.
Die dritte Gruppe bilden die **privaten Weiterbildungsträger**. Sie sind gekennzeichnet durch wechselnde Veranstaltungen in strenger Reihenfolge und benötigen vorgefertigte Stundenplanungs- und Verwaltungssoftware mit leistungsfähiger Textverarbeitung, Personalplanung, Kostenrechnung für Honorar- und Lohnabrechnung, die in der Regel komplett vom Anbieter auf die individuellen Bedürfnisse angepasst werden sollten.
Bei der vierten und letzten Zielgruppe den **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.

In allen vier Zielgruppen kommt eine mehr oder weniger komplizierte Stundenplanungskomponente zum Einsatz, die sich entsprechend der Zielgruppenanforderungen in der Leistungsfähigkeit der Planungskomponente unterscheidet. Bei kleinen Planungsproblemen mit ein bis drei Klassen, Kursen oder Semestern ist meist der Einsatz einer Stundenplanungssoftware nicht notwendig. Da die Komplexität des Planungsproblems mit weiteren Klassen, Kursen oder Semestern sehr schnell zunimmt, was auch durch die zusätzlich zu verwaltenden Ressourcen wie Lehrpersonal, Räumen und Unterrichtsmittel bedingt ist, ist der Einsatz einer Stundenplanungssoftware für eine schnelle und korrekte Planung zwingend erforderlich. In größeren Schulen mit ca. 20 Klassen dauert die Erstellung des ersten Basisstundenplans 14 Arbeitstage. Der Basisstundenplan ist dann noch nicht korrekt und kann so nicht durchgeführt werden, da im Stundenplan in der Regel noch Inkonsistenzen enthalten sind. In mehreren Konsolidierungsrunden mit allen Lehrern wird der Stundenplan dann weiter verbessert bis alle Konflikte in Form von Doppelbelegungen und Überschneidungen beseitigt sind.

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 Stundenplanungssoftware. Sie benötigen neben der automatischen und interaktiven Stundenplanerstellung, der schnellen und überschneidungsfreien Vertretungsplanerstellung auch die Möglichkeit der Editierbarkeit eines vorhandenen Stundenplans. Diese Editierbarkeit umfasst die Erstellung neuer Veranstaltung, das Löschen von Veranstaltungen und die Änderung von Daten der einzelnen Veranstaltungen.

Die privaten Weiterbildungsträger finanzieren sich größten Teils und haben damit ein gesteigertes Interesse an einer ressourcen- und kostenoptimierenden Stundenplanung. Die privaten Weiterbildungsträger haben den derzeitigen größten Bedarf an Stundenplanungssoftware. Es sind Ein- bis Zwei-Jahres-Pläne mit wöchendlich wechselnden Veranstaltungen erforderlich. Aus diesen Gründen war auch die Bereitschaft groß, für diese Arbeit praxisrelevante Daten mit erhöhtem Schwierigkeitsgrad zur Verfügung zu stellen.
Im folgenden Abschnitt werden die Funktionalitäten der Planungskomponente von bekannter Stundenplansoftware näher untersucht.

2.4 Übersicht über verfügbare Stundenplansoftware

Am Markt gibt es zur Zeit ca. 20 namhafte Hersteller von Stundenplanungssoftware. Die Stundenplanungssoftware wird zu Preisen von 9000,- EUR bis zu einigen 100,- EUR vertreiben. Der Preis richtet sich hierbei nach den zur Verfügung gestellten Funkionalitäten. Die Funktionalitäten reichen von der Stammdatenverwaltung von Schülern, Lehrern, Klassen, Veranstaltungen, Räume und Statistiken, über Notenverwaltung mit Klassenbuch und Zeugnisdruck, über die Korrespondenz mit Serienbrief-, Listen-, Formular- und Texteditor, über den Datentransfer mit Schnittstellen zu Datenbanken und Textformaten, über Inventarisierung und Archivierung, über verschiedene Formen der Visualisierung von Stundenpänen, bis zur automatischen und interaktiven Stundenplanung. Aus der Aufzählung der Funktionalitäten ist ersichtlich, dass der Stundenplanung noch keine große Bedeutung bei der Konzipierung der Stundenplansoftware geschenkt wird. Die Funktionalitäten der Verwaltung stehen meist im Vordergrund. Dieses Funktionalitäten der Verwaltung haben einen größeren Anteil bei der Entwicklung als die Planungskomponente. Dieses zeigt sich auch bei der Analyse der Planungsfunktionalitäten der Komponente der Stundenplansoftware.

Von den getesteten Softwaresystemen **gp-Units** von der Dialog Computer Systeme GmbH [21], **S-Plus** von der `Scientia GmbH` [72], **daVinci** von `Stüber Software` [73] und **Winschule** von `Schulsoftware A. Tillman` [79], werden die oben genannten Funktionalitäten der Stammdatenverwaltung, Notenverwaltung, Korrespondenz, Datenschutz, Inventarisierung, Archivierung und Stundenplanung abgebildet. Der Datentransfer ist bei dem preisgünstigsten Anbieter nicht enthalten. Die vier Anbieter gehören zu den Marktführern in Berlin und Deutschland. Sie liegen mit Ihren Preisen für eine Lightversion im Bereich von cirka 500,- EUR bis 3800,- EUR und für eine Vollversion im Bereich von cirka 600,- EUR bis 7600,- EUR.

Bei der näheren Betrachtung der Komponente für die Stundenplanung von den 20 Herstellern mit cirka 39 Produkten sind in 30 Produkten von 39 Produkten die Stammdatenverwaltung und Archivierung, in 21 Produkten von 39 Produkten die manuelle Stundenplanung, in 18 Produkten von 39 Produkten die automatische Stundenplanung, der Datenschutz, die Korrespondenz und die Notenverwaltung, in 16 Produkten von 39 Produkten die Erstellung von Vertretungsplänen enthalten. Es folgen die Funktionalitäten von Datentransfer und Inventarisierung, die in 10 bis 15 Produkten von 39 Produkten enthalten sind. In nur 6 Produkten von 39 Produkten ist ein frei editierbarer Stundenplandialog enthalten, mit

dem Veranstaltung angelegt, editiert und gelöscht werden können.
Bei der manuellen Stundenplanung können die Veranstaltungen auf einer visualisierten Stundenplantafel platziert werden. Im Modus der automatischen Planung übernimmt die Platzierung der Veranstaltungen die Stundeplanungssoftware.
Wie beschrieben, besitzt nicht einmal die Hälfte der Produkte eine automatische Planungskomponente. Das geringe Vorkommen von Komponenten zur Stundenplanung von 53,84 Prozent bei der manuellen und 46,15 Prozent bei der automatischer Stundenplanung, zeigen einen vorhandenen Bedarf an automatischen Planungsalgorithmen für das Stundenplanungsproblem. Die Komponente zur freien Editierbarkeit in einem Planungsdialog wird nur von 15,38 Prozent der verfügbaren Produkte angeboten.

In keinem der Stundenplanungssysteme ist eine Komponente mit Informationen zur Unterstützung der Entscheidung beim Stundenplanungsprozess vorhanden. Hierbei werden dem Nutzer die noch verfügbaren Ressourcen, wie freie Räume und Dozenten zu allen noch möglichen Zeitpunkten angezeigt. Für jeden einzelnen Zeitpunkt wird für jeden Raum und Dozenten angeben, ob er verfügbar ist. Dies erfolgt unter Beachtung aller vorher dem System bekannt gegebenen Randbedingungen, die während des Planungsprozesses beachtet werden sollen. Mit Hilfe dieser Entscheidungsunterstützung kann der Planer kurzfristig einen konsistenten Stundenplan erstellen, unter Beachtung aller vorgegebenen Randbedingungen.

Die Umsetzung der Komponente mit der Funktion zur Entscheidungsunterstützung lässt sich mit Hilfe von Constraint-Verfahren für Multiressourcenprobleme realisieren. Die Bedingungen werden in entsprechende Constraints überführt und die möglichen Zeitpunkte werden in Domänen abgebildet. Die verschiedenen Ressourcen (Räume, Dozenten, Gruppen, Kurse, Zeit usw.) bilden die einzelnen Dimensionen des Multiressourcenproblems.

2.5 Constraintlöser für Multiressourcenprobleme

Ein Verfahren mit diversen Anwendungsfeldern ist das Lösen von `Constraint satisfaction Problemen` (CSP). Die CSP bestehen aus einer Menge von Bedingungen (Constraints), durch die ein gesuchter Lösungszustand beschrieben wird, in dem alle Constraints erfüllt sein müssen. Im Allgemeinen sind die CSP's NP-vollständig, so dass im Gebiet der künstlichen Intelligenz nach Lösungsverfahren gesucht wird, die schnell genug für die einzelnen Anwendungsgebiete sind und eine oder mehrere Lösungen finden. CSP's können auch durch Constraintlöser bearbeitet werden. Die Constraintlöser generieren aus der domänspezifischen Beschreibung des Problems eine Lösung durch die Anwendung eines generischen Algorithmus auf das gegebene Constraintmodell. Die meisten Planungsprobleme lassen sich mit ganzen Zahlen modellieren, die dann mit `finite domain` Constraintlösern gelöst werden können. Im Constraintlösungsprozess werden den Variablen der Constraints Werte zugeordnet. Die Wertezuordnung erfolgt so, dass alle angegebenen Bedingungen (Constraints) erfüllt werden. Durch die Interaktion mit dem Benutzer können sich die Bedingungen ändern. Das Ändern der Bedingungen kann meistens als Hinzufügen von Constraints umgesetzt werden. Bei komplexen Änderungen oder der Rücknahme von Constraints wird das Constraintmodell unter Speicherung der schon festgelegten Einplanungen neu generiert. Das Hinzufügen von Constraints wird im Constraintlöser durch inkrementelle Propagation realisiert. Dabei bilden die aktuellen Domainenwerte der Variablen die Grundlage für die Propagagtion der Konsequenzen eines zusätzlichen Constraints.

Die Entwicklung eines Constraintsystems zur Lösung von Ressourcenproblemen ist aus der Sicht folgender derzeitiger Probleme mit den vorhandenen kommerziellen Constraintlösern sinnvoll. Die beiden größten und weltweit verbreiteten Anbieter von Constraint-logischen Programmiersystemen sind die beiden französischen Firmen ILOG und COSYTEC.
Die Produkte ILOG-Scheduler und CHIP5 von COSYTEC liegen in einem hohen Preisniveau und sind somit für Anwendungen im mittleren und unteren Preissegment aufgrund der hohen Lizenzkosten ungeeignet. Die Verwendung von günstigeren Constraint-logischen Programmiersystemen wie zum Beispiel SICSTus-Prolog oder ECL^iPS^e sind aufgrund ihres geringeren Funktionsumfangs gegenüber den bei-

den marktführenden Produkten, durch die Umsetzung von globalen Constraint nur in Basisvarianten, keine Alternative. Denn die globalen Constraints im ILOG-Scheduler und in CHIP5 ermöglichen eine bessere Modellierung und eine effizientere Suche bei der Problemlösung.
Die kommerziellen Constraintlöser sollen für möglichst viele Anwendungsfälle einsetzbar sein. Aus diesem Grund bestehen die entsprechenden Systeme meist aus mehreren Constraintlösern und beinhalten viele einzelne Constraints. Diese Constraintsysteme sind somit sehr umfangreich und enthalten für viele globale Constraints zahlreiche Features. Bei einer konkreten Anwendung werden aber nicht alle Features benötigt, die Verwaltung der Features und der verschiedenen Constraints führt zu einer Reduzierung der maximalen Bearbeitungsgeschwindigkeit.
Die kombinatorischen Probleme haben die Eigenschaft einen exponentiellen Aufwand bei ihrer Lösung zu benötigen. Man spricht in diesem Fall von NP-harten Problemen. In der Abbildung 2.1 ist als dicke Linie der exponentielle Aufwand dargestellt. Zum Vergleich wird in der gestrichelte Linie der lineare Aufwand ausgewiesen.

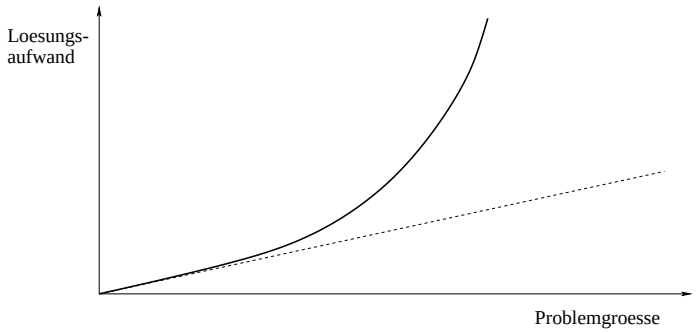

Abbildung 2.1: Lösungsaufwand in Abhängigkeit von der Problemgröße

Das kommerzielle Constraintsystem CHIP5, welches den besten Kompromiss zwischen Kosten und Funktionsumfang darstellt, ist aber durch die Auslegung auf 32-Bit Adressierung in der Anzahl der nutzbaren Variablen begrenzt. Die Begrenzung auf 32-Bit Adressierung zieht eine Reduzierung der verfügbaren Anzahl der Variablen und der Objektinstanzen nach sich und macht den Einsatz des Constraintsystems bei großen Projekten nur durch erhöhten Aufwand, durch den Einsatz einer Problemreduktion oder Problemzerlegung möglich. Zu den großen Problemen zählen Stundenplanungsprobleme über mehreren Wochen bis zu einem Jahre oder die Modellierung von Fahrplänen in großen Eisenbahnnetzen.
Bei den meisten kommerziellen Anwendungsfällen von Constraintsystemen handelt es sich um die Planung oder Simulation von begrenzten Ressourcen. Ein Constraintsystem, welches sich auf die Behandlung von größeren Ressourcenproblemen in kürzerer Zeit spezialisiert hat, ist zur Zeit nicht verfügbar.

Zur Suche von Lösungen in Constraintnetzen werden unter anderem die Suchalgorithmen Genetische Algorithmen, Simulated Annealing und Constraintpropagation mit Domänreduktion verwendet. In dem White Paper [45] der Firma Ilog kommt man zu dem gleichen Ergebnis, dass für die meisten Anwendungsfälle Constraintpropagation mit Domänreduktion eine schnelle und effektive Suche ermöglicht. Ähnliche Schlussfolgerungen im Bereich des Job-Shop-Scheduling zieht Goltz [34] in seinen Untersuchungen.

Kapitel 3

Constraint Logische Programmierung

3.1 Constraint-Programmierung

Die in der Constraint-Programmierung verwendeten Programmiersprachen gehören zu den deklarativen Programmiersprachen. Sie entwickelten sich Anfang der achtziger Jahre aus den logischen Sprachen, die heutzutage als eine Teilmenge der Constraintsprachen betrachtet werden. Es gibt derzeit viele Constraintsprachen mit verschiedenen Constraintdomänen und den entsprechenden Lösungsmechanismen. Die Constraint-Programmierung ist zu einem bedeutenden und weiter wachsenden Forschungs- und Anwendungsfeld geworden. Die Anwendungsfelder der Constraint-Programmierung liegen vorwiegend in der Modellierung, Simulation und im Lösen komplexer, kombinatorischer Probleme.
In diesem Kapitel wird das derzeitige Verständnisses zur Constraint-Programmierung nach Bartak [6], Cosytec [24] und Hofstedt und Wolf [43] zusammenfassend vorgestellt. Die Definitionen werden aus der Literatur zitiert, teilweise verallgemeinert und kommentiert, um sie zur Beschreibung eines Constraintlösers für Ressourcen anwenden zu können. Im darauf folgenden Kapitel wird dann ein spezielles Constraintsystem zur Lösung von Ressourcenproblemen neu definiert.

3.1.1 Erweiterung logischer Sprachen um build-in-Prädikate

Die Constraint-Programmierung entwickelte sich aus der Erweiterung der logischen Sprachen zunächst um built-in-Prädikate und später um Constraints.
Zu den build-in-Prädikaten, dass in Prolog neben den logischen weiterhin auch arithmetische Berechnungen realisiert, zählt zum Beispiel is/2. Dieses build-in-Prädikat wird zur Auswertung von Operationen (+, -, *, /, div, mod usw.) angewendet. Hier wird von der Auswertung mittels Unifikation abgewichen. Die rechte Seite eines solchen Ausdrucks `?- X is 4 + 1` wird zuerst ausgewertet. Hierfür müssen alle auf der rechten Seite vorkommenden Variablen gebunden sein, sonst tritt ein Laufzeitfehler auf. Danach wird das Ergebnis mit der linken Seite des Ausdrucks unifiziert.

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:

$$q(s_1,...,s_m) :- q_1(s_{1,1}, ..., s_{1,n}), ..., q_k(s_{k,1}, ..., s_{k,r}).,\ k \geq 0,$$

wobei s_i, $s_{i,j}$, Terme sind und q, q_i Prädikat-Symbole. Im Folgenden wird dafür Q:- Q_1,...,Q_k. geschrieben. Q_i nennt man dabei ein Literal. Eine Regel Q:- Q_1,...,Q_k. mit k > 0 nennt man Klausel und eine Regel Q:- Q_1,...,Q_k. mit k $= 0$, also Q., wird als Fakt bezeichnet.

Bei jeder Regel Q:- Q_1,...,Q_k. heißt Q Kopf (head) der Regel und Q_1,...,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:- Q_1,...,Q_k. ist als $\forall\ X\ .Q \leftarrow Q_1 \wedge ... \wedge Q_k$ zu interpretieren, mit X als der Menge der in Q, Q_i vorkommenden Variablen (siehe auch [43]).

Auf Grund der logischen Äquivalenz von a $\leftarrow$ b und a $\vee \neg$ b ergibt sich auch $\forall\ X\ .Q \vee \neg\ (Q_1 \wedge ... \wedge Q_k)$bzw. $\forall\ X\ .(Q \vee \neg\ Q_1 \wedge ... \wedge \neg\ Q_k)$. Der Fakt Q. repräsentiert die Formel $\forall\ X\ .Q$. Die Regeln und Fakten sind somit universell quantifizierte Disjunktionen von Literalen, bei denen genau ein Literal positiv (erfüllt / true) ist. Eine Horn-Klausel ist eine universell 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 ?- R_1,...,R_l. und damit wird die Formel $\forall\ X\ .(\neg R_1 \vee ... \vee \neg\ R_l)$ repräsentiert. Die leere Klausel ?-. gibt die Formel false $\leftarrow$ true, das heißt false wieder. Bei der Auswertung des logischen Programms P mit dem Ziel G = (?-R_1,...,R_l.) wird versucht, eine Widerlegung von G aus P unter Nutzung der (SLD-)Resolution zu finden. Hierbei wird geprüft, ob $\neg$ 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 Antwortsubstitution σ ermitteln, so dass P $\models \forall\ \sigma\ (R_1 \vee ... \vee R_l)$ 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, allgemeinster Unifikator, mgu** (nach [43]): Seien s und t Terme. Eine Substitution σ mit σ(s) und σ(t) wird Unifikation von s und t genannt. Ein Unifikator σ von s und t wird allgemeinster Unifikator (most general unifier, mgu) von s und t genannt, wenn es für jeden Unifikator ϕ von s und t eine Substitution ψ gibt, so dass $\phi = \psi$ o σ gilt.

Beispiel [43]: Gegeben seien die Terme s = f(x, g(z)) und t = f(1, y). Es gilt σ(s) = f(1,g(z)) = σ(t). Dann sei die Substitution σ = x/1, y/g(z) der Unifikator von s und t.
Gleichfalls ist die Substitution ϕ = x/1, y/g(2),z/2 ein Unifikator von s und t, denn es gilt ϕ(s) 0 f(1, g(2)) = ϕ(t).
In diesem Beispiel ist σ der allgemeinste Unifikator von s und t, da es für σ und ϕ ein ψ = z/2 gibt, so dass ψ o σ = ϕ ist.
Die allgemeinsten 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 = ?- R_1,...,R_l. mit l $\geq$ 1. Wenn es eine Variante C = (Q :- Q_1,...,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,...,l} und einen allgemeinsten Unifikator σ von R_i und Q gibt, dann sei G' =?-$\sigma(R_1)$,...,$\sigma(R_{i-1})$, $\sigma(Q_1)$,..., $\sigma(Q_m)$, $\sigma(R_{i+1})$,...,$\sigma(R_l)$. Dann wird $G \rightsquigarrow_{\sigma,C} G'$ ein Resolutionsschritt genannt. Wenn C klar aus dem Kontext hervor geht, kann auch $G \rightsquigarrow_{\sigma} G'$ geschrieben werden.

Mit der SLD-Resolution wird die Lineare Resolution mit einer Selektionsfunktion für Literale bezeichnet, bei der Definite 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 unerfüllbar 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,... von Zeilen, so dass $G \rightsquigarrow_{\sigma_1,C_1} G_1 \rightsquigarrow_{\sigma_2,C_2} G_2 \rightsquigarrow_{\sigma_3,C_3} G_3 \rightsquigarrow_{\sigma_4,C_4} ...$.

Definition **SLD-Refutation, SLD-Ableitung** (nach [43]): SLD-Refutation von (P,G) ist eine endliche SLD-Ableitung G, G_1, G_2 ,..., G_n von (P,G) mit n $\in$ N, so dass G_n = ?-. ist.

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

append([],X,X).
append([X|GX],GY,[X|GZ]) :- append(GX, GY, GZ).

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

?- append(A, [5|B], [7,5,1]).
$\rightsquigarrow_{\sigma_1}$ = {A/[7|GX1], X1/7, GY1/[5|B], GZ1/[5,1]}, (2)
?- append(GX1, [5|B],[5,1]).
$\rightsquigarrow_{\sigma_2}$ = {GX1/[],B/[1], X2/[5,1]}, (1)

In Prolog werden diese Konzepte umgesetzt. Mit der Selektionsfunktion wird das erste unbearbeitete Literal von links eines Ziels zur Ableitung ausgewählt. Für ein gewähltes Literal L in einem Ziel können mehrere Klauseln im Programm vorhanden sein, deren Köpfe mit L unifizierbar sind. Deshalb gibt es verschiedene Möglichkeiten bei der Regelauswahl zur SLD-Resolution. Somit gibt es für ein Ziel und ein Programm unterschiedliche SLD-Refutationen mit verschiedenen Antworten und nichtterminierende SLD-Ableitungen. Der Selektions-/ Auswertungsmechanismus wählt in PROLOG die erste passende Klausel des Programms zur Ableitung aus. Die unterschiedlichen Ableitungsfolgen lassen sich in einem Suchbaum abbilden. Der Suchbaum repräsentiert alle möglichen Berechnungspfade eines Ziels G eines Programms P. Das Ziel G befindet sich an der Wurzel des Suchbaumes. Die Knoten des Baumes sind Ziele, die leere Klausel ?-. oder FAIL. Mit FAIL wird die fehlgeschlagene Berechnung repräsentiert. Nach der Reihenfolge der Regelauswahl von PROLOG werden die Pfade im Suchbaum von links nach rechts geordnet (siehe auch [43]).
Für das Durchsuchen dieses Suchbaums existieren unter anderem zwei gegensätzliche Strategien: die Breitensuche und die Tiefensuche. Bei der Breitensuche [43] wird der Suchbaum Ebene für Ebene durchsucht. Hiermit wird die Vollständigkeit der SLD-Resolution gesichert, weil sich die Suche nicht in unendlichen Zweigen verfängt. Die Breitensuche ist sehr aufwendig, weil alle bisher noch nicht abgeleiteten Ziele mit ihren Substitutionen sich gemerkt werden müssen. In der Tiefensuche wird das vermieden. Der Auswertungsmechanismus von Prolog (nach [43]) verfolgt die Strategie der Tiefensuche, wenn er immer den am weitesten links liegenden, noch nicht besuchten Pfad bearbeitet. Kommt der Suchprozess bei seiner Abarbeitung in eine Sackgasse (FAIL), wird ein Schritt zurückgegangen (nach oben) und ein alternativer Weg weiterverfolgt. Dieses Vorgehen nennt man backtracking. Bei dieser Vorgehensweise kann es passieren, dass ein unendlicher Pfad verfolgt wird, obwohl eine endliche Abarbeitungsfolge existiert. Deshalb kann die Vollständigkeit der SLD-Resolution mit dieser Auswertungsstrategie von PROLOG nicht gesichert werden.

3.1.2 Erweiterung logischer Sprachen um Constraints

Die constraint-logische Programmierung entstand durch das Hinzufügen von Constraints zu der logischen Sprachen. Hierbei werden zusätzlich zu den Aufrufen von Prädikaten auf den rechten Seiten der Klauseln und im auszuwertenden Ziel als Constraints bezeichnete Bedingungen, die speziell behandelt werden, zugelassen. Außerdem wird in den PROLOG-Auswertungsalgorithmus ein entsprechender Constraint-Lösungsalgorithmus eingebettet. Die logischen und constraint-logischen Programme unterscheiden sich in der Verarbeitung, der in den rechten Seiten der Klauseln und in den Zielen neben Prädikaten vorkommenden Constraints (siehe auch [43]).
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 instanziiert (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.
Die Suchbäume bei constraint-logischen Programmen sind im Vergleich zu Suchbäumen von logischen Programmen im Allgemeinen kleiner und enthalten meistens weniger Knoten. Somit erfordert die Auswertung eines constraint-logischen Programms weniger Berechnungsschritte und weniger Arbeitsspeicher. Die Constraints können unvollständige Informationen enthalten und die Variablen brauchen bei der Auswertung nicht gebunden zu sein. Die Ursache für diesen Effizienzgewinn liegt in der Nutzung von Constraint Solving in den constraint-logischen Sprachen anstatt von Unifikation in logischen Sprachen [43].

3.2 Constraintsysteme und Constraintlöser

Unter dem Begriff Constraint versteht man im mathematischen Sinne Rand- bzw. Nebenbedingung. Zum Beispiel (siehe auch [43]) ist das Ohmsche Gesetz U = R * I eine Randbedingung die bei der Berechnung von elektrischen Schaltungen zu beachten ist.
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.

Boolsches Constraintsystem:
Das Constraintsystem CS_{bool} wird für Probleme eingesetzt, die sich als aussagenlogische Erfüllbarkeitsprobleme (SAT-Probleme) formulieren lassen. Diese Probleme werden mit boolschen 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 boolschen Gleichungen erfüllt (nach [43]).

Definition: Das boolsche Constraintsystem:

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

- $C_{bool} \subseteq \{0, 1, \neg, \wedge, \vee, =\}$,
- $D_{bool} = \{0,1\}$,
- $X_{bool} \subseteq Var_{bool}$ den boolschen Variablen

basiert auf der boolschen 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 boolschen Constraints.
Mit dem boolschen Constraintsystem lässt sich zum Beispiel das Zweifarbenproblem behandeln (siehe

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 $\in X_{bool}$ und durch die Constraints wie folgt formuliert.

$$((\mathrm{x} \wedge \neg \mathrm{y}) \vee (\neg \mathrm{x} \wedge \mathrm{y})) \wedge ((\mathrm{x} \wedge \mathrm{z}) \vee (\neg \mathrm{x} \wedge \neg\ \mathrm{z})) \wedge ((\mathrm{y} \wedge \neg \mathrm{z}) \vee (\neg \mathrm{y} \wedge \mathrm{z})) = 1.$$

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:

$$CS_{fd} = (C_{fd}, D_{fd}, X_{fd}) \text{ mit}$$

- $C_{fd} \subseteq \{0, 1, 1,-1,..., +, -, *, /, =, \neq, >,<, \geq, \leq\}$,
- $D_{fd} = Z$,
- $X_{fd} \subseteq Var_{fd}$ 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 einzulasten. 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 Lösen n-dimensionaler Ressourcenprobleme vorgestellt, die Inhalt dieser Arbeit sind.

3.2.1 Lösungen und Erfüllbarkeit

Bei der Menge von gegebenen Constraints, die im allgemeinen ein Constraintnetz bilden, ist es interessant zu wissen, ob die Konjunktionen von Constraints erfüllbar sind. Bei der Ermittlung der Erfüllbarkeit wird nach der Belegung von Variablen mit zulässigen Werten gesucht. Die Belegung der Variablen mit Werten wird auch als Lösung bezeichnet. Die Prüfung auf Erfüllbarkeit und die Prüfung der Lösbarkeit von Constraints ist somit gleichbedeutend, denn ist eine Lösung bekannt, so weiß man auch, dass die Constraints erfüllbar sind. Umgekehrt gilt: wenn alle Constraints gleichzeitig erfüllbar sind, gibt es mindestens eine Lösung (nach [43]).

Erfüllbarkeit
Definition Erfüllbarkeit (nach [43]):
Gegeben sei ein Constraintsystem $CS = (C,D,X)$, dann ist eine Konjunktion $\bigwedge_{j\in\{1,...,n\}} c_j$ von Constraints $c_1,...,c_n \in C$

- erfüllbar / konsistent in D, wenn $D \models \exists \bigwedge_{j\in\{1,...,n\}} c_j$ gilt,

- nicht erfüllbar / inkonsistent in D, wenn $D \models \neg\exists \bigwedge_{j\in\{1,...,n\}} c_j$ gilt.

Lösungen
Definition Lösungen (nach [43]):
Gegeben sei ein Constraintsystem $CS = (C,D,X)$ und eine Belegung σ: $X \longmapsto D$, mit der Abbildung der Variablen X auf die Domänen in D. Dann ist die Belegung σ eine Lösung einer Konjunktion $\bigwedge_{j\in\{1,...,n\}} c_j$ von Constraints $c_1,...,c_n \in C$, wenn $(D, \sigma) \models (\bigwedge_{j\in\{1,...,n\}} c_j^{'})\ \sigma$ gilt.

Zur Lösung von Constraints sind Verfahren zur Entscheidung der Erfüllbarkeit von Constraints notwendig. Die Algorithmen zur Entscheidung der Erfüllbarkeit haben als Nebeneffekt, dass sie auch zur Bestimmung von Lösungen verwendbar sind. Eine Vorgehensweise zur Entscheidung der Erfüllbarkeit wäre, alle möglichen Belegungen der Variablen mit Werten durchzuprobieren. Wenn die Zahl der Belegungen sehr groß oder unendlich ist, wird diese Vorgehensweise praktisch nicht umsetzbar. Zum Beispiel bei folgendem Problem:

$$(x * y > 3) \wedge (x + 1 < y) \wedge (x,y \in [1,10])$$

wären einhundert Belegungen ($x \longmapsto 1$, $y \longmapsto 1$), ($x \longmapsto 2$, $y \longmapsto 1$), ..., ($x \longmapsto 10$, $y \longmapsto 10$) 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_{fd}) ein und zwar jeweils genau auf einen Wert.
Bei den linearen Constraints kann die Erfüllbarkeit und die Lösung durch geometrische Verfahren bestimmt werden. Es wird dazu das Verfahren der Überschneidung von Hyperebenen und der dadurch entstehenden Halbräume verwendet. Dieses soll an folgendem Beispiel erläutert werden.

Beispiel:
Gegeben sei die Konjunktion linearer Ungleichungen ($y \leq 5x) \wedge (x \leq 5y$). 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: $D \models \exists x\ \exists y: (y \leq 5x \wedge x \leq 5y)$. 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.

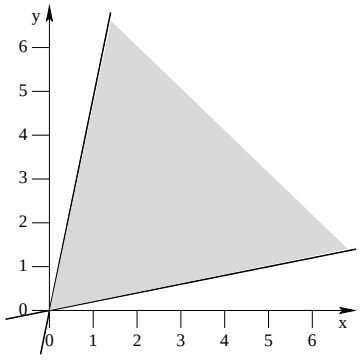

Abbildung 3.1: Geometrische Interpretation von $y \leq 5x \wedge x \leq 5y$

Die Nichterfüllbarkeit der Konjunktion (5y < x) ∧ (5x < y) in D lässt sich durch den Schnitt der zwei Geraden 5y = x und 5x = y und durch die zu suchende Überlappung 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 ∧ 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.

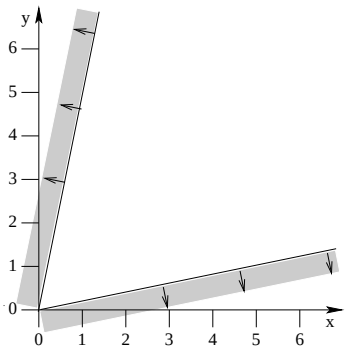

Abbildung 3.2: Geometrische Interpretation von 5y < x ∧ 5x < y

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 $\triangle C = \{c_1 \wedge ... \wedge c_m \mid$ m ∈ N und c_1, ..., $c_m \in C\}$ und $\delta C = \{C_1 \vee ... \vee C_l \mid$ l ∈ N und C_1, ..., $C_l \in \triangle C\}$:

Erfüllbarkeit (Satisfiability / Consistency) (nach [43]):
Gegeben ist eine beliebige Konjunktion von Constraints $C \in \triangle C$. Mit dem Test solv:$\triangle C \rightarrow$ {true, false, unbekannt} soll versucht werden zu entscheiden, ob die Konjunktion von Constraints C in der Domäne D erfüllbar ist oder nicht.

Der Test solv(C) muss korrekt sein. Wenn solv(C) = true ist, dann muss C in D erfüllbar sein und wenn solv(C) = false ist, dann muss C in D nicht erfüllbar sein. Der Test ist vollständig, wenn er für alle C ∈ $\triangle C$ deren Erfüllbarkeit in der Domän D entscheidet und es gilt solv(C) ∈ {true, false}, das heißt, 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 $C \in \triangle C$ und eine beliebige Disjunktion von Konjunktionen von Constraints $CD \in \delta C$ in D. Mit dem Test entail: $\triangle C \times \delta C \rightarrow$ {true, false, unknown}, mit der Transformation T zur Umwandlung einer Konjunktion von Constraints $\triangle C$ in eine Disjunktion von Constraints δC, soll versucht werden zu entscheiden, ob aus der Konjunktion von Constraints C eine Disjunktion von Constraints CD in der Domän D folgt oder nicht.

Der Test entail(C, CD) muss korrekt sein. Wenn entail(C, CD) = true ist, dann muss $D \models \forall\ (C \rightarrow CD)$ gelten und wenn entail(C, CD) = false ist, dann muss $D \models \neg\forall\ (C \rightarrow CD)$ gelten. Der Test ist vollständig, wenn er für alle $C \in \triangle C$ und alle $CD \in \delta C$ die Folgerbarkeit in D entscheidet und es gilt entail(C, CD) ∈ {true, false}, das heißt, es kommt kein {unknown} vor. Beim Vorkommen des

Ergebnisses {unknown} ist der Test unvollständig.

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 `proj`: $\triangle C \times P(X) \rightarrow \triangle C$ sollen in einer gegebenen beliebigen Konjunktion von Constraints $C \in \triangle C$ Variablen eliminiert werden. Das Verfahren `proj` berechnet dann eine Konjunktion von Constraints $\texttt{proj}(C, x^*) \in \triangle C$, die äquivalent zu $\exists_{-x^*} C$ ist und es gilt: $D \models \forall\ (\exists_{-x^*} C \leftrightarrow \texttt{proj}(C, x^*))$ und für die $x^* \subseteq \text{var}(\texttt{proj}(C, x^*)) \subseteq \text{var}(C)$.

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 Raums, der durch die Variablen x^* aufgespannt wird, ist dann gleichbedeutend mit der Elimination der Variablen, die nicht in x^* enthalten sind (Abbildung 3.3).

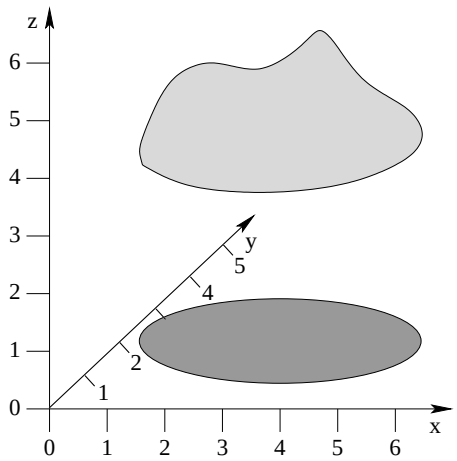

Abbildung 3.3: Geometrische Projektion

Determinationsdetektion: (nach [43]):
Gegeben ist eine beliebige Konjunktion von Constraints $C \in \triangle C$ und beliebige Variablen $\text{x} \in X^s$ einer beliebigen Sorte s. Mit dem Test `det`: $\triangle C \times X \rightarrow C$ 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 $\text{x}=\sum^s \text{z}$ aus C folgt oder nicht. Somit gilt $\texttt{det}(C, \text{x}) = (\text{x}=\sum^s \text{z})$, wenn $D \models \forall(C \rightarrow \text{x}=\sum^s \text{z})$ für eine Konstante z ist oder es gilt $\texttt{det}(C, \text{x}) = (\text{x}=\sum^s \text{x})$, wenn $D \models \neg\forall(C \rightarrow \text{x}=\sum^s \text{z})$ für alle Konstanten z ist.

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

Simplifikation: (nach [43]):
Gegeben ist eine beliebige Konjunktion von Constraints $C \in \triangle C$. Das Verfahren `simpl`: $\triangle C \rightarrow \triangle C$ vereinfacht eine beliebige Konjunktion von Constraints $C \in \triangle C$. Wenn $\texttt{simpl}(C) = C'$ ist, gilt $D \models C \leftrightarrow C'$ 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 $\texttt{simpl}(C) = \text{false}$, wenn $\text{solv}(C) = \text{false}$ ist und es gilt $\texttt{simpl}(C) = \text{true}$, wenn $\text{entail}(C, \text{true}) = \text{true}$ ist.

Zusätzlich zur Vollständigkeit oder Unvollständigkeit und der Korrektheit sollte ein Constraintlöser auch die Eigenschaft Wohlverhalten aufweisen. Unter Wohlverhalten eines Constraintlösers versteht man, dass seine Lösungen sowohl unabhängig von der Reihenfolge und der Mehrfachnennung von Constraints als auch unabhängig von der Benennung der Variablen ist sowie Monotonie aufweist.

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

- mengenbasiert ist; das heißt, es gilt solv(C) = solv(C'), wenn $C = c_1 \wedge ... \wedge c_n$, $C' = c_1' \wedge ... \wedge c_m'$ und $\{c_1, ..., c_n\} = \{c_1', ..., c_m'\}$ gilt.
- monoton ist; das heißt, falls solv(C) = false ist, dann auch solv($C \wedge C'$) = false.
- unabhängig von Variablenbenennungen ist; das heißt, solv(C) = solv(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 nun Probleme betrachtet, bei denen die an der Problemdefinition 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 $\{e_1,...,e_k\}$ gibt, wobei $e_1,...,e_k$ Konstanten einer bestimmten Sorte s sind, die definiert ist: für alle x gilt $x \in (\ C_E\ \{e_1,...,e_k\}^{fd})$ genau dann, wenn $x \in \{\ e_1^{fd},...,e_k^{fd}\}$ ist. Es wird angenommen, dass es zu jedem Element der Trägermengen der Struktur ein entsprechendes Konstantensymbol gibt. Somit gibt es für jede Sorte s und jedes Element e $\in D_{fd}^s$ ein Konstantensymbol c, so dass c^{fd} = e ist und ein beliebiges Constraint C_E [m,n] kann auch als Constraint C_E {m,...,n} betrachtet werden.

Definition **Constraint-Satisfaction-Problem (CSP)** (nach [43]):
Gegeben sei eine Konjunktion von Constraints über endlichen Domänen $(C \wedge x_1 \in D(x_1) \wedge ... \wedge x_n \in D(x_n)) \in \triangle CS_{fd}$. Diese Konjunktion von Constraints über endlichen Domänen ist ein Constraint-satisfaction-Problem (CSP), wenn

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

3.3.2 Konsistenz

Zur Lösung von Constraint-Satisfaction-Problemen werden Konsistenztechniken verwendet. Mit ihnen wird der Aufwand zur Lösung des kombinatorischen Problems reduziert. Die Konsistenztechniken versuchen, die Werte aus den Wertebereichen der Variablen zu entfernen, die nicht Teil einer Lösung sein können. Es handelt sich hierbei um Werte, die auf grund der Constraints nicht in der Lösung vorkommen dürfen. Es gehen durch die Anwendung der Konsistenztechniken keine potentiell in Frage kommenden Lösungen verloren.

Knoten-Konsistenz

Es gibt verschiedene Konsistenzenarten 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 $C \wedge x_1 \in D(x_1) \wedge ... \wedge x_n \in D(x_n)$ mit $C = c_1 \wedge ... \wedge c_k$. Eine Variable ist knoten-konsistent (node consistent) wenn:

- vars(c) nicht einelementig ist ($|\text{vars(c)}| \neq 1$) oder
- vars(c) = $\{x\}$ ist und für jeden Wert d $\in D_{fd}(x)$ eine Belegung mit x $\longmapsto$ d eine Lösung von c ist.

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

Beispiele:

$$(\text{X} > 4) \wedge (\text{Y} < 6) \wedge (\text{X} + \text{Y} = \text{Z}) \wedge (\text{X} \in [0,10]) \wedge (\text{Y} \in [0,10]) \wedge (\text{Z} \in [0,10])$$

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.

$$(\text{X} > 4) \wedge (\text{Y} < 6) \wedge (\text{X} + \text{Y} = \text{Z}) \wedge (\text{X} \in [5,10]) \wedge (\text{Y} \in [0,5]) \wedge (\text{Z} \in [0,10])$$

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

Die Verfahren zu Prüfung der Konten-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 (zweistellige) Constraint, einzeln betrachtet, erfüllt ist.

Definition **Kanten-Konsistenz** (nach [43]):
Gegeben sei ein Constraint Satisfaction Problem CSP $C \wedge x_1 \in D(x_1) \wedge ... \wedge x_n \in D(x_n)$ mit $C = c_1 \wedge ... \wedge c_k$. Ein Constraint c in der Konjunktion C ist kanten-konsistent (arc consistent) wenn:

- vars(c) nicht zweielementig ist ($|\text{vars(c)}| \neq 2$) 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 $\longmapsto$ d, y $\longmapsto$ e eine Lösung von c ist.

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

Beispiel:

$$(\text{X} > 4) \wedge (\text{Y} < 6) \wedge (\text{X} + \text{Y} = \text{Z}) \wedge (\text{X} \in [5{,}10]) \wedge (\text{Y} \in [0{,}5]) \wedge (\text{Z} \in [0{,}10])$$

Dieses Beispiel ist knoten- und kanten-konsistent, obwohl die Belegung für die Variable Z $\longmapsto$ 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 $O(c^x d^y)$ [43] mit c der Anzahl der zweistelligen (binären) Constraints (x dem konkretem Aufwand) und d der Größe der größten Domäne (y dem konkretem 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 einelementig 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

In Constraint-Satisfaction-Problemen kommen nicht nur ein- oder zweistellige (unäre oder binäre) Constraints vor sondern meist auch mehrstellige Constraints. Die Konsistenzbildung über mehrstelligen Constraints ist die Verallgemeinerung der Knoten- und Kanten-Konsistenz.

Definition **lokale Konsistenz** (nach [43]):
Gegeben sei ein Constraint-Satisfaction-Problem CSP $C \wedge x_1 \in D(x_1) \wedge ... \wedge x_n \in D(x_n)$ mit $C = c_1 \wedge ... \wedge c_k$. Ein Constraint c in der Konjunktion C ist lokal konsistent (local/hyper-arc consistent) wenn vars(c) = $\{x_1, ..., x_m\}$ ist und es gibt für jedes i $\in \{1, ..., m\}$ und zu jedem Wert $d_i \in D_{fd}(x_i)$ für alle j $\in \{1, ..., m\} \setminus \{i\}$ Werte $d_j \in D_{fd}(x_j)$, so dass eine Belegung mit $x_1 \longmapsto d_1$, ..., $x_m \longmapsto d_m$ eine Lösung von c ist.
Ein gegebenes CSP ist lokal konsistent, wenn die Constraints c_1,...c_k local konsistent sind.

Beispiel:

$$(\text{X} \neq \text{Y}) \wedge (\text{Y} \neq \text{Z}) \wedge (\text{Z} \neq \text{X}) \wedge (\text{X} \in [0{,}1]) \wedge (\text{Y} \in [0{,}1]) \wedge (\text{Z} \in [0{,}1])$$

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 den Verfahren der lokale Konsistenz Grenzen bei dem Lösen von CSP's gesetzt. Die notwendige stärkere Einschränkung der Domänen und damit

die Erkennung eines Widerspruchs lässt sich nur durch stärkere Konsistenzeigenschaften, globalere Betrachtung des Problems oder durch die Zusammenfassung einfacher Constraints ($\neq$) zu globaleren Constraints erreichen. Eine solche Zusammenfassung für die einfachen $\neq$ - Constraints stellt zum Beispiel das Constraint alldifferent(X,Y,Z) [24] dar. Bei diesem alldifferent-Constraint müssen die drei Variablen X, Y, Z voneinander verschiedene Werte annehmen. Dafür ist es aber notwendig, dass die Vereinigung der Domänen der Variablen mindestens drei verschiedene Werte enthält. Ansonsten wird durch das globale alldifferent-Constraint ein Widerspruch und die Inkonsistenz erkannt.
Verfahren zur Herstellung von lokalen 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:

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

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 $\wedge$ $x_1 \in [\min(x_1), \max(x_1)] \wedge ... \wedge x_n \in [\min(x_n), \max(x_n)]$, wobei $C = c_1 \wedge ... \wedge c_k$ sei. Ein Constraint c in der Konjunktion C ist grenzen-konsistent (bounds consistent), wenn var(c) = $\{x_1, ..., x_m\}$ ist, und es für jedes i $\in \{1, ..., m\}$ und jeden Wert $d_i \in \{\min(x_i)^{fd}, \max(x_i)^{fd}\}$ für alle j $\in \{1, ..., m\} \setminus \{i\}$ Werte $d_j \in [min_j^{fd}, max_j^{fd}] \subset R$ gibt, so dass eine Belegung mit $x_1 \longmapsto d_1$, ..., $x_m \longmapsto d_m$ 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 $X^2 = 2 \wedge$ X $\in [1, 2]$ ist nicht knoten-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 $\wedge$ D mit S $\equiv$ 12 * X + 107 * Y - 17 * Z + 5 * T = 2 und D $\equiv$ X $\in$ [1,1000] $\wedge$ Y $\in$ [1,1000] $\wedge$ Z $\in$ [1,1000] $\wedge$ 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 Intervallarithmetik erfolgen. Die Intervallarithmetik schränkt die Intervallgrenzen und damit die Wertebereiche der Variablen ein. Hierzu gelte: für alle beteiligten Variablen $x_i \in [x_i^{min}, x_i^{max}]$.

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

x + y > 20 ∧ x + 5 < y ∧ x ∈ [1, 15] ∧ y ∈ [1, 15]
wird durch die Betrachtung der Ungleichung x + 5 < y zu:
x + y > 20 ∧ x + 5 < y ∧ x ∈ [1, 9] ∧ y ∈ [7, 15]

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

x + y > 20 ∧ x + 5 < y ∧ x ∈ [1, 9] ∧ y ∈ [7, 15]
wird durch die Betrachtung der Ungleichung x + y > 20 zu:
x + y > 20 ∧ x + 5 < y ∧ x ∈ [6, 9] ∧ y ∈ [12, 15]

In jeweiliger einzelner Betrachtung der zwei Ungleichungen sind weitere Einschränkungen nicht erreichbar. Mit diesem einfachen Verfahren werden die Möglichkeiten der Wertebelegungen ohne Verlust von Lösungen stark eingeschränkt. Von 15 * 15 = 225 Wertebelegungen der Variablen müssen nur noch vier Wertebelegungen je Variable also 16 Wertebelegungen der Variablen betrachtet werden.
Aus dieser Einschränkung ist noch nicht erkennbar, dass nicht alle 16 Wertebelegungen eine Lösung des CSPs sind. Zum Beispiel ist die Belegung x ⟼ 7 und y ⟼ 12 keine Lösung und die Belegungen (x ⟼ 8, y ⟼ 14), (x ⟼ 8, y ⟼ 15) und (x ⟼ 9, y ⟼ 15) sind Beispiellösungen.

Wenn man annimmt, dass es zu jedem FD-Constraint (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: $C_{fd} \models C \leftrightarrow$ bounds-consistency(C) und bounds-consistency(C) ist grenzen-konsistent für jedes beliebiges CSP C, dessen Domänen der Variablen durch Intervalle bestimmt sind.

Das Verfahren 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 bounds-consistency() 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 / Domänreduzierung)

Der Suchraum für die Lösung ist i. allg. so groß, dass er in einer sinnvollen Zeitspanne nicht vollständig durchsucht werden kann. Die Suche wird deshalb durch die Anzahl der erlaubten Backtrackingschritte begrenzt. Ist diese Anzahl überschritten, wird die Suche abgebrochen und gegebenenfalls zu einer anderen Suchstrategie gewechselt. Die Lösungssuche erfolgt meistens unter Anwendung unterschiedlicher Suchstrategien in Verbindung mit Labeling-Prozeduren und Backtracking.

Labeling

Durch die Lösungssuche werden den Constraintvariablen schrittweise Werte aus ihrem Domänenbereich zugeordnet. Dazu wird entsprechend der Suchstrategie zunächst einer Variablen ein Wert aus dem Domänenbereich zugeordnet. Es werden die Konsequenzen aus dieser Zuordnung entsprechend des verwendeten lokalen Konsistenzverfahrens berechnet. Ergibt sich ein Widerspruch, wird Backtracking durchgeführt und ein anderer Wert einer bereits betrachteten Constraintvariablen festgelegt. Diese Prozedur wird als Labeling bezeichnet. Die Suchstrategie legt fest, in welcher Reihenfolge den Variablen Werte zugeordnet werden und in welcher Reihenfolge diese Werte aus dem Domänenbereich der Variable verwendet werden.

Domänreduzierung

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)

Die Verfahren zur Ermittlung der lokalen und Grenzen-Konsistenz ermöglichen die Realisierung von unvollständigen und vollständigen Constraintlösern. Die vollständigen Constraintlöser lassen sich durch die Integration der Suche mit Rücksetzen (Backtracking) in den unvollständigen Constraintlöser realisieren.
Beim Backtracking werden die Variablen nacheinander mit Werten belegt. Nach jeder Belegung der Variablen mit einem Wert wird die Konsistenz der gegebenen Constraints überprüft. Liegt eine Inkonsistenz vor oder wurde ein Wertebereich einer Variablen leer - der Variablen kann kein Wert mehr zugewiesen werden -, werden die Belegungen der Variablen nacheinander wieder rückgängig gemacht. Es wir dabei ein Schritt zurückgegangen (Backtracking) und die in jeden Belegungschritt letzte Belegung der Variablen aufgehoben und eine alternative Belegung der Variable gewählt.

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

$$c_1 \wedge ... \wedge c_k \wedge x_1 \in \mathrm{D}(x_1) \wedge ... \wedge x_n \in \mathrm{D}(x_n),$$

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

$$\mathrm{backtrack}(C) = c_1 \wedge ... \wedge c_k \wedge x_{i_1} = e_{i_1} \wedge ... \wedge x_{i_j} = e_{i_j} \wedge x_1 \in \{e_1\} \wedge ... \wedge x_n \in \{e_n\}\ ,$$

wobei $x_{i_1} = e_{i_1}$, ..., $x_{i_j} = e_{i_j}$ Zuweisungen und e_1, ..., e_n Konstanten sind, wenn C konsistent ist. Des weiteren ist bei Konsistenz von C jede Substitution mit $x_1 \longmapsto e_1^{fd}$, ..., $x_n \longmapsto e_n^{fd}$ eine Lösung des CSP C. Somit ist ein Verfahren $solv_{bt}$

$$solv_{bt}(C) = \text{true, falls backtrack}(C) \neq \text{false ist oder}$$
$$solv_{bt}(C) = \text{false, falls backtrack}(C) = \text{false ist,}$$

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 backtrack(C) angewendet.

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

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

$$C \equiv \mathrm{x} + \mathrm{y} > 20 \wedge \mathrm{x} + 5 < \mathrm{y} \wedge \mathrm{x} \in [6, 9] \wedge \mathrm{y} \in [12, 15].$$

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

$$C_1 \equiv \mathrm{x} + \mathrm{y} > 20 \wedge \mathrm{x} + 5 < \mathrm{y} \wedge \mathrm{x} \in [6, 6] \wedge \mathrm{y} \in [15, 15].$$

Eine Lösung für die Ungleichungen $\mathrm{x} + \mathrm{y} > 20 \wedge \mathrm{x} + 5 < \mathrm{y}$ ist $\mathrm{x} \longmapsto 6$, $\mathrm{y} \longmapsto 15$.

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

$$C_2 \equiv \mathrm{x} + \mathrm{y} > 20 \wedge \mathrm{x} + 5 < \mathrm{y} \wedge \mathrm{x} \in [7, 7] \wedge \mathrm{y} \in [14, 14].$$

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

3.4 Propagationstechniken

Propagation

Die grundlegende Strategie der constraint-logischen Programmierung besteht darin, Widersprüche im Constraintnetz zu erkennen. Klassische Lösungsverfahren überprüfen erst am Ende der Lösungssuche, ob die ermittelten Ergebnisse mit den einschränkenden Randbedingungen übereinstimmen. Die dabei festgestellten Widersprüche werden anschließend gelöst. Bei Verwendung der Constraint-Technik werden schon vor Beginn der Lösungssuche die Wertebereiche entsprechend den Randbedingungen (Constraints) eingeschränkt, so dass die Widersprüche früher erkannt werden. Damit verbunden ist eine Reduktion der notwendigen Berechnungen für die Lösungsermittlung. Dieser Vorgang wird als Propagation bezeichnet. Er soll anhand eines Beispiels erläutert werden.
Als Beispiel wird die Platzierung von drei Blöcken in einer Reihe verwendet. Es werden Ungleichungen der Form $Position_i + BreitedesBlocks_i < Position_j$ 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.

$B_1 = B_2 = B_3 = 1 \,..\, 20$

Constraint: $B_2 + 5 < B_1$
Propagation der Variablen B_1 und B_2

$B_1 = \mathbf{7}$ .. 20, $B_2 = 1$ .. $\mathbf{14}$, $B_3 = 1$ .. 20

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

$B_1 = \mathbf{18}$ .. 20, $B_2 = \mathbf{12}$ .. 14, $B_3 = 1$ .. $\mathbf{3}$

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

$B_1 = \mathbf{18}$, $B_2 = \mathbf{12}$, $B_3 = \mathbf{1}$

Der wesentliche Vorteil der constraint-basierten Verfahren ist, dass einerseits ein zulässiger Wert, der einer Variablen zugeordnet wird, zu einer Suchraumeinschränkung führt, und andererseits, ein unzulässiger Wert einer Variablen erkannt und zurückgewiesen wird. Dieses Verhalten wird erreicht, indem, anders als in der klassischen Programmierung, Gleichheits- oder Ungleichheits-Bedingungen weit vorn im Programm (und Programmablauf) stehen, auch wenn sie noch nicht berechenbar sind (weil Variable noch keinen Wert haben).
Für die Repräsentation komplexer Zusammenhänge hat sich die Integration verschiedenartiger Methoden zu einem Verfahren der Constraintbehandlung als sehr vorteilhaft erwiesen. Dies erfolgt auf hohem Abstraktionsniveau durch die Angabe von relationalen Beziehungen zwischen Objekten, die Variablen enthalten können. Derartige Konstrukte werden als globale Constraints bezeichnet. Solche globalen Constraints werden auch zur Sicherung der Überlappungsfreiheit verwendet Die Aufbereitung der globalen Constraints erfolgt im Prozess der Vorverarbeitung, bei dem die benötigten Constraints ermittelt werden.

Forward checking und Look ahead

Neben dem Backtracking und den Konsistenztechniken, die zwei gegensätzliche Möglichkeiten zum Lösen von Constraint-Satisfaction-Probleme (CSP) darstellen, gibt es Mischformen davon. Dabei wird ein Konsistenzalgorithmus in einen Backtracking-Algorithmus integriert.

Forward checking

Mit Forward checking werden die Constraints zwischen der aktuellen Variable und den Variablen, die noch keinen Wert zugeordnet bekommen haben, überprüft.
Die constraint-logische Programmiersprache der französischen Firma COSYTEC macht bei der Propagation nur eine Minimum- und eine Maximum-Einengung der Domänen. Um aber Lücken in der Domän auffinden zu können, muss man einen Forwardcheck machen. Der Forwardcheck fragt alle Werte der Domän unter Beachtung des Constraintnetzes (Aktivierung des Solvers) ab und schlägt bei nicht vorhanden Werten in der Domän fehl. P. Van Hentenryck [40] [41] hat Forward checking und das nachfolgend erwähnte Look ahead detailliert untersucht und beschrieben.

Look ahead

Von Look ahead gibt es zwei Ausprägungen: teilweiser Look ahead (PLA) und vollständiger Look ahead (LA). Der vollständige Look ahead wird auch Maintaining Arc Consistency (MAC) genannt. Mit Look ahead werden die Domänen soweit reduziert, dass mögliche zukünftige Konflikte ausgeschlossen werden, und der Suchbaum wird dadurch stärker und früher beschnitten als bei Forward checking. Das kann dazu führen, dass eine Teillösung mit wesentlich mehr Aufwand erzeugt wird als beim Forward checking.
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 den vorhergehenden Variablen (Knoten) getestet. Beim

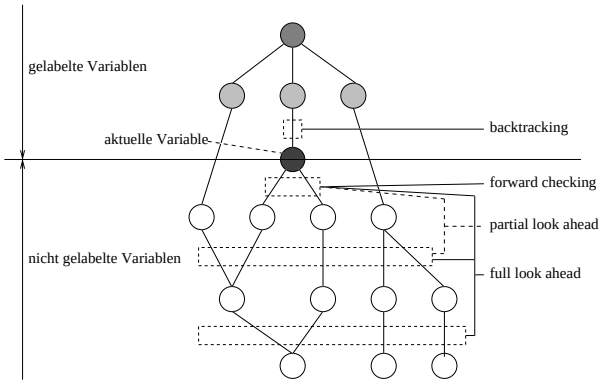

Abbildung 3.4: Übersichtsbild (Backtracking, Forward checking, Look ahead)

Forward checking werden die Constraints getestet, die mit der aktuellen Variable und den noch nicht instanziierten Variablen in Beziehung stehen. Beim Look ahead werden alle noch nicht instanziierten Variablen, die mit Constraints verbunden sind, getestet. Wenn wenige Variablen einen Wert zugewiesen bekommen haben, kann der Look ahead-Test sehr aufwendig sein, da viele Constraints getestet werden müssen. Die stärkere Propagation des Look ahead-Tests hat zur Folge, dass der Suchraum stärker eingeschränkt ist, hierfür die Gesamtkosten aber hoch sind. Look ahead wird kaum genutzt, da es häufig ineffizienter ist als Forward checking und Backtracking (″Simple Backtracking″) [6].

3.5 Heuristische Suche

In einem Constraintnetz lassen sich durch heuristische Suche Inkonsistenzen 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 nächte Variable zur Wertebelegung nach dem Zustand des Suchprozesses dynamisch bestimmt.

Die dynamische Festlegung der Reihefolge 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 problemspezifische Auswahlstrategien (Heuristiken) wie die Engpassanalyse 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

denen ein Fehlschlag am wahrscheinlichsten ist. Im Constraintnetz wird hierbei die Variable mit den wenigsten Alternativen und damit der kleinsten Domäne ausgewählt. Bei dieser Auswahlstrategie sollte die nächste Variable dynamisch bestimmt werden, da die Größe der Domänen mit der Anzahl der aktiven Constraints und deren Propagation, der Herstellung der Konsistenz in den verschiedenen Zweigen des Suchbaums schwankt. Dieses Auswahlverfahren setzt auf die Annahme, dass in größeren Domänen die Wahrscheinlichkeit höher ist, dass in der verbleibenden Domäne noch eine Lösung gefunden werden kann. Mit diesem Prinzip wird der Suchbaum von Anfang an klein gehalten, was zur schnelleren Erkennung von unlösbaren Problemen und erfolglosen Zweigen im Suchbaum führt.

Für die statische Festlegung der Reihenfolge der Variablen oder in Kombination mit dem "first-fail"-Prinzip, wenn mehrere Variablen mit gleich großen Domänen zur Auswahl stehen, wird das **"most-constrained"-Prinzip** angewendet. Bei diesem Prinzip wird die schwierigste Anforderung zuerst bearbeitet, um erfolgreich zu sein. Im Constraintnetz sind hierfür die Variablen zuerst zu belegen, die in den meisten Constraints enthalten sind. Diese Heuristik kann dahingehend verfeinert werden, dass die Variablen zuerst belegt werden, die zusammen mit den belegten Variablen in den meisten Constraints vorkommen. Hierbei ist eine entsprechende Reihenfolge der zu belegenden Variablen in der Menge der betrachtete Variablen statisch festlegbar. Diese Verfeinerung wird zur Bildung von Kolonien in Graphen benutzt, bei denen benachbarte Knoten des Graphen unterschiedlich einzufärben sind.

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 erfolgversprechenste 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 Domäneneinschränkung, 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 **Domänenreduktion**. Bei diesem Prinzip (nach [43]) wird die Werteauswahl solange wie möglich verzögert, um nicht vorzeitig eine definierte Entscheidung zu treffen. Bei der Verzögerung der Werteauswahl wird die jeweilige Variable nicht mit einem Wert belegt, sondern der Wertebereich der Variable mittels Intervallschachtelung beschränkt.

Die Constraint-Satisfaction Probleme(CSP) werden mittels Constraints modelliert. Die Art der Umsetzung der Modellierung der CSP's hat einen entscheidenden Einfluss auf die Effizienz des Lösungsprozesses. Bei der Modellierung sollten deshalb bestimmten Aspekten der Constraint-Programmierung beachtet werden, um die effiziente Nutzung der Systemressourcen zu gewährleisten. Zu diesen Aspekten der Constraint-Programmierung gehört das Aufbrechen von Symmetrien, die Verwendung von redundanten und globalen Constraints.

Aufbrechen von Symmetrien

Die Komplexität eines Constraint-Satisfaction Problems lässt sich durch das Erkennen und Aufbrechen von probleminhärenten Symmetrien erreichen. Bei diesen Symmetrien handelt es sich meist um mehrere Problemparameter gleichen Charakters. Bei diesen Problemparametern, die als Variablen modelliert werden, ist dann eine mehrfache Unterscheidung ineffizient. Die probleminhärenten Symmetrien können vorliegen, wenn die (initialen) Domänen von Variablen identisch sind, die Variablen mit den gleichen Constraints verknüpft sind oder die Variablen permutierbar sind, ohne dass sich die Semantik des Programms durch Variablenumbenennung ändert. In allen diesen Fällen ist es nicht notwendig, alle Lösungen zu betrachten. Die symmetrischen Lösungen sind durch Variablenpermutationen aus anderen Lösungen herleitbar und können durch das Einfügen zusätzlicher Constraints in das Constraintnetz ausgeschlossen werden. Die zusätzlichen Constraints führen dann zum Beispiel eine Ordnungsrelation über den Variablen ein, was in vielen Anwendungsfällen zu einer Reduzierung des Suchraums und zu einer Beschleunigung des Suchprozesses führt (siehe auch [43]).

Beispiel:
Gegeben sein ein CSP $(A + B = S) \wedge (A \in D(A)) \wedge (B \in D(B)) \wedge (S \in D(S))$ 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 Wertebelegung 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 **redundante Constraints (allgemein)** (nach [43]):

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

Die Reduzierung des Suchraumes durch die Anwendung zusätzlicher redundanter Constraint führt nicht immer zu dem gewünschten Ergebnis. Bei der Hinzufügung von Constraints, die weder bei der Herstellung der lokalen Konsistenz noch bei der Herstellung der Grenzen-Konsistenz zu einer Einschränkung der Domänen und des Suchraumes führen, kann kein positiver Effekt zu erreicht werden. Im ungünstigsten Fall wird durch die Hinzufügung von zusätzliche Constraints das Laufzeitverhalten der Suche durch die überflüssige Verarbeitung der zusätzlichen Constraints verschlechtert.

Beispiel (Redundante Constraints mit keiner Reduzierung des Suchraums):

Gegeben sei ein CSP $(A + B = S) \wedge A \in [0,20] \wedge B \in [0,20] \wedge S \in [0,20]$. Diesem CSP werden die redundanten Constraints $A \geq 0 \wedge A \leq 20 \wedge B \geq 0 \wedge B \leq 20 \wedge S \geq 0 \wedge S \leq 20$ 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

Die Anwendung redundanter Constraints in Verbindung mit dem Aufbrechen von Symmetrien kann bei der Lösung des Problems des Pythagoras zu einer spürbaren Reduzierung des Suchraums führen.

Beispiel (Redundante Constraints und Aufbrechen von Symmetrien mit Reduzierung des Suchraums) (siehe auch [43]):

Gegeben sei ein CSP (A * A + B * B = C * C) $\wedge$ A $\in$ [0,100] $\wedge$ B $\in$ [0,100] $\wedge$ C $\in$ [0,100]. Aus der Definition des CSP's folgt, dass A < C und B < C gilt. Dann kann das CSP in folgendes äquivalente CSP (A * A + B * B = C * C) $\wedge$ (A < C) $\wedge$ (B < C) $\wedge$ A $\in$ [0,100] $\wedge$ B $\in$ [0,100] $\wedge$ 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 * A + B * B = C * C $\wedge$ A < C $\wedge$ B < C $\wedge$ **A $\leq$ B** $\wedge$ A $\in$ [0,100] $\wedge$ B $\in$ [0,100] $\wedge$ 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 * B $\geq$ 2 * C hinzugefügt werden. Dieses Constraint ergibt sich, da A $\leq$ B gefordert wird und 2 * B * B $\geq$ C * C ist, folglich ist $\sqrt{2}$ * B $\geq$ C, somit 1.5 * B $\geq$ C. Die Umformung auf ganze Zahlen ergibt dann 3 * B $\geq$ 2 * C. Das CSP wird nun um das redundante Constraint 3 * B $\geq$ 2 * C erweitert:

A * A + B * B = C * C $\wedge$ A < C $\wedge$ B < C $\wedge$ **A $\leq$ B** $\wedge$ **3 * B $\geq$ 2 * C** $\wedge$ A $\in$ [0,100] $\wedge$ B $\in$ [0,100] $\wedge$ C $\in$ [0,100].

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

3.6 Globale Constraints

Bei der lokalen Konsistenz und der Grenzen-Konsistenz werden zum Lösen von CSP's lediglich die unmittelbaren Zusammenhänge zwischen den Variablen und ihren Domänen zur Einschränkung der Domänen und des Suchraums benutzt. Alle weiterreichenden und global wirkenden Zusammenhänge werden nicht betrachtet und genutzt. Die Nutzung übergreifender und globaler Zusammenhänge kann zu einer weiteren Einschränkung des Suchraums führen und den Suchprozess wesentlich beschleunigen. Die Constraints, die globale Zusammenhänge verarbeiten, werden globale Constraints genannt.

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 spezifiziert werden. Für n Variablen sind dann (n * (n-1))/2 Ungleichungen notwendig. Bei 100 Variablen werden (100 * (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 Constraintnetz verarbeiten.

Definition **alldifferent/1**
Gegeben seien n Variablen X_1, ..., X_n die verschiedene Werte annehmen sollen. Das Constraint

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

realisiert, dass gilt:

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

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)

Bei der exklusiven Belegung von Ressourcen (siehe auch [43]) geht es um die Planung / Einlastung von Arbeitsgängen (Tasks) auf Maschinen (Ressourcen). Das Haupteinsatzgebiet der exklusiven Belegung von Ressourcen liegt in der Termin-, Personaleinsatz-, Maschinenbelegungs- und Ressourcenplanung. Für diese genannten Planungsprobleme soll am Beispiel für die Planung von Arbeitsabläufen (Jobs) auf Maschinen das bestehende Ressourcenbelegungsproblem erläutert werden.
Das Ressourcenbelegungsproblem besteht darin, für eine bestimmte Ressource n Arbeitsgänge mit Startzeiten S_1 ..., $S_n \in N$ und den Werten für die Dauer D_1 ..., $D_n \in N$ so zu planen, dass sich jeweils zwei unterschiedliche Arbeitsgänge nicht überlagern. Das bedeutet, dass für zwei Arbeitsgange 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 Problem könnte man als CSP mit Ungleichungen modellieren. Hierfür müssten alle $2^{(n*(n-1))/2}$ 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*(n-1))/2 Variablen der Form $B_{i,j} \in$ [0,1] mit (i $<$ j) und der Formulierung der ursprünglichen Disjunktion als Konjunktion $S_i + D_i \leq S_j + B_{i,j}$ * maxInt $\wedge$ $S_j + D_j \leq S_i$ + (1 - $B_{i,j}$) * maxInt vermeiden. Bei dieser Definition ist ein maxInt $\in N$ hinreichend groß zu wählen. Das Problem der exklusiven Ressourcenbelegung kann dann als CSP der Form:

$$(\bigwedge_{1 \leq i < j \leq n} S_i + D_i \leq S_j + B_{i,j} * \text{maxInt}) \wedge (S_j + D_j \leq S_i + (1 - B_{i,j}) * \text{maxInt}) \\ \wedge B_{i,j} \in [0,1] \wedge \bigwedge_{1 \leq k \leq n} S_k \in N$$

formuliert werden. Die Generierung eines solchen CSP's würde vermutlich einen quadratisch hohen Modellierungsaufwand in Abhängigkeit von der Zahl der Arbeitsgänge erfordern und einen mindest ebenso großen Aufwand bei der Herstellung der lokalen oder Grenzen-Konsistenz mit geringen Einschränkungen der Domänen.

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 überlappungsfreien Ressourcenbelegung mit unterschiedlichen Bezeichnungen und Funktionsumfängen ausgeprägt. Die allgemeinste Form ist das disjunctive/serialize/2- Constraint mit:

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

Mit diesem Constraint können Ressourcen behandelt werden. Ein disjunktives Constraint drückt aus, dass in einer Ressource nur eine Task zu einer Zeit bearbeitet werden kann. Jede Task nutzt eine eigene Ressourceneinheit und das Ressourcenlimit über allen Tasks beträgt eins.

Globales Constraint (cumulative)

Das Constraint cumulative (nach [24]) wird zur Lösung von Planungsproblemen mit begrenzten Ressourcen eingesetzt. Bei diesen Planungsproblem ist eine Anzahl von Tasks mit unterschiedlichen Abarbeitungsdauern gegeben. Die einzelnen Tasks benötigen einen gewissen vorgegeben Anteil von einer Ressource, um ausgeführt zu werden. Diese Ressource ist für die Planungsperiode auf ein maximales Limit begrenzt. Dieses maximale Limit der Ressource darf durch die Einplanung aller Tasks zu keinem Zeitpunkt im Planungsproblem überschritten werden.
Dieses Ressourcenproblem kann mit einer einfachen Form des Constraints cumulative/4 modelliert werden. Hierfür wird für jede der n Tasks eine Domänvariable S_i für die Startzeit, eine Domänvariable D_i für die Dauer der Bearbeitung und eine Domänvariable R_i für die Größe der Ressourcennutzung der begrenzten Ressource eingeführt. Mit einer Domänvariablen $Limit$ im Bereich von 0 bis zum verfügbaren Ressourcenlimits wird das Limit für die Ressourcennutzung der begrenzten Ressource angegeben. Das Constraint cumulative hat folgende Form:

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

Das Constraint cumulative hat folgende mathematische Definition:

$$\forall \text{i} \in [min_{1 \leq j \leq n}(S_j), max_{1 \leq j \leq n}(S_j + D_j)] : \sum_{k:S_k \leq i < S_k + Dk} R_k \leq Limit$$

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_i. Sie hat eine Bearbeitungsdauer D_i und eine Höhe R_i. In dem Profil des Constraints cumulative aller Tasks darf das Ressourcenlimit von keiner Task überschritten werden.
Hier wurde nur die Grundform des Constraints cumulative/4 dargestellt. Dieses Constraint hat noch eine Vielzahl von zusätzlichen Parametern, zu denen die Festlegung des Endzeitpunkts der einzelnen Tasks, der Fläche der Rechtecke, des Endzeitpunkts der Planung aller Tasks oder der Zwischenressourcenlimits gehören. Durch diese Parameter wird das Constraint cumulative/4 flexibler und es können mit dem Constraint cumulative/8 komplexere Bedingungen ausgedrückt 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: $S = [S_1, ... ,S_n]$ die Liste der Startzeiten, $D = [D_1, ... ,D_n]$ die Liste der Werte für die

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

a = min($min_d(S_1)$, ... , $min_d(S_n)$),
b = max($max_d(S_1) + max_d(D_1)$, ... ,$max_d(S_n) + max_d(D_n)$),
E: E_i mit von S_i: $\forall$ i: $1 \leq$ i $\leq$ n, $S_i + D_i = E_i$,
F: F_i mit von S_i: $\forall$ i: $1 \leq$ i $\leq$ n, D_i * $R_i = F_i$,
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, **Ress** = max($\sum R_j$), $\forall$ j: $S_j \leq$ i $\leq S_j + D_j$ - 1
End: $\forall$ i: $1 \leq$ i $\leq$ n, **End** = max($S_i + D_i$),
Über, Mitte: $\forall$ i: a $\leq$ i $\leq$ b, **Über** = $\sum$ (max(0, $\sum R_j$ - **Mitte**)), $\forall$ j: $S_j \leq$ i $\leq S_j + D_j$ - 1,
dann ist **Cum** ein Cumulative-Constraint der Form:

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

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 (Ress) und dem letzten Endzeitpunkt aller Tasks (End) gegeben.

```
bspgc11(S,E,Ress,End):-
    S = [S1,S2,S3], S :: 1...5,
    D = [2,3,2],
    R = [2,1,2],
    E = [E1,E2,E3], E :: 1...5,
    Ress :: 1...3,
    End :: 1...5,
    cumulative(S,D,R,E,_,Ress,End,_),
    min(labeling(S),Ress).
```

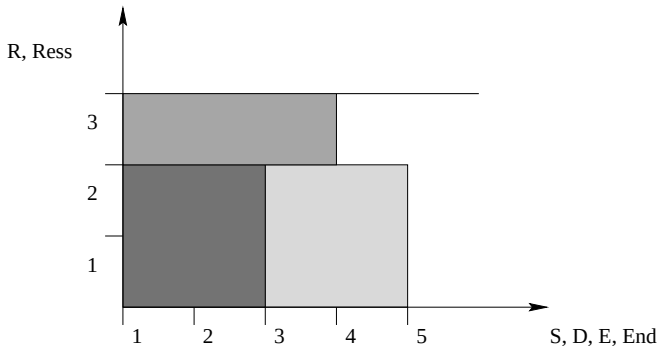

Abbildung 3.5: Beispiel 1 - GC1: Cumulative - Constraint

[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 Ressourcennutzung (Mitte) und die Fläche oberhalb des Mittelwerts der Ressourcennutzung (Über) gegeben.

```
bspgc12(S,E,Ress,End):-
    S = [S1,S2,S3], S :: 1...5,
    D = [2,3,2],
    R = [1,1,1],
    E = [E1,E2,E3], E :: 1...5,
    Ress :: 1...3,
    End :: 1...5,
    Mitte = 2,
    Über :: 1...5,
    cumulative(S,D,R,E,_,Ress,End,[Mitte,Über]),
    min(labeling(S),Über + Ress).
```

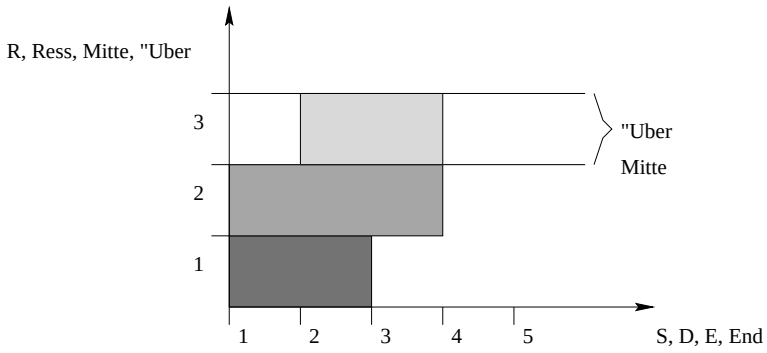

Abbildung 3.6: Beispiel 2 - GC1: Cumulativ - Constraint

Das Constraint cumulative() eignet sich zur Modellierung von Planungsprobleme, zum Beispiel beim 'job-shop scheduling' und bei dem Problem der Einhaltung von Taskreihenfolgen [4]. Das Constraint cumulative() kann auch zur Modellierung von Packungsproblemen in Behältern, Ladeflächen oder Containern benutzt werden. Bei dem Packungsproblem sind Einheiten verschiedener Größe in Räumen mit festem vorgegeben Volumen zu platzieren. Das Problem ist, so viele Einheiten wie möglich auf kleinstem Raum unterzubringen. Es wird das Minimum des benötigten Raums gesucht. Bei diesen Platzierungs- und Planungsproblemen könnten die Räume mit den Ressourcen korrespondieren, die global limitierte Kapazitäten haben (siehe [4]).

Des Weiteren kann das Constraint cumulative() für Produzenten/ Verbraucher-Probleme (producer/-consumer) eingesetzt werden. Diese Consumerressourcen werden eingesetzt bei der Modellierung von Prozessen der Produktion von Rohstoffen und Zwischenprodukten, welche von anderen Prozessen genutzt werden. Das Constraint sichert, dass zu jedem Zeitpunkt von einem Produkt mehr produziert werden muss als verbraucht wird, aber auch die Einhaltung des Mindestvorrats und die Nichtüberfüllung des Lagers.

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 Verschiedenheit von Domänen sichert, in ein globales Constraint umzusetzen, welches die Überlappungsfreiheit 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.

$$(Y_j + H_j \leq Y_i) \vee (Y_i + H_i \leq Y_j) \vee (X_i + L_i \leq X_j) \vee (X_j + L_j \leq X_i)$$

Bei dieser Spezifikation wächst die generierte Anzahl der Constraints quadratisch zur Anzahl der Rechtecke. Die große Anzahl von Constraints verbraucht viel Speicherplatz und ist auch in der Generierung der einzelne Constraints aufwendig, hier wäre eine globale Lösung sinnvoll. Die globale Lösung könnte mit speziellen Algorithmen aus dem Bereich des Operation Reseach die Überlappungsfreiheit von Rechtecken effizienter sichern. Eine solche globale Realisierung für die Überlappungsfreiheit von Rechtecken ist das globales Constraint diffn/1.

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

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

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.

Hier wurde nur die Grundform des Constraints diffn/1 dargestellt. Dieses Constraint existiert noch in Varianten mit einer Vielzahl von zusätzlichen Parametern, zu denen zum Beispiel die Festlegung der minimalen und maximalen Fläche aller Rechtecke im Constraint, der Endzeitpunkte der Planung bzw. der Grenzen des Planungs- / Lösungsraums aller Rechtecke, die Distanz zwischen den einzelnen Rechtecken, die Festlegung einer Regionen für die Prüfung von Eigenschaften (Schnittlängen der Rechtecke, Flächeninhalt der Rechtecke, der Abstand zwischen kleinstem und größten Rechteck) der in der Region enthaltenen Rechtecke, der zulässigen Anzahl von Kontakten zwischen den Rechtecken oder die Begrenzung des temporären Planungsbereichs, in dem alle Rechtecke sich befinden können, gehört. Durch diese Parameter wird das Constraint diffn flexibler und es können dadurch komplexere Bedingungen ausgedrückt werden. Die allgemeine Definition des Constraints diffn wird nachfolgend beschrieben.

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 Domänvariablen der Form (O_{11}, ..., O_{mn}, L_{11}, ..., L_{mn}), 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 Domänvariable oder eine ganze Zahl
- $\forall$ i $\in$ [1, m], $\forall$ j $\in$ [1, n]: L_{ij} ist eine Domänvariable 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] $\setminus$ $(O_{ik} \geq O_{jk} + L_{jk}) \vee (O_{jk} \geq O_{ik} + L_{ik})$ [2],

dann sei **Diffn** ein Diffn-Constraint der Form:

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

Beispiel GC2:

In diesem Beispiel sind die drei Rechtecke gegeben, die überlappungsfrei platziert werden sollen. Im nachfolgenden Aufruf der Lösungssuche labeling werden den Domänvariablen 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}]).

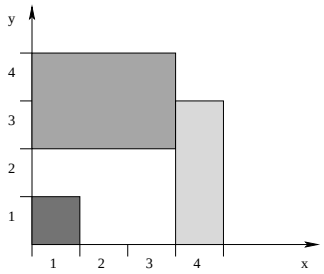

Abbildung 3.7: Beispiel 1 - GC2: Diffn - Constraint

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 Zuschnittproblemen verschiedenster Form gelöst werden. Im dreidimensionalen Raum lässt sich das Packungsproblem von rechteckigen Boxen auf Paletten oder in Container 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.

[2] 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.

Das Constraint diffn ist auch für Produktionsplanungsprobleme einsetzbar. Mit diesem Constraint können **disjunktive** Maschinenzuordnungen von Tasks erfolgen. Hierfür repräsentiert eine Dimension des Constraints die Zeit und jeder Wert einer zweiten Dimension steht für eine mögliche Maschinenzuordnung. Jede Task wird als ein Rechteck repräsentiert, welches durch die Startzeit und die Ressorcenzuordnung positioniert wird. Dieses Rechteck ist in seiner Ausdehnung durch die Bearbeitungsdauer und die Höhe 1 eindeutig bestimmt. Die Bedingung der Nichtüberlappung von Rechtecken sichert, dass bei der Planung keine Task zur gleichen Zeit auf der gleichen Maschine eingeplant werden kann.

Dieses globale Constraint diffn kann auch für die Stundenplanung (timetabling) und Personaleinsatzplanung verwendet werden. Wie in den Arbeiten von Goltz [32] gezeigt wurde, ist die Modellierung von Stundenplänen mit dem Constraint diffn anwendungsfreundlich.
Außerdem können mit dem globalen Constraint diffn andere Constraints nachgebildet werden, wie zum Beispiel der Modellierung von Umrüstzeiten (set-up times) bei den Problemen der Produktionsplanung oder der Sicherung der lokalen Continuität (location continuity) bei Transportproblemen. Die Umrüstzeiten liegen zwischen der Ausführung von zwei Tasks, wenn für die Abarbeitung der nachfolgenden Task die Werkzeuge der Maschine gewechselt werden müssen. Für den Zeitraum der Umrüstung muss die Maschine angehalten werden. Eine Minimierung dieser Stillstandszeiten ist aus Effizienzgründen anstrebenswert. Ist die Zeit zwischen zwei Tasks abhängig von der Reihenfolge der Arbeitsschritte und dem Typ der Maschine, kann das Constraint diffn zur Modellierung dieser komplexen Bedingung eingesetzt werden. Zur Sicherung der lokalen Kontinuität bei Transportproblemen wird das Constraint diffn eingesetzt, um eine Ressource zwischen zwei verschiedenen Orten zu bewegen.
Des Weiteren ist das globale Constraint auch bei Routingproblemen bei dem VLSI -Kanalrouting einsetzbar.

Kapitel 4

Constraintlöser für Multiressourcenprobleme CS_{fd-MR}

Die Constraints sind in ihrer einfachsten Form Gleichungen und Ungleichungen mit Variablen. Die Variablen umfassen Mengen von Werten, die eine potentielle Lösung sein können. Diese Wertemengen der Variablen werden auch Domänen genannt. Aus Constraints und Domänen lassen sich Constraintnetze bilden, die mit speziellen logischen Constraintlösern durch das Finden einer Wertebelegung der Variablen gelöst werden. Während des Lösungsprozesses werden die einzelnen Variablen mit Werten aus ihren Domänen belegt (gelabelt). Durch die Festlegung einer Variablen auf einen Wert ergeben sich Auswirkungen auf die Domänen der anderen Variablen. Wegen der gegebenen Bedingungen im Constraintnetz. Dieser Abgleichprozess wird Propagation genannt und stellt im allgemeinen eine lokale Konsistenz, in verfügbaren Werkzeugen meist die arc-consistency (Knoten- und Kantenkonsistenz) her.

Dieser Abgleichprozess aller noch nicht auf einen Wert gesetzten Variablen (gelabelte Variablen) hat einen Aufwand, der abhängig ist von der Größe der Domänen, der Anzahl der Domänen und der Repräsentation der Domäne als Intervall- oder vollständige Domäne. Bei der vollständigen Domäne werden alle Werte der Domäne propagiert. Bei der Intervalldomäne werden nur die Intervallgrenzen propagiert. Die vollständigen Domänen werden zum Beispiel im CHIP-System bei Domänengrößen bis 100.000 einzelnen Werten angewendet, darüber erfolgt die Repräsentation als Intervalldomänen. In vielen Constraintlösern erfolgt die Umschaltung zwischen den einzelnen Domänrepräsentationen automatisch.

Für die Verfahren zur Herstellung der arc-consistency werden zum Beispiel für den schlimmsten Fall die Laufzeit mit einer Komplexizität von $O(c^n l^n)$ [43] mit n $\in N$ abgeschätzt, mit c der Anzahl der Constraints und l der Größe der Domäne. Diese Verfahren wurde 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 $O(cl^2)$. 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 herrühren. 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 Domänreduzierung [34]. Bei der backtrackbaren Domänreduzierung 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 Domänzerlegung erfolgt nach der Erzeugung des Constraintnetzes eine Auswertung der

Wertebereiche der Domänen. Durch die Propagation in den Constraintnetzen sind meist Bereiche zu erkennen, in denen viele potentielle Lösungen liegen können und Bereiche, in denen wenige Lösungen vermutet werden. Die Bereiche mit der größeren Anzahl an potentiellen Lösungen werden als eine Einheit betrachtet, die als eine Teilzerlegung bezeichnet wird. Die Teilzerlegungen werden durch Auftrennungen der Bereiche der Domänen erzeugt, in denen wenige potentielle Lösungen vorhanden sind. Durch diese Auftrennung wird der Bereich der Lösungssuche verringert und die Effizienz der Lösungssuche gesteigert. Da der Lösungsprozess bei der Vergrößerung der Domäne im Aufwand exponentiell ansteigt, wird durch die Domänzerlegung versucht, den Aufwand gering zu halten. Wenn die durch die Domänenzerlegung ermittelten Teilzerlegungen zu groß sind, werden sie weiter zerlegt, bis eine Größe erreicht ist, die effizient genug zu verarbeiten ist. Das Auffinden der potentiellen Bereiche für die Lösungen, kann Aufwand durch das Auslassen nicht potentieller Bereiche für die Lösung sparen. Der Algorithmus zum Auffinden der potentiellen Bereiche der Lösungen für die Zerlegung der Domänen in Teilzerlegungen ist im folgenden Kapitel im Abschnitt Problemzerlegung beschrieben.
Zusätzlich kann bei der Verwaltung der Domänen angesetzt werden. Derzeit hat jede Variable ihre eigene Domänenrepräsentation im Constraintnetz. Diese einzelnen Domänen müssen zur Herstellung der Konsistenz alle einzeln betrachtet werden. Die Änderung der Verwaltung der Domänen lässt sich am Beispiel von Platzierungsproblemen erläutern. Die zu platzierenden Körper des Platzierungsproblems haben meist Domänvariablen für den Ursprung des Körpers und Domänvariablen oder Festwerte für die Ausdehnung der Körper. In dieser Methode werden nicht in jedem Fall für die Körper des Platzierungsproblems die Domänen mit mehreren Werten für den Ursprung und die Ausdehnung des Körpers erzeugt. Wenn die Ausdehnung oder der Ursprung des Körpers ein Festwert ist, ist die Erzeugung einer Domäne für mehrere Werte nicht notwendig und es wird gleich die Domäne mit einem Festwert angelegt. Nur bei variabler Ausdehnung ist die Erstellung eines Bereiches von möglichen Ausdehnungswerten sinnvoll. Dieses Verfahren wird in einem globalen Constraint umgesetzt und wird im Folgenden Ressourcenconstraint genannt.
Das Ressourcenconstraint erhält alle Körper mit ihren möglichen Werten für den Ursprung und für die Ausdehnung der Körper. Im Ressourcenconstraint sind Domänen für den Platz, auf dem die Körper platziert werden sollen enthalten. Diese Repräsentation mit teilweise festen Werten in den Domänen für die möglichen Plätze und Ausdehnungen der Körper für die Platzierung benötigt einen geringeren Aufwand für die Speicherung und Konsistenzsicherung, als wenn jede einzelne Domäne mit mehreren Werten auf Konsistenz überprüft und gegebenenfalls angepasst werden muss. Die Methode der verringerten Verwaltung von Domänenvariablen mit einem Ressourcenconstraint für n-dimensionale Räume wird in diesem Kapitel im Abschnitt globale Constraints beschrieben.
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 Constraintsystem 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.

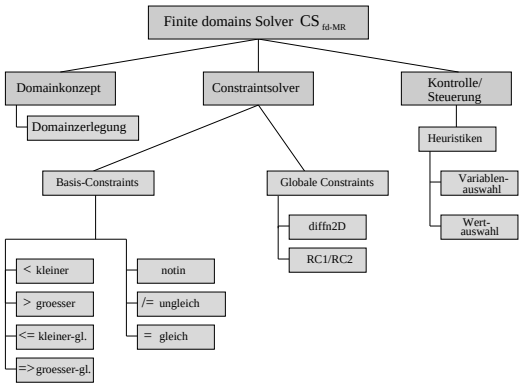

Abbildung 4.1: Komponenten des Constraintsolvers CS_{fd-MR}

Das Constraintsystem CS_{fd-MR}:

$$CS_{fd-MR} = (C_{fd-MR}, D_{fd-MR}, X_{fd-MR}) \text{ mit}$$

- $C_{fd-MR} \subseteq \{0, 1, ..., =, \neq, >, <, \geq, \leq,$ notin, diffn2D, RC1, RC2 $\}$,
- $D_{fd-MR} = N$,
- $X_{fd-MR} \subseteq Var_{fd}$ 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 $R = R_1, ..., R_m$ 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:

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

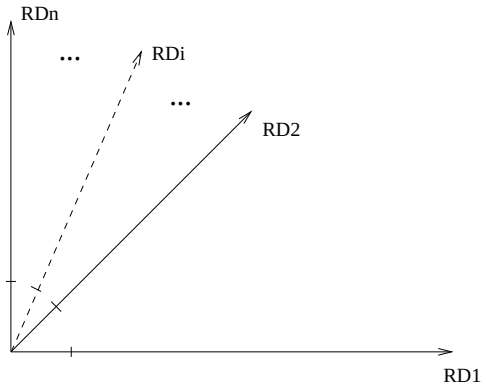

Abbildung 4.2: N - Ressourcendimensionen

Definition R2 `Ereignis`:
Ein Ereignis E benötigt aus jeder Ressourcendimension RD_i eine Ressource R_i für eine Dauer D_i mit i $\in$ der Ressourcendimensionen:

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

Das Ereignis E ist ein Vorgang, der in den einzelnen Ressourcendimension RD_i ($RD_1 = RD_t$) mit i $\in$ der Ressourcendimensionen zu einem bestimmten Startzeitpunkt ZS_i beginnt, eine bestimmte Dauer D_i andauert und zu einem bestimmten Endzeitpunkt ZE_i endet, mit:

$$ZS_i + D_i = ZE_i, \text{ i} \in N$$

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 $\in$ der Ressourcendimensionen Domänenvariablen oder feste Werte sein.

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.

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

Definition R4 `Teilsuchraum`:
Der Teilsuchraum ST_i mit i $\in N$ ist ein Teilstück des gesamten Suchraums S.

$$\forall \text{ i: } ST_i \subseteq S \text{ mit i} \in N$$

Definition R5 `zerlegter Teilsuchraum`:
Der zerlegte Teilsuchraum zST_j mit j $\in N$ 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.

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

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

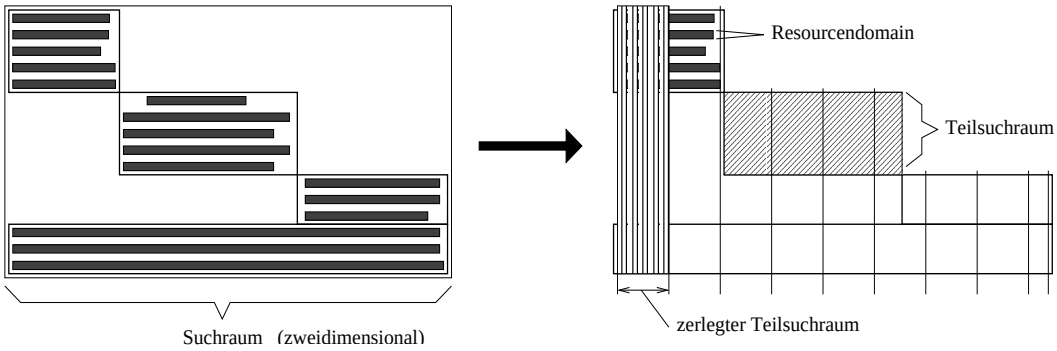

Abbildung 4.3: Suchraum S, Teilsuchraum ST und zerlegter Teilsuchraum zST

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.

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

4.2 Domäne und Constraints

Die Definitionen im Unterabschnitt "Domäne" werden für die Techniken der Domänenzerlegung, der Nichtzeitenhierarchie und der Reihenfolge-Constraints benötigt. Die Domänenzerlegung und die Nichtzeitenhierarchie werden in den folgenden Unterabschnitten definiert. Das Reihenfolge-Constraint wird im nachfolgenden Abschnitt "Globale Constraints" erläutert.

4.2.1 Domäne

Zur Nutzung der Vorteile der lokalen und Grenzen-Konsistenz ist die Einführung einer neuen Domänenstruktur notwendig. Um die Eigenschaft der schnellen Propagation bei der Grenzen-Konsistenz mit der Eigenschaft der wertgenauen lokalen Konsistenz zu verbinden, wird ein Domänentyp eingeführt, der aus Intervallen besteht, die nur gültige Werte enthalten. Wenn in einem Intervall ungültige Werte entstehen, wird das Intervall in mehrere Teilintervalle unterteilt, die dann nur die gültigen Werte des Ausgangsintervalls beinhalten. Durch diesen Ansatz wird der Nachteil der Grenzen-Konsistenz, die ungültige Werte in den Domänen enthalten kann und der Nachteil der lokalen Konsistenz, die einen großen Aufwand zur Konsistenzherstellung benötigen, aufgehoben.

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), ... (A_n, E_n)] mit n $\in$ N, A_n, E_n $\in N$.

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 $\in$ N in der Form D :: [W_1, W_2, ..., W_n], verknüpft werden.

$$\text{D :: AW ... EW} = \text{D[(AW,EW)] oder}$$
$$\text{D :: } [W_1, W_2, ..., W_n] = \text{D[}(W_1, W_n)\text{] mit n} \in N$$

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

Auf diesem neuen Domänentyp können Operation 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 $\in N$ und j < k < l < m < n < o < p < q < r:

- Hinzufügen + mit

1. (a) D[(A_1, E_j), ..., (A_k, E_l)] + D[(A_m, E_n), ..., (A_o, E_p)] = D[(A_1, E_j), ..., (A_k, E_n), ..., (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_n), ..., (A_o, E_p)] + D[(A_1, E_j), ..., (A_k, E_l)] = D[(A_1, E_j), ..., (A_k, E_n), ..., (A_o, E_p)], mit $A_k < E_l < A_m < E_n$, $E_{l+1} = A_m$, (Symmetrie)

3. (b) D[(A_1, E_j), ..., (A_k, E_l)] + D[(A_m, E_n), ..., (A_o, E_p)] = D[(A_1, E_j), ..., (A_k, E_l),(A_m,E_n), ..., (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_n), ..., (A_o, E_p)] + D[(A_1, E_j), ..., (A_k, E_l)] = D[(A_1, E_j), ..., (A_k, E_l),(A_m,E_n), ..., (A_o, E_p)] mit $A_k < E_l < A_m < E_n$, $E_{l+1} < A_m$, (Symmetrie)

5. (c) D[(A_1, E_j), ..., (A_k, E_l)] + E_m = D[(A_1, E_j), ..., (A_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)] + E_m = D[(A_1, E_j), ..., (A_k, E_l), (A_m, E_m)], mit $A_k < E_l < E_m$, $E_{l+1} < E_m$, (Erweiterung am Ende - Anhängen)

7. (d) D[(A_l, 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_l, E_m), ..., (A_o, E_p)] + A_k = D[(A_k, E_k), (A_l, E_m), ..., (A_o, E_p)], mit $A_k < A_l < E_m$, $A_{k+1} < A_l$, (Erweiterung am Anfang - Anhängen)

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_n), ..., (A_o, E_p)], 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_n)] -— D[(A_l, E_l)] = D[(A_1, E_j), ..., (A_k, E_{l-1}), (A_{l+1}, E_n)] mit $A_k < A_l$, $E_l < E_n$, (Splittung - einelementig)

2. (b) D[(A_1, E_j), ..., (A_k, E_n)] -— D[(A_l, E_q), ..., (A_r, E_m)] = D[(A_1, E_j), ..., (A_k, E_{l-1}), (A_{m+1}, E_n)], mit $A_k < A_l$, $E_m < E_n$, $A_l < E_q < A_r < E_m$, $A_k \leq E_{l-1}$, $A_{m+1} \leq E_n$, (Splittung - 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 $A_k < S_l < E_m$, $A_k \leq E_{l-1}$, $A_{l+1} \leq E_m$, (Splittung 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 $A_k \leq S_l < E_m$, $A_k \leq E_l$, $A_{l+1} \leq E_m$, (Splittung mit Rest)

- Entfernen - mit

1. (c) D[(A_1, E_j), ..., (A_k, E_l)] - E_l = D[(A_1, E_j), ..., (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)]

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

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

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

Bei der Aufteilung von großen Suchräumen in kleinere Teilsuchräume und diese in noch kleinere zerlegte Teilsuchräume, müssen die Domänen eventuell zerlegt werden. Die Zerlegung erfolgt an vorher berechneten Stellen der Domänen, diese Stellen werden Zerlegungsgrenzen genannt. Eine Zerlegung der Domänen ist nicht notwendig, wenn die Domäne kleiner oder genau so groß wie die Zerlegungsgrenze ist. Im anderen Fall, wenn die Domäne über die Zerlegungsgrenzen auf beiden Seiten hinausragt, ist eine Zerlegung der Domäne notwendig. Zur Zerlegung der Domäne wird die auf der Domäne definierte Operation Splitten mit Rest verwendet. Hierbei wird am Anfang der Domäne begonnen und immer ein definierten Stück der Domäne abgetrennt. Der verbleibende Rest der Domäne bildet danach eine neue Domäne.

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.

(1): $D[(1, 100)]$ -—**R** $25 = D_1[(1, 25)], D_{12}[(26, 100)]$
(2): $D_{12}[(26, 100)]$ -—**R** $50 = D_2[(26, 50)], D_{23}[(51, 100)]$
(3): $D_{23}[(51, 100)]$ -—**R** $75 = D_3[(51, 75)], D_4[(76, 100)]$

Es ergeben sich dann aus der Domäne $D[\mathbf{(1, 100)}]$ die vier Domänen:

$$D_1[\mathbf{(1, 25)}],\ D_2[\mathbf{(26, 50)}],\ D_3[\mathbf{(51, 75)}],\ D_4[\mathbf{(76, 100)}].$$

4.2.3 Nichtzeitenhierarchie

Bei vielen Ressourcen- und Platzierungsproblemen stehen von anfang an viele ungültige Werte von Domänen fest.
Zum Beispiel gibt es bei einem Stundenplan in einer Bildungseinrichtung Feiertage, Ferien und Wochenenden, aber auch Pausenzeiten. Zu allen diesen Terminen dürfen keine Unterrichtsstunden stattfinden. In der Problemmodellierung würde man die Feiertage, Ferien und Wochenenden auf einer Ebene, da sie volle Tage sperren und die Pausenzeiten auf einer weiteren Ebene zusammenfassen, da dort kleine Zeitspannen innerhalb eines Tages entfallen.
Die Ausschlusszeiten - auch Nichtzeiten genannt - auf den beiden Ebenen sollten zusammengefasst werden und dem CSP bekannt gegeben werden, weil dadurch der Suchraum wesentlich eingeschränkt wird.
Mit dem Konzept der Nichtzeitenhierarchie werden die Nichtzeiten, die auf verschiedenen Modellierungsebenen vorliegen, zur untersten Ebene durchgereicht und dort zusammengefasst. Dies erfolgt mit einer Transformation, die die Listen der Nichtzeiten der einzelnen Modellierungsebenen in einem Vorverarbeitungschritt zu einer Liste zusammenfässt.

Definition D4 `Nichtzeitenliste`:
Die Nichtzeitenliste $N_{i,p}$ mit i $\in N$ der Anzahl der Nichtzeitenlisten und p $\in N$ 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 $\in$ E den Ebenen der Hierarchie und p $\in$ MP 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:

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

Definition D6 `Domänenumwandlung`:
Die Umwandlung einer Domäne D_i in eine Domäne D_{i+1} mit i $\in$ 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:

$$D_i \rightarrow D_{i+1} \setminus N_{i,p} \text{ mit i} \in E, \text{ p} \in \text{MP}.$$

Die Einzelschritte haben dann folgende Form:

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

Beispiel D6:
Gegeben seien Nichtzeitenlisten $N_{x,p}$ mit den Ebenen x $\in$ {G, K, V, P}. Dabei ist $N_{G,p}$ die Nichtzeitenliste des gesamten Plans p, $N_{K,p}$ die Nichtzeitenliste des Kurses K des Plans p, $N_{V,p}$ die Nichtzeitenliste der Veranstaltung V des Plans p und $N_{P,p}$ die Nichtzeitenliste des Praktikums P des Plans p.

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) $D_0 \rightarrow D_1$ / $N_{G,p}$
  D_0[(1,20)] - $N_{G,p}$[(2,2),(6,6),(10,10),(14,14),(18,18)] $\rightarrow$ D_1[(1,1),(3,5),(7,9),(11,13),(15,17)]
- (2) $D_1 \rightarrow D_2$ / $N_{K,p}$
  D_1[(1,1),(3,5),(7,9),(11,13),(15,17)] - $N_{K,p}$[(3,3),(5,5)] $\rightarrow$ D_2[(1,1),(4,4),(7,9),(11,13),(15,17)]
- (3) $D_2 \rightarrow D_3$ / $N_{V,p}$
  D_2[(1,1),(4,4),(7,9),(11,13),(15,17) - $N_{V,p}$[(7,7)] $\rightarrow$ D_3[(1,1),(4,4),(8,9),(11,13),(15,17)]
- (4) $D_3 \rightarrow D_4$ / $N_{P,p}$
  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 Domänentyps 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:

$$\forall \text{ i, j, m, n: } D_1[..., (A_i, E_j), ...] = D_2[..., (A_m, E_n), ...]$$

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:

$$\exists \text{ i, j, m, n: } D_1[..., (A_i, E_j), ...] \neq D_2[..., (A_m, E_n), ...]$$

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 $((E_l < A_m) \vee (A_i = A_m, E_j = E_n, A_k = A_o, E_l = E_p))$.
Beispiel C5: $D_1[(2, 5), (7, 10)] < D_2[(12, 16), (18, 20)]$
$D_1[(2, 5), (7, 10)] = D_2[(2, 5), (7, 10)]$

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 $((A_i > E_p) \vee (A_i = A_m, E_j = E_n, A_k = A_o, E_l = E_p))$.
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 notin(D,Von,Bis) und fordert für die Domäne D, dass die Domäne D alle Wert zwischen den Domänenwerten A_k (Von) und E_l (Bis) nicht enthält (entfernt), wenn diese in der Domäne D existieren sollten:

$$D[..., (A_i, E_j), (A_k, E_l), (A_m, E_n), ...]$$
$$\text{notin}(D,A_k,E_l),$$
$$D[..., (A_i, E_j), (A_m, E_n), ...]$$

mit $A_i < E_j < A_k < E_l < A_m < E_n$ und wenn $\exists\ A_k, E_l$ in D.
Beispiel C7: $D[(2, 5), (8,10), (17, 20)]$, notin($D$,8,10), $D[(2, 5), (17, 20)]$
$D[(2, 5), (8,10), (17, 20)]$, notin($D$,7,15), $D[(2, 5), (17, 20)]$

4.3 Globale Constraints

4.3.1 Ressourcen-Constraint

In diesem Abschnitt soll das globales Ressourcen-Constraint zur Überlappungsfreiheit eingeführt werden. Bei diesem Constraint gibt es eine zentrale Verwaltung der Konsistenz und die zu betrachtenden Objekte werden als Körper bezeichnet. Bei den bisherigen globalen Constraints, zum Beispiel diffn(), hat jeder Körper eigene Domänenvariablen. In diesem Fall kann über die Domänenvariablen eine Propagation der Variablenwerte erfolgen. Bei Problemen, die viele Constraints zwischen den Domänenvariablen enthalten und wenige globale Zusammenhänge, wie Überlappungsfreiheit, die zum Beispiel mit diffn-Constraints gelöst werden, benötigen, ist die Propagation dieser Probleme meist gut.
Bei Problemen, die viele globale Zusammenhänge enthalten, die mit diffn-Constraints modelliert werden und Gleichheitsconstraints mit und ohne Abstandsfunktion enthalten, ist die Propagation meist nicht vorhanden. Besonders wenig Propagation in den Constraintnetzen tritt auf, wenn die globalen Constraints zu Sicherung der Überlappungsfreiheit und die Gleichheitsconstraints zur Abbildung von Reihenfolgebeziehungen genutzt werden. Wenn diese Domänen dann große Werte besitzen und die Körper somit über weite Bereiche verschoben werden können oder eine sehr feine Abbildungen der Realität in großen Domänen erfolgt, ist in der Regel keine Propagation mehr vorhanden. Man sagt dann, das Problem propagiert nicht, da es durch die Constraints im Constraintnetz zu keiner Einschränkung der Werte der Domänen kommt. Erst durch die Platzierung von Körper erfolgt dann eine Propagation.

Im Fall geringer oder keiner Propagation kann die Verwaltung der globalen Konsistenz (meist Überlappungsfreiheit) der Domänenvariablen auch zentral verwaltet werden. Bei der zentralen Verwaltung der Überlappungsfreiheit der Domänen hat nicht jeder Körper seine eigenen Domänen, sondern es gibt nur eine zentrale Domäne, in der alle Werte verwaltet werden. In diese zentrale Domäne werden die einzelnen Körper platziert. Da im Extremfall zu jedem Wert einer Domäne nur maximal ein Körper mit der Ausdehnung von eins platziert wird, wird die Überlappungsfreiheit gewährleistet. Im Suchprozess wird die zentrale Domäne vollständig durchsucht, wobei schon einmal untersuchte Werte einer Domäne, die keine Lösung darstellen, markiert werden und nicht mehrfach angefragt werden, dies reduziert den Suchaufwand durch die zentrale Domäne auf O(n) mit n der Domänengröße. Eine Funktion zur Abfrage freier Domänenwerte der zentralen Domäne ermöglicht die Ermittlung von Domänenbereichen, in denen Körper platziert werden können. Die Gleichheitsconstraints mit und ohne Abstand und die Reihenfolgeconstraints werden bei der Anfrage der zentralen Domäne nicht beachtet. Diese Constraints werden im Suchprozess verwendet und bei der Auswahl der Werte für die Platzierung von Körpern als Zusatzbedingungen beachtet. Wenn für ein Problem die Modellierung die Bildung mehrer zentraler

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.
- c) Zwei oder mehrere Körper sollen einen bestimmten Abstand zu einander haben und eine bestimmte Reihenfolge einhalten, dann werden für diese Körper die betreffenden zentralen Domänen geprüft, ob die gesuchten Werte in allen betreffenden Domänen vorhanden sind. Nur wenn dies der Fall ist, werden die ermittelten Wert als Position für die Platzierung der Körper verwendet.

Dieser Ansatz lässt sich für Probleme einsetzen, bei denen die Überlappungsfreiheit 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

Mit dem globalen Constraint diffnD/1 kann die Nichtüberlappung von einer Menge von n-dimensionalen Körpern im n-dimensionalen Raum modelliert werden. Es werden hierdurch zum Beispiel multidimensionale Platzierungs-, mehrdimensionale Ressourcen- und Packungsprobleme lösbar. Die Idee des Constraints diffnD/1 ist es, das Constraint alldifferent/1, welches die paarweise Verschiedenheit von Domänen sichert, in ein globales Constraint umzusetzen, welches die Überlappungsfreiheit zwischen einer Menge von Objekten im n-dimensionalen Raum modelliert. Zusätzlich zu dieser Eigenschaft werden bei dem Constraint diffnD/1 die Größe jeder einzelnen Dimension definiert und es können gesperrte Bereiche im Lösungsraum vorgegeben werden. Die Definition der Größen der Dimensionen dient zur Abbildung begrenzter Ressourcen, wie sie zum Beispiel bei der Lagerhaltung vorkommen. Mit der Angabe gesperrten Bereichen lassen sich zum Beispiel die geometrische Form von Lagerbehältern oder schon gegebene Vorbelegungen in einem Lager abbilden.
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 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 $(X_i + L_i \leq X_j)$ 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: $(Y_j + H_j \leq Y_i) \vee (Y_i + H_i \leq Y_j) \vee (X_i + L_i \leq X_j) \vee (X_j + L_j \leq X_i)$.
Bei dieser Spezifikation wächst die generierte Anzahl der Constraints quadratisch zur Anzahl der Rechtecke. Die große Anzahl von Constraints verbraucht viel Speicherplatz und ist auch in der Generierung der einzelne Constraints aufwendig, hier wäre eine globale Lösung sinnvoll. Die globale Lösung könnte mit speziellen Algorithmen aus dem Bereich des Operation Reseach die Überlappungsfreiheit von Rechtecken effizienter sichern. Eine solche globale Realisierung für die Überlappungsfreiheit von Rechtecken mit der Angabe der Größe der Dimensionen und von vordefinierten gesperrten Bereichen ist das globales Constraint diffnD/1. Die allgemeine Definition des Constraints diffnD/1 wird nachfolgend beschrieben.

Definition GCR1 `diffnD / diffn2D-Constraint`:
Gegeben sei i der Index für die folgende Liste LRe mit $LRe = [Re_1, ..., Re_m]$ mit Re_i: 1≤i≤m: m $\in N$ und m sei die Anzahl der n-dimensionalen Körper, dann seien n-dimensionale Körper Re_i gegeben durch ein Tupel von Domänvariablen der Form (O_{11}, ..., O_{1n}, V_{11}, ..., V_{1n}), wobei O_{ij} der Ursprung und V_{ij} die Ausdehnung des n-dimensionalen Körpes in der i-ten Dimension ist. Außerdem sei LD^{max} die Liste der maximalen Ausdehnungen der Dimensionen D_i^{max} mit der Form: $LD^{max} = [D_1^{max}, ..., D_n^{max}]$ mit D_i^{max}: 1≤i≤n: n $\in N$ und n sei die Anzahl der Dimensionen und LSp die Liste der n-dimensionalen gesperrten Bereiche Sp_k mit der Form: $LSp = [Sp_1, ..., Sp_o]$ mit Sp_k: 1≤k≤o: o $\in N$ 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_{11}, ..., SO_{1n}, SV_{11}, ..., SV_{1n}), 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 Domänvariable oder eine ganze Zahl
- $\forall$ i $\in$ [1, m], $\forall$ j $\in$ [1, n]: V_{ij} ist eine Domänvariable oder eine ganze Zahl
- $\forall$ k $\in$ [1, o], $\forall$ l $\in$ [1, n]: SO_{kl} ist eine Domänvariable oder eine ganze Zahl
- $\forall$ k $\in$ [1, o], $\forall$ l $\in$ [1, n]: SV_{kl} ist eine Domänvariable oder eine ganze Zahl
- $\forall$ i $\in$ [1, n]: D_i^{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] $\setminus$ $(O_{ip} \geq O_{jp} + V_{jp}) \vee (O_{jp} \geq O_{ip} + V_{ip})$ [1],
- $\forall$ k $\in$ [1, o], $\forall$ l $\in$ [1, n] k $\neq$ l, $\exists$ q $\in$ [1, n] $\setminus$ $(SO_{kq} \geq SO_{lq} + SV_{lq}) \vee (SO_{lq} \geq SO_{kq} + SV_{kq})$ [2],

dann sei **DiffnD** ein diffnD-Constraint der Form:

$$\textbf{DiffnD} = \textbf{diffnD}(LRe,\ LD^{max},\ LSp) =$$

$$\textbf{diffnD}([[O_{11}, ..., O_{1n}, V_{11}, ..., V_{1n}], ..., [O_{m1}, ..., O_{mn}, V_{m1}, ..., V_{mn}]], [D_1^{max}, ..., D_n^{max}],$$
$$[[SO_{11}, ..., SO_{1n}, SV_{11}, ..., SV_{1n}], ..., [SO_{o1}, ..., SO_{on}, SV_{o1}, ..., SV_{on}]]) \text{ mit m, n, o} \in N$$

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

$$\textbf{Diff2D} = \textbf{diffn2D}(LRe,\ LD^{max},\ LSp) =$$

$$\textbf{diffnD}([[O_{11}, ..., O_{1n}, V_{11}, ..., V_{1n}], ..., [O_{m1}, ..., O_{mn}, V_{m1}, ..., V_{mn}]], [D_1^{max}, ..., D_n^{max}],$$
$$[[SO_{11}, ..., SO_{1n}, SV_{11}, ..., SV_{1n}], ..., [SO_{o1}, ..., SO_{on}, SV_{o1}, ..., SV_{on}]]) \text{ mit n} \in \{1, 2\}, \text{m, o} \in N.$$

Beispiel GCR1:

In diesem Beispiel sind drei Körper mit einer Sperrfläche gegeben, die durch ein Constraints diffn2D() überlappungsfrei platziert werden sollen. In der nachfolgenden Prozedur zur Lösungssuche labeling() werden den Domänenvariablen Werte zugeordnet. Das Ergebnis ist in der Abbildung: 4.4 dargestellt.

[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 l ist oder k nach l ist.

bspgcr1():-
$O_1 = [O_{11}, O_{12}, O_{13}] :: 1...7,$
$O_2 = [O_{21}, O_{22}, O_{23}] :: 1...6,$
diffn2D([[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}]).

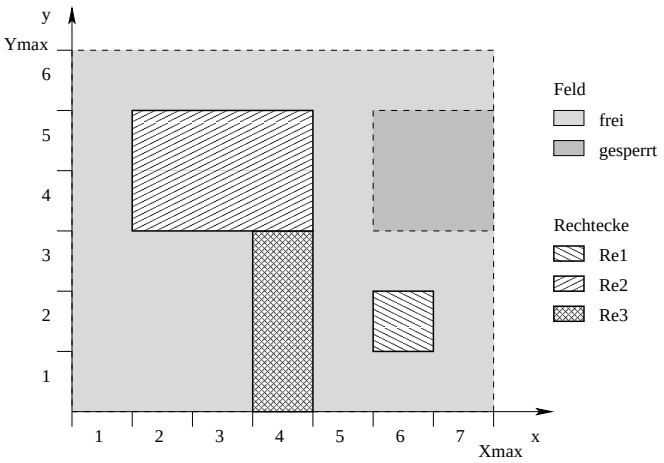

Abbildung 4.4: Beispiel GCR1: diffn2D - Constraint

Das globale Ressourcen-Constraint **diffn2D** 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 Körpern 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$\in N$. 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 **diffn2D**" aufgeführten Funktionen können für ein globales Ressourcen-Constraint **diffn2D** ausgeführt werden. Mit Hilfe der Funktionen wird eine dynamische Planung mit Einplanen, Verschieben und Ausplanen realisiert.

Das Constraint **diffn2D** unterstützt direkt einen zweidimensionaler 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 **diffn2D** 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 $\in N$ 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) $<$ -1 einfügen, aber nicht wieder löschen, verschieben oder mit einem 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-

tusinformationen oder Abhängigkeiten. Der Statuswert f(x, y) = -1 ist definiert als frei, das heißt diese Koordinate ist noch nicht belegt. Freiräume werden die Rechtecke genannt, für die folgende Bedingung erfüllt ist: $\forall\ (o_1 \leq \mathrm{x} < (o_1 + v_1) \wedge o_2 \leq\ \mathrm{y} < (o_2 + v_2))$: f(x, y) = -1. Die Rechtecke werden derartig angeordnet, dass Schnittmengen vermieden werden. Überlappungen von einzelnen Rechtecken werden somit ausgeschlossen. Die Anordnung der Rechtecke hängt ebenfalls davon ab, ob Distanzen definiert sind. Distanzen bestimmen, welche Entfernungen zwischen den einzelnen Rechtecken bestehen sollen. Sie können dynamisch zur Laufzeit geändert werden und beeinflussen immer nur die nachfolgenden Operationen, hiermit werden flexible Anpassungen der Distanzen zwischen den Rechtecken ermöglicht. Freiräume können sowohl spalten- als auch zeilenbezogen ermittelt werden. Auch kann die Schnittmenge verschiedener Freiräume ermittelt werden, wodurch es möglich ist, mit Hilfe mehrerer zweidimensionaler Felder den mehrdimensionalen Fall abzudecken.

4.3.2 Reihenfolge-Constraint

Abbildung der Reihenfolgen

Die Abbildung von Reihenfolgen wird derzeit nicht mit einem globalen Constraint sondern mit den vorhandenen Constraints <, <=, > oder >= umgesetzt. Wobei aber einige neue Eigenschaften nicht nachgebildet werden können. Zum Beispiel die Nachbildung der Eigenschaft, des beliebigen Vertauschens von Ereignissen, ist mit einfachen <- und <=-Constraints nicht möglich.

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

Eine Reihenfolgebeziehung von Ereignissen E_i wird 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 D_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 beiden Startzeitpunktdomänen ZSD_i und ZSD_{i+1} und den 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.

$$ZSD_i + DD_i < ZSD_{i+1} + DD_{i+1}$$
$$ZSD_i + DD_i > ZSD_{i+1} + DD_{i+1}$$

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).

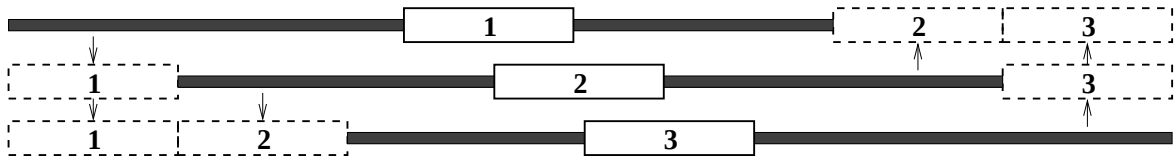

Abbildung 4.5: Reihenfolgebeziehung von drei Ereignissen mit Domänen

Diese Propagation der Domäne an den Anfängen und Enden ist mit Backtracking zurücknehmbar. Durch das Backtracking kann die Wirkung der Constraints < oder > nur schrittweise, in umgekehrter Reihenfolge des Constraintabsetzens zurückgenommen werden. Das Aufheben von zum Beispiel drei Constarints < innerhalb der Absetzreihenfolge der Constraints (nicht die drei zuletzt abgsetzten Constraints), um zwei Ereignisse in ihrer Reihenfolge zu vertauschen, ist nicht ohne das Zurücknehmen weitere Constraints < möglich.
Die Eigenschaft der definierten Reihenfolge der Rücknahme der Constraints und damit der Propagationen durch die Constraints kann in der praktischen Anwendung zu höheren Laufzeiten und damit zu Problemen bei der interaktiven Plangestaltung mit kurzen Antwortzeiten führen. Bei der interaktiven Plangestaltung durch den Nutzer werden eventuell Aktionen gewünscht, die Ereignisse entgegen ihrer vordefinierten Reihenfolge vertauschen, verschieben oder entfernen würden. Diese Aktionen können mit der genannten Verknüpfung durch Constraints < oder > bei großen Planungsproblemen nicht mit kurzen Antwortzeiten realisiert werden. Zum Beispiel eine Vertauschung von Ereignissen ist nicht ohne aufwendiges Backtracking realisierbar, da die Constraint zurück genommen und in einer anderen Reihenfolge abgesetzt werden müssten. Die derzeitigen Constraintsysteme unterstützen eine Zurücknahme von einzelnen, vom Nutzer ausgewählten, Constraints ohne Backtracking nicht. Um die Antwortzeiten zu reduzieren könnte man das Constraintnetz nur lösen und die unveränderten und veränderten Constraints zu einem neuen Constraintnetz verbinden (auch absetzen von Constraints genannt). Jedes Constraintabsetzen kostet aber Zeit und Speicherplatz, da nicht alle Verweise und Referenzen im Speicher gelöscht werden. Bei großen Problemen ist dieses Vorgehen nicht handhabbar, da der zur Verfügung stehende Speicherplatz bei den Constraintsystemen begrenzt ist. Zum Beispiel bei SICStus-System des schwedischen SICS-Instituts beträgt der verfügbare Speicherplatz 256 MB.
Bei der Stundenplanung im interaktiven Modus könnten bei Ereignissen, die in einer vorher festgelegten zeitlichen Reihenfolge geplant werden sollen, vom Nutzer die Aktion des Verschiebens (Entfernen und wieder an anderer Stelle Einfügen) eines oder mehrerer Ereignisse notwendig werden, zum Beispiel wegen Krankheit des Lehrenden. Im interaktiven Modus sollte die Erstellung von notwendigen Vertretungsplänen zeitnah erfolgen. Ein Verschieben von einem Ereignis, bei dem ein Ereignis aus einer bestehenden Folge von Ereignissen entfernt werden muss, führt dazu, dass das nachfolgende Ereignisse in der Reihenfolgebeziehungskette einen neuen Vorgänger erhält. Desweiteren muss die Vorgänger- und Nachfolgerbeziehung am Einfügepunkt in der Reihenfolgebeziehungskette neu organisiert werden. Da der betrachtete Teilsuchraum begrenzt ist, kann unter Umständen nur eine geringe oder gar keine Verschiebung möglich sein. Ein Verschieben (Entfernen und Neueinfügen an anderer Stelle) von Ereignissen ist derzeit nur durch die Rücknahme (Backtracking) und Neudefinition (Absetzen) von Constraints möglich.
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 lösbar ist. Der Nutzer erhält keine Hinweis 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.
Die Verschiebung von Ereignissen in einer Reihenfolgekette durch das Entfernen und an anderer Stelle der Reihenfolgekette wieder Einfügen und die Abschätzung der Löschbarkeit von Reihenfolgebeziehungen soll durch das globalen Reihenfolge-Constraints ermöglicht werden. Das Reihenfolge-Constraint bildet Reihenfolgebeziehungen von Ereignissen ab und wird im folgenden Abschnitt definiert und in seiner Wirkung beschrieben.

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 Anz_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 **ET1** wird nur das Ereignis E_i zur Definition benötigt.
- In dem zweiten Ereignistyp **ET2** kann zu dem Ereignis E_i die Anzahl Anz_i der Ereignisse angegeben werden, die in einem Teilsuchraum geplant werden sollen.
- In dem dritten Ereignistyp **ET3** kann zu dem Ereignis E_i die Anzahl Anz_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 zerlegten Teilsuchraum gleichzeitig geplant werden.

  ET1: $ET1_i \subseteq \{ E_i \}$,
  ET2: $ET2_i \subseteq \{ (E_i, Anz_i) \}$,
  ET3: $ET3_i \subseteq \{ (E_i, Anz_i, [EP_{i1}, EP_{i2}, ..., EP_{in}]) \}$ mit i, n $\in N$

Ereignistypenlisten ETL
Die Ereignistypen werden in Ereignistypenlisten $ETL_i = [ETx_1, ..., ETx_n]$ mit x $\in \{1, 2, 3\}$ und n $\in N$ zusammengefasst. Die einzelnen Ereignistypen ETx_i können miteinander kombiniert werden. Es sind folgende Kombinationen zulässig:

ETL1: $ETL1_i = [ET1_1, ET1_2, ..., ET1_n]$
ETL2: $ETL2_i = [ET2_1, ET2_2, ..., ET2_n]$
ETL3: $ETL3_i = [ET3_1, ET3_2, ..., ET3_n]$
ETL4: $ETL1_i = [ET1_1, ET1_2, ..., ET1_n, ET2_1, ET2_2, ..., ET2_n]$
ETL5: $ETL1_i = [ET1_1, ET1_2, ..., ET1_n, ET3_1, ET3_2, ..., ET3_n]$
ETL6: $ETL1_i = [ET1_1, ET1_2, ..., ET1_n, ET2_1, ET2_2, ..., ET2_n, ET3_1, ET3_2, ..., ET3_n]$

Durch die Möglichkeit der Verarbeitung von auch parallelen Ereignissen zu Ereignissen, die in einer Reihenfolgebeziehung stehen, können Blöcke von Ereignissen gebildet werden. Die Ereignisgruppierungsmöglichkeit wird im Folgenden Blockbildung genannt.
Das Reihenfolge-Constraint RC wird durch eine Liste von Ereignistypenlisten ETL_i definiert. Es gibt zwei Reihenfolge-Constraints RC1 und RC2, die sich durch die verarbeitbaren Ereignislistentypen voneinander unterscheiden. Mit dem Reihenfolge-Constraint RC1 werden alle Varianten von Ereigniskonstellationen, wie parallele, aufeinander folgende, sich wiederholende und blockbildende Ereignisse verarbeitet. Hingegen wird mit dem Reihenfolge-Constraint RC2 nur der spezielle Ereignistyp der Blockbildung beschrieben. Die Unterscheidung ermöglicht vereinfachte Algorithmen für den komplizierten Fall der Blockbildung, der in verschiedensten Anwendung vorrangig vorkommt (Stundenplanung und Fahrplanung).

Reihenfolge-Constraint RC1

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

Beispiel RC1:

1. RC1([[E_1, E_2, E_3], [E_6, E_8, E_{10}]]),
2. RC1([[(E_1,Anz_1), (E_2, Anz_2), (E_3, Anz_3)], [(E_5,Anz_5), (E_{13}, Anz_{13}), (E_{20}, Anz_{20})]]),
3. RC1([[(E_1,Anz_1,[E_2, E_3]), (E_5, Anz_2), (E_7, Anz_7)], [(E_8,Anz_8), (E_{10}, Anz_{10}), (E_{17}, Anz_{17})]]),

Reihenfolge-Constraint RC2

Definition Reihenfolge-Constraint RC2
Das Reihenfolge-Constraint **RC2** hat eine Liste von Ereignistypenlisten $ETLx$ mit x $\in \{3\}$. In diesem Reihenfolge-Constraint darf nur die Ereignistypenliste ETL3 mit dem Ergebnistyp $ET3_i$ 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 einem zerlegten Teilsuchraum zST_k mit k $\in N$ der Anzahl der Teilsuchräume zuordnen. Die reduzierten Domänen können auch wieder erweitert werden, indem eine andere Zuordnung zu einem zerlegten Teilsuchraum zST_k erfolgt.
Gegeben seien die Mengen der Ressourcenkapazitäten $MRK_{j,k}$ für die Kapazitäten $KD_{i,j,k}$ mit i $\in N$ der Anzahl der Kapazitäten der Ressourcendimensionen RD_j für jeden zerlegten Teilsuchraum zST_k und eine Menge von Ereignisse ME mit E_l: 1, ..., l, ..., n: n $\in N$ der Anzahl der Ereignisse. Es gibt drei Gruppen von Ereignissen: $E_l \subseteq \{Eg_l, Evb_l, Ef_l\}$ die in Abhängigkeit von ihrer Variablität in der Zuordnung zu den zerteilten Teilsuchräumen unterschieden werden:

- Die Ereignisse Eg_l werden **genau** einem zerlegten Teilsuchraum zu geordnet.
- Die Ereignisse Evb_l werden **von** einem zerlegten Teilsuchraum beginnend **bis** zu einem letzten zerlegten Teilsuchraum fortlaufend zugeordnet, solange sie noch nicht eingeplant sind.
- Die Ereignisse Ef_l sind **frei** in ihrer Zuordnung zu einem zerlegten Teilsuchraum und werden bei vorhandenen Kapazitäten in einem zerlegten Teilsuchraum diesem zerlegten Teilsuchraum zugeordnet.

und **RCx(**ME,zST_k,$MRK_{j,k}$**)** für x $\in$ { 1, 2 }:

RCx([]**,**zST_k**,**$MRK_{j,k}$**).**
RCx([$E_{Teilliste}$‖**Rest**]**,**zST_k**,**$MRK_{j,k}$**):-**
ermittle-nichtgeplante-Ereignisse(zST_{k-1},$E_{Nichtgeplantliste}$),
verschiebe-nichtgeplante-Ereignisse(zST_k,$E_{Nichtgeplantliste}$,$MRK_{j,k}$),
bearbeite($E_{Teilliste}$,zST_k,$MRK_{j,k}$),
RCx(Rest,zST_{k+1},$MRK_{j,k+1}$).

Die Ermittlung nichtplanbarer Ereignisse im zerlegten Teilsuchraum zST_{k-1} (Teilschritt 1) erfolgt in `ermittle-nichtgeplante-Ereignisse(`zST_{k-1}`,`$E_{Nichtgeplantliste}$`)`. Jedes Ereignis E_l besitzt eine Eigenschaft `geplant`, die den Planungszustand angibt. Es gibt die zwei Planungszustände `geplant` und `ungeplant`. Ermittelt werden alle Ereignisse E_l, die `ungeplant` sind und diese Ereignisse werden in der Liste $E_{Nichtgeplantliste}$ weitergegeben.
Die Verschiebung nichtplanbarer Ereignisse in den nächsten zerlegten Teilsuchraum zST_k bei gleichzeitigem Abgleich der benötigten Ressourcen in den einzelnen Ressourcendimensionen RD_j (Teilschritt 2) erfolgt in `verschiebe-nichtgeplante-Ereignisse(`zST_k`,`$E_{Nichtgeplantliste}$`,`$MRK_{j,k}$`)`. Die Ereignisse E_l der Liste $E_{Nichtgeplantliste}$ werden entsprechend ihrer Zuordnungsmöglichkeiten zu den zerteilten Teilsuchräumen zugeordnet. Die Ereignisse aus der Gruppe Eg_l, die nur genau einem zerlegten Teilsuchraum zugeordnet werden konnten, werden nicht neu zugeordnet und bleiben ungeplant. Die Ereignisse der Gruppe Evb_l werden im Bereich der angegebenen, zerlegten Teilsuchräumen neu zugeordnet. Wenn

der Bereich der zerlegten Teilsuchräume ausgeschöpft ist und keine Einplanung erfolgen konnte, bleiben die Ereignisse ungeplant. Die Ereignisse der Gruppe Ef_l werden dem nächsten zerlegten Teilsuchraum zugeordnet. Erst wenn der gesamte Suchraum bearbeitet wurde und das Ereignis nicht eingeplant werden konnte, bleibt es ungeplant. In den Zustand `ungeplant` kann ein Ereignis E_l 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 Teilsuchraum zST_k (Teilschritt 3) erfolgt in `bearbeite(`$E_{Teilliste}$`,`zST_k`,`$MRK_{j,k}$`)`. Die in der Liste $E_{Teilliste}$ dem zerlegten Teilsuchraum zST_k zugeordneten Ereignisse werden nun in einem Suchprozess unter Beachtung der bekannt gegebenen Constraints eingeplant. Hierbei werden in den Mengen der Ressourcenkapazitäten $MRK_{j,k}$ Kapazitäten $KD_{i,j,k}$ der jeweiligen Ressourcendimension RD_j verbraucht. Hiernach erfolgt der Aufruf des Constraints für den nächsten zerlegten Teilsuchraum zST_{k+1}.

Durch das Ermitteln nichtgeplanter Ereignisse innerhalb der Teilsuchräume während des Lösungsprozesses werden die Reihenfolgeketten von Ereignissen regelmäßig überprüft. Beim Eintreten einer Nichteinplanung eines Ereignisses auf grund eines Ressourcenengpasses innerhalb eines Teilsuchraums wird das entsprechende Ereignis und folgenden Ereignisse unter Einhaltung der Reihenfolge in den nächsten Teilsuchraum übernommen. Es handelt sich hier um eine Verschiebung auf grund eines Ressourcenengpasses ohne Anpassung der Vorgänger- und Nachfolgerbeziehungen in der Reihenfolgekette. Im letzten Teilsuchraum können dann die Ereignisse angegeben werden die auf grund von Ressourcenbeschränkungen nicht eingeplant wurden. Hierdurch kann dem Nutzer bekannt gegeben werden, welche Ereignisse zu überprüfen sind. Es kann sich dann um eine Beschränkung in der zeitlichen Ressource oder in den anderen Ressourcen zum Beispiel in den Räumen oder Lehrkräften handeln.

Im interaktiven Modus der Stundenplanung können Ereignisse innnerhalb eines oder über mehrere Teilsuchräume 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 nachfolgendnen Ereignisses auf das davorliegende Ereignis um und fügt es an der neuen Position (entsprechend des interaktiv festgelegten Zeitpunkts) wieder in die Reihenfolgekette, durch Ändern 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 folgendem 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 `bearbeite(`$E_{Teilliste}$`,`zST_k`,`$MRK_{j,k}$`)` verwendete Suchprozess zur effizienten Planung näher erläutert.

Kapitel 5

Suchverfahren für effiziente Planungen

5.1 Problemzerlegung

In diesem Kapitel wird ein Suchverfahren für effiziente Planungen vorgestellt. Der Aufwand für die Lösungssuche steigt mit der Vergrößerung des Problems exponentiell an. Um auch bei größeren Problemen eine Lösungsermittlung in angemessener Zeit zu ermöglichen, wird nun versucht, den Lösungsaufwand zu verringern. Die Verringerung des Lösungsaufwands wird durch die Zerlegung von mindestens einer Dimension des Suchraums und die Anwendung der Lösungssuche auf diese zerlegten Teilsuchräume erreicht. Diese Vorgehensweise entspricht dem Prinzip Divide-and-Conquer (dt. teile und herrsche). Bei dem Lösungsverfahren Divide-and-Conquer wird ein Problem in kleinere Teilprobleme zerlegt, die Teilprobleme dann einzeln gelöst und danach aus den Teillösungen eine Gesamtlösung erstellt [62]. Ein Beispiel für die Nutzung des Prinzips Divide-and-Conquer ist Quicksort nach Divide-and-Conquer, dessen Komplexität in Durchschnitt bei $O(n\ log\ n)$ liegt und im ungünstigsten Fall bei sortierten Listen $O(n^2)$ beträgt [69]. Durch diese Verfahrenweise werden geringe Aufwände für die Lösungssuche in den zerlegten Teilsuchräumen erreicht, die sich aufaddieren und somit einen wesentlich geringer Aufwand haben als der Aufwand für die Lösungssuche im gesamten Lösungsraum.
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)

Bei der Lösung großer Planungsprobleme ist durch die einzuhaltenden Bedingungen eine Verringerung der Problemgröße gegeben. Die Verringerung der Problemgröße wird durch Ausschluss von Werten erzeugt, die nicht zur Erzeugung der Lösung benutzt werden dürfen. Der Ausschluss von Werten erfolgt meist bezüglich der Zeitressource. Zum Beispiel sind in bestimmten Zeitabschnitten (Wochenenden und Feiertagen) keine Ereignisse einzuplanen. Diese nicht planbaren Zeitabschnitte (auch Nichtzeiten genannt) sind hierarchisch geordnet. In der ersten Hierarchiestufe sind die immer wiederkehrenden Nichtzeiten enthalten, wie Wochenenden und Pausenzeiten. In der zweiten Hierarchiestufe sind dann speziellere Nichtzeiten enthalten, wie Feiertage, die sich von Bundesland zu Bundesland unterscheiden. Die Informationen in den Hierarchiestufen werden somit immer feiner und spezieller.
Durch die einzuhaltenden 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.

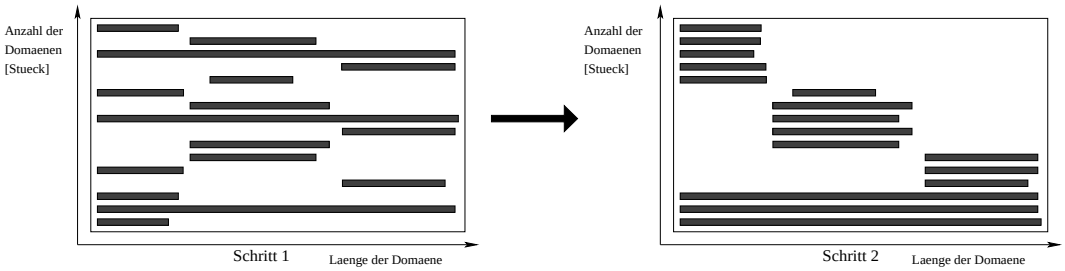

Abbildung 5.1: Sortierung der Domänen und Bildung einer Treppenstruktur

Die Treppenstruktur definiert die Bereiche im Suchraum, in denen die Lösung liegt. Die Bereiche, die keine Werte enthalten, liefern keinen Beitrag zur Lösungsfindung. Diese Bereiche ohne Werte wurden durch die abgesetzten Constraints ausgeschlossen (siehe Bild 5.1). Die Sortierung der Bereiche nach dem Absetzen der Constraints führt zu einer Reduzierung des Suchraumes. Die Abschätzung der oberen Grenze des Suchaufwandes vor dem Absetzen der Constraints und danach wird im Folgenden betrachtet.
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:

$$\mathrm{O(Y * X)} \text{ mit X,Y} \in N.$$

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:

$$\sum_{i=1}^{J} \mathrm{O}(Y_i * X_i).$$

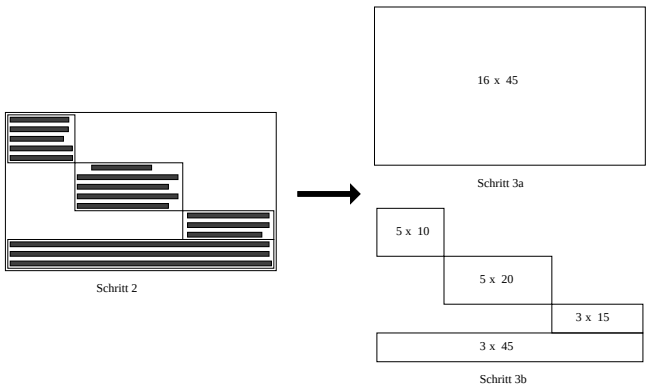

Abbildung 5.2: Abschätzung der oberen Grenze des Suchaufwandes

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

In Schritt 3a ergibt sich der Aufwand aus der Anzahl der Domänen und der Länge der Domänen mit O(16 * 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 * 10) + O(5 * 20) + O(3 * 15) + O(3 * 45) = O(50) + O(100) + O(45) + O(135) = O(330).

Es kommt durch das Absetzen der Constraints zu einer Reduzierung des Suchraums. Vor der Sortierung der Bereiche des Suchraums musste der gesamte Suchraum mit 720 Suchschritten durchsucht werden. Nach dem Absetzen der Constraints sind zur Lösungsermittlung nur noch 330 Suchschritte notwendig. Für dieses Beispiel ergibt das eine Reduzierung um 54,16 %. Dieser Faktor ist problemspezifisch und schwankt von Problem zu Problem. Im ungünstigsten Fall ist der Reduzierungsfaktor 1.

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 Suchraums nicht mehr durchgeführt.

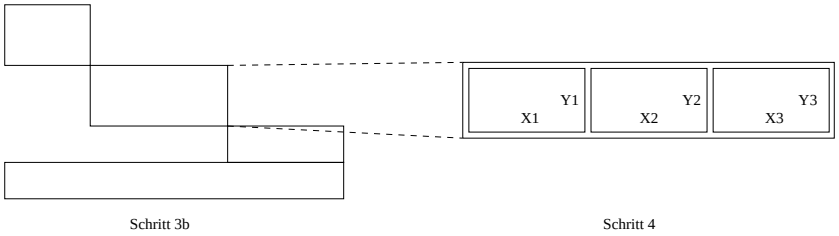

Abbildung 5.3: Unterteilung der Suchräume

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

Zerlegung in Teilsuchräume (Vorgang 2)

Die ermittelten Bereiche aus dem Vorgang 1 werden nun im Vorgang 2 auf die Häufigkeit des Vorkommens von Werten in den Domänen der Ressourcen untersucht. Die Werte der Domäne in den Ressourcen werden im Schritt 5 in zwei Gruppen eingeteilt. In der Gruppe 1 sind alle Domänen, deren Anzahl der Werte in den einzelnen Bereichen kleiner als die Hälfte der Gesamtgröße des Suchraums ist (oberer Bereich des Schrittes 5 im Bild 5.4). In der Gruppen 2 sind alle Domänen, deren Anzahl der Werte der Ressourcen in den einzelnen Bereichen größer und gleich der Hälfte der Gesamtgröße des Suchraums sind (unterer Bereich des Schrittes 5 im Bild 5.4). Auf dieser Gruppeneinteilung der Domäne werden die Häufigkeiten des Auftretens der einzelnen Werte in den Domänen der Ressourcen ermittelt. Es werden die wertgleichen Werte der unterschiedlichen Domänen aufaddiert. Die Summen werden für die zwei definierten Gruppen einzeln gebildet. Die entstandenen Häufigkeitsverteilungen werden vom Algorithmus in horizontaler Ebene auf die Möglichkeiten der Unterteilungen in die Teilsuchräumen untersucht. Solche Unterteilungen ergeben sich bei Planungsproblemen mit einer Zeitachse durch den Ausschluss von zum Beispiel Wochenenden und Feiertagen. Diese ausgeschlossenen Tage ergeben eine Lücke in dem Suchraum. Die nebeneinander liegenden Summen von Häufigkeiten der Domänenwerte werden zu einem Bereich zusammengefasst. 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 5.4 Schritt 6).

Im Beispiel ist folgende Anzahl von gleichlangen Bereichen von Werten der Domänen der Ressourcen (auch Ressourcendomänen genannt) ermittelt worden:

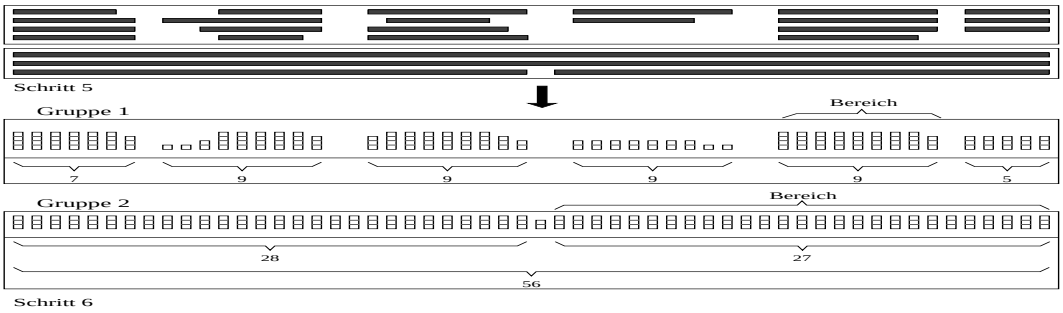

Abbildung 5.4: Bestimmung der Zerlegungspositionen

Bereiche von Domänen der Ressourcen für das Beispiel		
Gruppe	Länge eines Bereichs [Anzahl der Domänenwerte]	Anzahl gleichgroßer Bereiche
1	9	4
1	7	1
1	5	1
2	56	2
2	28	1
2	27	1

In der Gruppe 1 (< 50 % der Teilbereichsgröße) werden die Bereiche der Werte der Domäne der Ressource ausgewählt, die den größten Bereich haben und gleichzeitig eine hohe Anzahl haben. Es wird das Maximum $ZTSR0$ über die Anzahl und die Größe der Bereiche gesucht. Das Maximum $ZTSR0$ ergibt dann die vorläufige Größe des zerlegten Teilsuchraumes.

$$\text{ZTSR0} = \max(\text{Größe}, \text{Anzahl})$$
$$\text{ZTSR0} = \max(9, 4) \rightarrow 9.$$

Der zerlegte Teilsuchraum zST soll gleichmäßig sein und sich an den Grenzen überlappen. Für die Überlappung wird der zerlegte Teilsuchraum 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 Ressource berücksichtigt und sich aus dem aufgerundeten Wert des Quotients aus dem größten Wert der Domäne der Ressourcen des Bereichs MW und der Größe $ZTSR0$ des ausgewählten zerlegten Teilsuchraums zST ergibt. Die endgültige Größe $ZTSR$ des zerlegten Teilsuchraums zST ergibt sich nach folgender Formel:

$$ZTSR = \max((ZTSR0 + 3), \text{rund}(MW \ / \ ZTSR0))$$
$$ZTSR = \max((9 + 3), \text{rund}(\ 56 \ / \ 9)) = \max((9 + 3), 6) = \max(12, 6) = 12.$$

Im Schritt 7 wird dann nach der endgültigen Größe $ZTSR$ des zerlegten Teilsuchraumes der Teilsuchraum in mehrere kleinere zerlegte Teilsuchräume zST in horizontaler Ebene umgrupppiert (siehe Bild 5.5 Schritt 7). Die Teilsuchräume können am Anfang und Ende des Suchbereichs kleiner sein als die Größe des zerlegten Teilsuchraums $ZTSR$. Die Größe des zerlegten Teilsuchraums $ZTSR$ 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

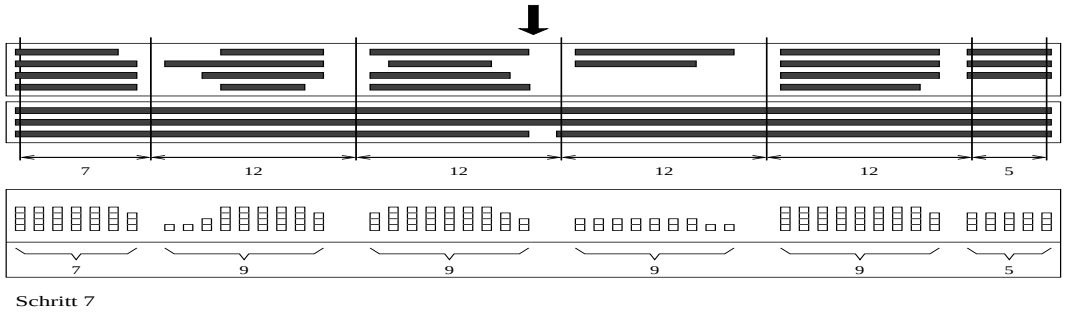

Abbildung 5.5: Zerlegung des Teilsuchraums an den Zerlegungspositionen

und reisen zum Anfang der Woche an und am Ende der Woche wieder ab. Im Stundenplan wird hierfür eine An- und Abreisezeit von zwei Stunden eingeplanten. Die ersten zwei Stunden am Montag und die letzten zwei Stunden am Freitag einer Woche sind für die An- und Abreise reserviert.
Die Planung eines Stundenplans umfasst in der Regel ein bis zwei Jahre mit in wöchentlich wechselndem Unterricht. In den einzelnen Fächern existiert eine bestimmte Aufeinanderfolge der Veranstaltungen von der Basisveranstaltung bis zur Spezialisierungsveranstaltung. Hierbei bietet sich die Modellierung mit einer zusammenhängenden Zeitachse über einen Zeitraum von ein bis zwei Jahre an.
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 Gruppen sind alle Domänen, deren Anzahl

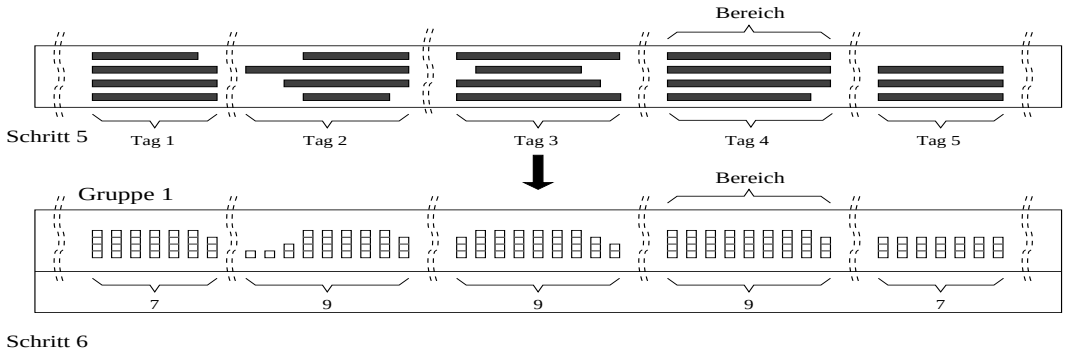

Abbildung 5.6: Bestimmung der Zerlegungspositionen

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

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 zusammengefasst. 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.

Bereiche von Domänen der Ressourcen für das Beispiel		
Gruppe	Länge eines Bereichs [Anzahl der Domänenwerte]	Anzahl gleichgroßer Bereiche
1	9	3
1	7	2
2	168	0

ZTSR0 = max(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 Aufteilung 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.

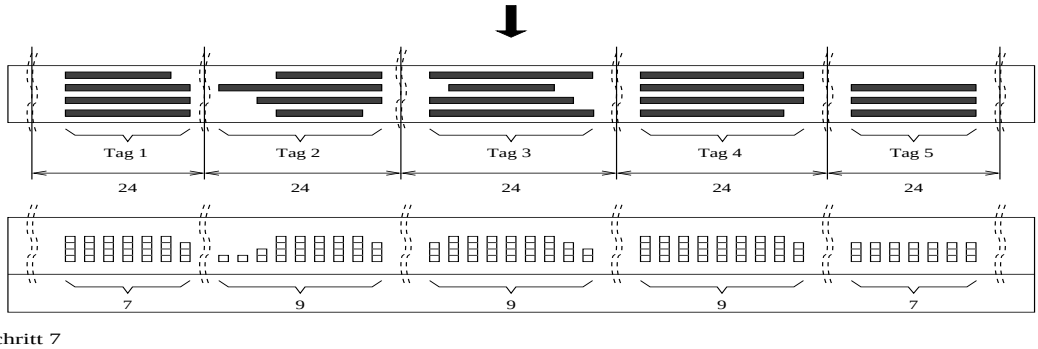

Abbildung 5.7: Zerlegung des Teilsuchraums an den Zerlegungspositionen

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 ermöglichen, werden die Pausenzeiten hier nicht betrachtet. Aus den Domänen werden hierfür keine Werte für die Abbildung der Pausenzeiten entfernt.
Bei der weiteren Betrachtung des Beispiels der Stundenplanung an Weiterbildungseinrichtungen wird von einer Zerlegung in Tagen ausgegangen, die in Wochen zusammengefasst sind.

Kapazitätsermittlung der zerlegten Teilsuchräume (Vorgang 3)

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

Definition Kapazität KD
Die Kapazität $KD_{i,j,k}$ der Ressourcendomäne $D_{i,j,k}$ sei die Anzahl der Element der Ressourcendomäne $D_{i,j,k}$ mit 1≤ i ≤ n mit n der Anzahl der Ressourcen, j > 1; 2 ≤ j ≤ m mit m der Anzahl der Ressourcendimensionen und 1 ≤ k ≤ p mit p der Anzahl der zerlegten Teilsuchräume, die sich in Abhängigkeit von der Ressource R_1 der Ressourcendimension RD_1 ergibt, wenn mit der Ressourcendimension RD_1 die Ressource Zeit t modelliert ist. Die Ressource R_1 der Ressourcendimension RD_1 wird dann mit R_t bezeichnet.

$$KD_{i,j,k} = \text{AnzahlElemente}(D_{i,j,k})$$
$$\text{mit i} = 1, ..., \text{n; j} = 2, ..., \text{m; k}= 1, ..., \text{p; n, m, p} \in N$$

Definition Ressourcenkapzität MRK
Die Ressourcenkapazität $MRK_{j,k}$ sei die Menge der Kapazitäten $KD_{i,j,k}$ der Ressourcendimensionen RD_j der zerlegten Teilsuchräume zST_k für jede Ressource $R_{i,j}$ einer Ressourcendimension RD_j mit 1≤ i ≤ n mit n der Anzahl der Ressourcen, j > 1; 2 ≤ j ≤ m mit m der Anzahl der Ressourcendimensionen und 1 ≤ k ≤ p mit p der Anzahl der zerlegten Teilsuchräume.

$$MRK_{j,k} = \bigcup KD_{i,j,k}$$
$$\text{mit i} = 1, ..., \text{n; j} = 2, ..., \text{m; k}= 1, ..., \text{p; n, m, p} \in N$$

Beispiel ZL2:

Gegeben sind eine Ressourcendimension RD_2 eines zerlegten Teilsuchraums zST_1 mit sechs Ressourcen $R_{i,2}$ mit 1 ≤ i ≤ 6 der Ressourcendimension RD_2 und den dazugehörigen Ressourcendomänen $D_{i,2,1}$ über der Ressource Zeit R_t der Ressourcendimension RD_1, für die die Menge der Ressourcenkapazitäten $MRK_{2,1}$ der einzelnen Kapazitäten $KD_{i,2,1}$ ermittelt wird.

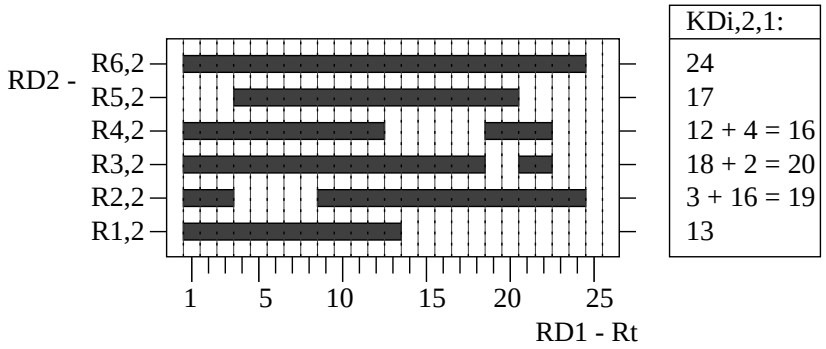

Abbildung 5.8: Kapazitäten der Ressourcendimension RD_2 zur Ressourcendimension RD_1

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 zST_k und deren ermittelte Mengen der Ressourcenkapazitäten $MRK_{j,k}$ mit 1 ≤ j ≤ n, 1 ≤ k ≤ n:n ∈ N werden danach der Planungskomponente übermittelt.

5.1.2 Planung

Die Planung soll am Beispiel der Kursplanung für Weiterbildungseinrichtungen erläutert werden. Der Planungprozess untergliedert sich in zwei Teile. Der erste Teil der Planung umfasst die Initialisierung der Planungskomponente und der zweite Teil den eigentliche Planungsprozess. Die Planung kann sowohl interaktiv, als auch automatisch stattfinden, wobei sich dieser Abschnitt vor allem auf die automatische Planung beschränkt werden soll. Um die Planungskomponente schnell zu halten, musste sie in einer Sprache geschrieben werden, aus der ein Compiler schnellen nativen Code generieren kann. Dazu eignete sich am besten C^{++}. Während des Ablaufes der einzelnen Teilbereiche werden ständig Mitteilungen generiert, die während der Ausführung abgefragt und ausgegeben werden können.

Initialisierung

Um den Planungsaufwand zu reduzieren, werden zuvor die für die Planung notwendigen Daten ermittelt, auf die dann schnell zugegriffen werden kann. Die Veranstaltungen werden auf konsistente Daten geprüft. Beim Auftreten von inkonsistenten Daten in den Definitionen der Veranstaltungen werden diese Veranstaltungen ausgeplant, falls sie eingeplant waren. Dies kann zum Beispiel dann auftreten, wenn bereits der Planungsvorgang zu einem früheren Zeitpunkt stattfand und nachträglich bei einer Definition zu einer Veranstaltung die Daten verändert wurde. Bei der Initialisierung werden Relationen von den Veranstaltungsdefinitionen zu den entsprechenden Veranstaltungen verwaltet. Innerhalb dieser Relationen werden Listen zusammengestellt, die sortiert nach Wünschen, Präferenzen und möglichen Ressourcen die Dozenten und Räume zu den Definitionen der Veranstaltungen enthalten. Außerdem werden die Zeitwünsche vorsortiert. Auf dieser Grundlage werden die Abhängigkeiten in Bezug auf die Reihenfolge der Veranstaltungen ermittelt. Dies umfasst die Bildung von Blockstunden und zeitgleichen Stunden.
Die ungeplanten Veranstaltungen sind zu Beginn der Planung bereits vorsortiert. Um die Suche zu beschleunigen, ist der gesamte Planungszeitraum mehrfach unterteilt. Er ist zum einem 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.

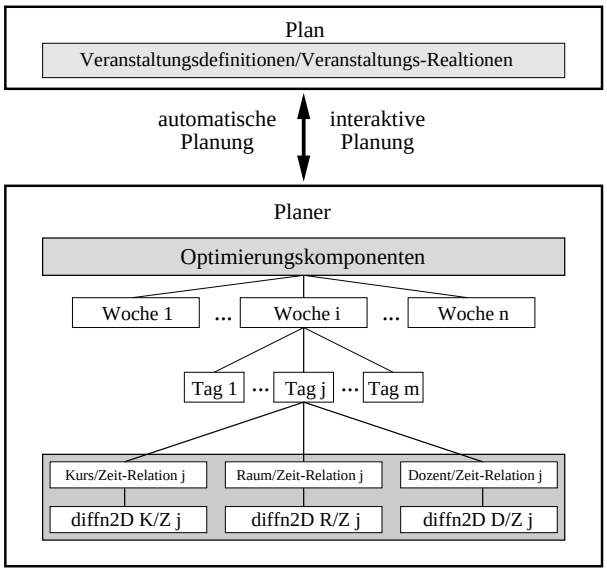

Abbildung 5.9: Aufbau der Planungskomponente für das Beispiel der Kursplanung an Weiterbildungseinrichtungen

Die Tage sind untergliedert in die drei Relationen: Zeit/Dozent, Zeit/Raum und Zeit/Kurs (siehe Abbildung 5.9). Jede Relation wird durch ein globales Constraint **diffn2D** umgesetzt, welches die überschneidungsfreie Ressourcenverteilung sicherstellt. Hierbei wird in der ersten Dimension des zweidimensionalen globalen Constraints **diffn2D** die Zeit abgebildet und in der zweiten Dimension die dazugehörige Ressourcendimension. Die Ressourcendimensionen sind in diesem Beispiel: Dozent, Raum und Kurs.
Um weitere globale Überprüfungen und Optimierungen zuzulassen, besteht die Möglichkeit, zusätzliche globale Optimierungskomponenten zu definieren, die Funktionalitäten bereitstellen, die während der Planung aufgerufen werden. Genutzt wird dieses Konzept zur Zeit zum Beispiel bei der Sicherstellung der maximalen Stundenanzahl der Dozenten. In dieser Phase werden außerdem bereits schon in vorherigen Planungen eingeplante und gespeicherte Veranstaltungen berücksichtigt. Kommt es hierbei zu Verletzungen von Restriktionen, so werden die Veranstaltungen als ungeplant gekennzeichnet und damit ausgeplant. Dies kann zum Beispiel dann auftreten, wenn nach der letzten Planung zusätzliche Restriktionen ergänzt wurden und nun das Einplanen der Veranstaltung verhindern.

Die Einteilung in Wochen dient der Grobverteilung der Veranstaltungen innerhalb des Planungszeitraumes zur Sicherung der Reihenfolgebedingungen. Diese Unterteilung bietet sich wegen der unterrichtsfreien Wochenenden an, die eine Unabhängigkeit zwischen den einzelnen Wochen erzeugen. In einigen Bildungseinrichtungen sind die Stundenpläne in jeder Woche gleich, weshalb in den einzelnen Wochen nur die Vertretungsstunden geplant werden müssen. Der restliche Hauptstundenplan kann ohne Veränderungen übernommen werden. Jede Woche besitzt eine Liste, die die ihr zugeteilten Veranstaltungen entsprechenden der Reihenfolgebeziehungen oder des Hauptstundenplans enthält. Um den größtmöglichen Erfolg bei der Planung sicherzustellen, unterliegt die Sortierung dieser Listen zusätzlichen Kriterien. So sind die Elemente zum einen danach sortiert, ob sie Zeitwünsche besitzen und zum anderen danach, wie viele Ressourcen ihnen in den einzelnen Ressourcendimensionen (zum Beispiel Räume und Dozenten) bei der Planung zur Auswahl stehen. Dabei bekommen Veranstaltungen mit geringerer Anzahl von möglichen planbaren Dozenten und Räumen eine höhere Priorität zugewiesen, als Veranstaltungen mit einer höheren Anzahl von Varianten der Ressourcenzuordnungen. Dies stellt sicher, dass auch bei beschränkten Ressourcen noch Möglichkeiten gefunden werden können, diese Veranstaltungen einzuplanen.

Das Einplanen

Der eigentliche Planungsprozess für eine ungeplante Veranstaltung umfasst zwei Schritte. Als erstes werden die Zeitwünsche zu einer Veranstaltung auf ihre Realisierbarkeit getestet. Sollte kein geeigneter Zeitpunkt gefunden werden, an dem ein Dozent und ein Raum zur Verfügung steht, so muss ein alternativer Zeitpunkt gesucht werden. Hierfür werden an jedem Tag die Zeiten ermittelt, an denen für den Kurs oder die Kursgruppe noch keine Veranstaltungen vorgesehen sind. Je nach Priorität der Ressourcen Raum und Dozent werden nacheinander Schnittmengen zwischen den zuvor ermittelten Zeiten und den freien Zeiten der höherpriorisierten Ressource gebildet. Wurden Schnittmengen zwischen den ermittelten und freien Zeiten der höherpriorisierten Ressource gefunden, wird die Schnittmenge der übrigen Zeiten mit der niedriger priorisierten Ressource gebildet. Dabei wird so vorgegangen, dass nacheinander Schnittmengen mit den Wünschen, anschließend mit den bevorzugten Ressourcen und letztlich mit den nutzbaren Ressourcen gebildet werden. Dieser Prozess beginnt am Anfang des Planungszeitraums und wird so lange fortgeführt, bis entweder das Ende des Planungszeitraums erreicht ist, oder bis ein Zeitpunkt gefunden wurde, an dem die Veranstaltung eingeplant werden kann.
Durch diese Vorgehensweise wird eine übermäßige Generierung von Freistunden verhindert, da die Veranstaltung immer zum frühest möglichen Zeitpunkt eingeplant wird. Dies ist eine Heuristik, in deren Ergebnis 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 Wochentage 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 belieben 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

Das Ausplanen von Veranstaltung ist im Vergleich zum Einplanen einfacher. Es wird nacheinander versucht, belegte Bereiche in der Ressourcen-Relationen Zeit/Dozent, Zeit/Raum und Zeit/Kurs freizugeben und daran anschließend werden die Constraints getestet, die auf diese Veranstaltung bezugnehmen. Bei den getesteten Constraints können zwei Fälle auftreten. Im ersten Fall wird das getestete Constraint nicht verletzt und im anderen Fall wird das getestete Constraint verletzt. Ein oder mehrere Constraints können zum Beispiel in einer Reihenfolgebeziehung verletzt worden sein. Die verletzten Constraints werden zurückgestellt und beim erneuten Einplanen der Veranstaltung wieder aktiviert. Nun wird die Veranstaltung als ungeplant markiert. Letztlich wird die ausgeplante Veranstaltung in den Listen für ungeplante Veranstaltungen ergänzt. Der Vorgang ist für alle Veranstaltungen relativ schnell abgeschlossen. Die entsprechenden Methoden zum Ausplanen einer Veranstaltung werden in der Schnittstellenbeschreibung näher beschrieben.

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 $Ressource \in \{Raum, Dozent, Zeit\}$, dabei wird die Menge der entsprechenden Ressource mit $R^{Ressource}$ betitelt. Folgendermaßen definieren sich die Prioritätsstufen:

- Alle möglichen Ressourcen: $A_{ed_i}^{Ressource}$
- Alle bevorzugten Ressourcen: $P_{ed_i}^{Ressource}$
- Alle Wünsche: $W_{ed_i}^{Ressource}$
- Es gilt: $W_{ed_i}^{Ressource} \subseteq P_{ed_i}^{Ressource} \subseteq A_{ed_i}^{Ressource} \subseteq R_{ed_i}^{Ressource}$

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 $W_{ed_i}^{Zeit} = P_{ed_i}^{Zeit}$ 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 Wunsches, der letzten bevorzugten sowie der letzten möglichen Ressourcen initialisiert werden. Es ergeben sich folgende Listen:

- Für $Ressource \in \{Dozent, Raum\}$:
  $PL_{ed_i}^{Ressource} = \left\{ pl_1, \ldots, pl_a, pl_{a+1}, \ldots, pl_b, pl_{b+1}, \ldots, pl_n \;\middle|\; \begin{array}{l} pl_1, \ldots, pl_a \in W_{ed_i}^{Ressource}; \\ pl_{a+1}, \ldots, pl_b \in P_{ed_i}^{Ressource}; \\ pl_{b+1}, \ldots, pl_n \in A_{ed_i}^{Ressource} \end{array} \right\};$
  $lastWish_{ed_i}^{Ressource} = a;$
  $lastPrefered_{ed_i}^{Ressource} = b;$
  $lastPossible_{ed_i}^{Ressource} = n;$ mit $1 \leq a \leq b \leq n; a, b, n \in N$

- Für $Ressource \in \{Zeit\}$:
  $PL_{ed_i}^{Ressource} = \{pl_1, \ldots, pl_n \,|pl_1, \ldots, pl_n \in W_{ed_i}^{Ressource}\};$
  $lastUsedWish_{ed_i}^{Ressource} = a;$ mit $1 \leq a \leq n; n \in N$

Auch hier ist wieder eine Einschränkung in Bezug auf die Zeit vorzunehmen. Es existieren hier keine positionsbestimmenden Werte. Es gibt nur eine Variable, die die Position der zuletzt verwendeten Wunschzeit angibt. Außerdem umfasst die Liste der Zeiten nur die Wunschzeiten. Zu jeder Veranstaltungsdefinition ist außerdem der zuletzt verwendete Tag hinterlegt, an dem eine Veranstaltung eingeplant wurde. Diese Variable soll $lastUsedDay_{ed_i}$ genannt werden.

Die Menge der Wochen wird $WEEKS$ genannt und deren einzelnen Elemente $week_1, week_2, \ldots, week_n \in WEEKS$. 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 also folgendes:

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

- 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:
  $$L_{week_i}^3 \subseteq L_{week_i}^1; l_1^3, \ldots, l_n^3 \in L_{week_i}^3$$

Dementsprechend wird die Menge der Tage $DAYS$ genannt und deren einzelne Elemente (day_1, day_2, ..., $day_n \in DAYS$). Den 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 Ressourcendimension minimiert werden. Die einzelnen Bedeutungen der Parameter werden im Anhang näher erläutert. Es gilt somit:

$$param \subseteq \left\{ \begin{array}{l} USELASTINSERTION, \\ SCHEDULETIME, \\ USEWISCHES, \\ USEPOSSIBILITIES, \\ USEALL, \\ PRIORITY\_TIME, \\ PRIORITY\_ROOM, \\ PRIORITY\_TEACHER, \\ LOG\_STANDARD, \\ LOG\_DETAILED \end{array} \right\}$$

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 Ermitteln 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, $ZeitraumID_v$, $KursID$, $Freelist$)
  Es wird eine Liste $Freelist$ mit freien Zeiträumen für den Kurs/-gruppe $KursID$ des Ereignisses e_i für einen Zeitraum $Zeitraum_v$ erstellt.

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

- **isfreeressource**($Ressource$, $((Beginn, Ende)$, **Result**)
  Es wird geprüft, ob die Ressource $Ressource$ im Zeitraum $(Beginn, Ende)$ verfügbar ist.

- **einplanen**($Zeitpunkt_v$, e_i)
  Plant zum Zeitpunkt $Zeitpunkt_v$ das Ereignis e_i ein.

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

- **!!freelist**(e_i, day_v, $Kurs$, FT_v^{Kurs})**!!**
  Ermittelt die Liste FT_v^{Kurs} mit freien Zeiträumen für den Kurs/-gruppe $Kurs$ des Ereignisses e_i an Tag day_v.

- **!!freelist**($\left(ft_v^{Kurs}\right)_x$, $pl_{highRes}^{highPrio}$, day_v, $Kurs$, $FT_v^{highPrio}$)**!!**
  Erstellt die Liste $FT_v^{highPrio}$ mit freien Zeitabschnitten, die die Schnittmenge von Zeitabschnitt $\left(ft_v^{Kurs}\right)_x$ des Kurses $Kurs$ und freien Zeiten der Ressource $pl_{highRes}^{highPrio}$ enthält.

- **!!freelist**($\left(ft_v^{highPrio}\right)_y$, $pl_{lowRes}^{lowPrio}$, day_v, y, $FT_v^{lowPrio}$)**!!**
  Erstellt die Liste $FT_v^{lowPrio}$ mit freien Zeitabschnitten, die die Schnittmenge von Zeitabschnitt $\left(ft_v^{highPrio}\right)_y$ und freien Zeiten der Ressource $pl_{lowRes}^{lowPrio}$ enthält.

- **!!einplanen**($\left(ft_v^{lowPrio}\right)_1$, e_i)**!!**
  Plant zum Zeitpunkt $\left(ft_v^{lowPrio}\right)_1$ das Ereignis e_i ein.

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

- **!!isfreeressource**($Kurs$, $(ft, ft + l)$, **Result**)**!!**
  Es wird geprüft, ob der Kurs $Kurs$ im Zeitraum $(ft, ft + l)$ verfügbar ist.

- **!!isfreeressource**($pl_{highRes}^{highPrio}$, $((ft, ft + l)$, **Result**)**!!**
  Es wird geprüft, ob die Ressource $pl_{highRes}^{highPrio}$ im Zeitraum $(ft, ft + l)$ verfügbar ist.

- **!!isfreeressource**($pl_{lowRes}^{lowPrio}$, $(ft, ft + l)$) **!!**
  Es wird geprüft, ob die Ressource $pl_{lowRes}^{lowPrio}$ im Zeitraum $(ft, ft + l)$ verfügbar ist.
- **!!einplanen**(ft, e_i)**!!**
  Plant 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 **diffn2D** stattfindet. Am Beispiel des Einplanens einer Veranstaltung innerhalb des Planungsalgorithmus **!!...!!** wird der prinzipielle Ablauf aller Methoden der Aufrufe der globalen Constraints diffn2D dargestellt.

einplanen(v, e_i)
{
 add(v, e_i, ft_v, pl^{Kurs}, pl^{Raum}, pl^{Dozent}, D_i, I^{Kurs}, I^{Raum}, I^{Dozent}, state, canOver)
  add(v, $e_{i,j}$)
   Look(ft_v, pl^{Kurs}, D_i, I^{Kurs}, state, canOver)
  add(v, $e_{i,j}$)
   Look(ft_v, pl^{Raum}, D_i, I^{Raum}, state, canOver)
  add(v, $e_{i,j}$)
   Look(ft_v, pl^{Dozent}, D_i, I^{Dozent}, state, canOver)
}

Beim Einplanen einer Veranstaltung e_i innerhalb des Planungsalgorithmus **einplanen**(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 pl^{Kurs}, pl^{Raum} und pl^{Dozent}, die Dauer D_i des Ereignisses e_i, die Größe der Ressourcennutzung I^{Kurs}, I^{Raum} und I^{Dozent}, der Status state und das Flag canOverwrite canOver ermittelt. Die Hauptmethode `add/12` hat die Form:

`add(`v`,` e_i`,` ft_v`,` pl^{Kurs}`,` pl^{Raum}`,` pl^{Dozent}`,` D_i`,` I^{Kurs}`,` I^{Raum}`,` I^{Dozent}`, state, 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`,` $e_{i,j}$`)` mit j $\in$ {Kurs/Zeit, Raum/Zeit, Dozent/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`,` $e_{i,j}$`)` Untermethoden der Relationen wird der Hauptmethode `add(`v`,` e_i`,` ft_v`,` pl^{Kurs}`,` pl^{Raum}`,` pl^{Dozent}`,` D_i`,` I^{Kurs}`,` I^{Raum}`,` I^{Dozent}`, state, 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 Ermitteln von freien Zeiträumen/ -abschnitten einer Ressource entsprechen im Grundschema der Vorgehensweise des Sequenzdiagramms zum Einplanens einer Veranstaltung aus Abbildung 5.10. Bei den Zwischenschritten 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 Aufruftiefe bis zum eigentlichen Aufruf notwendig macht. Dadurch können verschiedenste zusätzliche Funktionalitäten zur Bedingungsüberprüfung und Optimierung effektiv gekapselt werden.

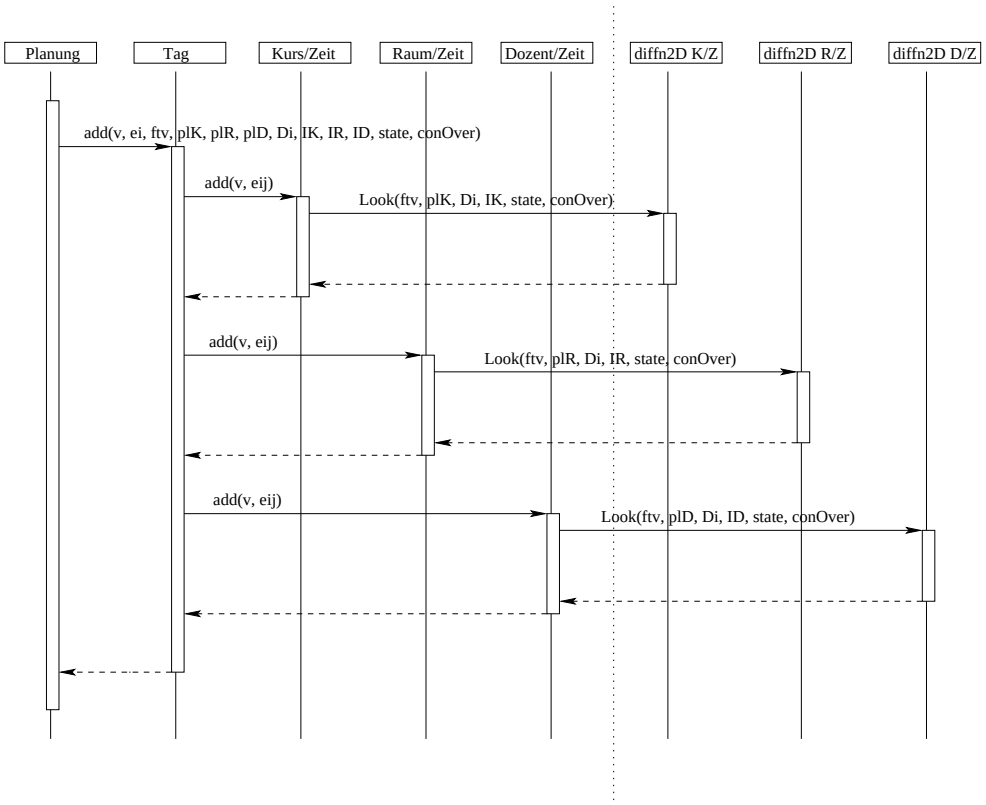

Abbildung 5.10: Ablauf des Einplanens einer Veranstaltung mit Zeit, Dozent und Raum

Die Initialisierung

Die ganze Planungskomponente untergliedert sich in verschiedene Teilkomponenten, die ihrerseits unterschiedliche Aufgaben beim Planungsablauf übernehmen. Diese müssen, bevor die eigentliche Planung stattfindet, initialisiert werden. Neben der Initialisierung werden vorhandene Veranstaltungen eingeplant oder als ungeplant gekennzeichnet. Dieser Vorgang findet für jeden Tag einzeln statt.

Die Funktion **initScheduling()** beinhaltet alle Unterfunktion für die Initialisierung der Planungskomponente. Die größeren Unterfunktionen **Initialisierung ()**, **sort-Def-Veran()**, **einplan-Veran-Versuch()** werden im Folgenden noch einzeln vorgestellt. Die kleinere Unterfunktion **Initialisiere-Planungskomponente (Wochen)** wird erst nach der Ausführung der vorgenannten Unterfunktionen ausgeführt. In dieser Unterfunktion wird die Verteilung der Veranstaltungen auf die einzelne Planungswochen grob vorgenommen. Es wird dabei nur die Kapazität der notwendigen Ressourcen ohne eine zeitliche Zuordnung geprüft. So wird sichergestellt, dass nur so viele Veranstaltungen einer Woche zugeordnet werden, wie von der Kapazität der benötigten Ressourcen maximal möglich wären. Es gibt auch Veranstaltungen die nur in bestimmten Wochen oder in einer bestimmten Reihenfolge mit anderen Veranstaltungen stattfinden dürfen. Diese Bedingungen werden auch bei der Wochenzuordnung beachtet. Es werden drei Listen für die Planung erzeugt, die Liste der geplanten Veranstaltungen, die Liste der ungeplanten Veranstaltungen, die 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 Listenerstellung erfolgt mit dem Wochenmanager, in dem die init-Funktion aufgerufen wird.
Mit der Funktion **Initialisiere-Planungskomponente(Blöcke)** wird hinterlegt, wie die einzelnen Veranstaltungen in Blöcken zusammengefasst werden sollen, und wie sie voneinander abhängen. Die

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

initScheduling()
{
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(Stundenüberwachung)
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 Speicherreservierungen 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 instanziiert, welche die zusätzlichen Restriktionen verarbeiten.
In der Funktion **ErstelleTageMenge(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 instanziiert und die eigentlichen Felder generiert.
Hierbei wird auf die zuvor instanziierten Konfiguratoren für die einzelnen Relationen und deren Restriktionen zurückgeriffen. Des Weiteren werden die Sperrzeiten 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 Sperrzeiten und andere Abhängigkeiten in den zweidimensionalen Feldern ergänzt
ErstelleTageMenge(DAYS) mit DAYS=$\{day_1, \ldots, day_n\}$
Erstelle-Veranstaltung-/Veranstaltungsdefinition-Relation $ed_i / \left\{e_1^{ed_i}, \ldots, e_k^{ed_i}\right\}$
i=1

```
  while (i ≤ |ED|?)
    {
      Vertausche Element in der Ressourcenliste zufällig, um lokale
      Häufung bei einer Ressource während der Planung zu vermeiden
    Hole-Liste PL_{ed_i}^{Ressource}; Ressource ∈{Raum, Dozent}
      Vertauschung der Elemente innerhalb der Teillisten von Wünschen, bevorzugten und
      möglichen Ressourcen
    Vertausche-zufällig-Elemente(PL_{ed_i}^{Ressource})
    i=i+1
    }
}
```

In der Funktion **sort-Def-Veran()** werden die Veranstaltungen sortiert nach der Zugehörigkeit zu einer 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 wird 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_1^{ed_j}, ..., 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 betroffnen Veranstaltungen nicht mit in den Plan aufgenommen. Diese Veranstaltungen werden dann erneut eingeplant.

einplan-Veran-Versuch()

```
{
  j=1
  while (j ≤ |ED|?)
    {
    Hole-Relation ed_j/{e_1^{ed_j}, ..., e_k^{ed_j}} der Definition ed_j
    i=1
    while (i ≤ k ?)
      {
      if (Veranstaltung e_i^{ed_j} bereits verplant?
        {
        Ermittle-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
```

}
else
{
Ergänze Veranstaltung $e_i^{ed_j}$ bei ungeplanten Veranstaltungen

}
Nehme nächste Veranstaltung: i=i+1
}
Nehme nächste Definition von Veranstaltungen: j=j+1
}

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

In der Funktion **Einplanen von Veranstaltung $e_i^{ed_j}$ 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 eingeplant ist. Es wird dafür der Tag (das Objekt des Tages) ermittelt, an dem die Veranstaltung eingeplant 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 Lockfunktion fehl schlägt, so wird das Einfügen der Veranstaltung rückgängig gemacht.
Die Übermittlung von inkonsistenten und falschen Planungsdaten kann zum Ausplanen einer schon eingeplanten Veranstaltung führen. Hier erfolgt eine Überprüfung der gegebenen Daten und nur die Daten 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 vorverteilt und enthalten auch die vorab interaktiv geplanten Veranstaltungen. Die Listen mit den verteilten, ungeplanten Veranstaltungen werden von der ersten bis zur letzten Woche des Planungshorizontes abgearbeitet 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 $lastUsedDay_{ed_i} = day_1$ und $lastUsedWish_{ed_i}^{Ressource} = 1$
Hole-Menge $WEEKS = \{week_1, \ldots, week_n\}$
i=1
while (i $\leq$ m) {
Hole $L_{week_i}^3 = \{l_1^3, \ldots, l_n^3\}$ von $week_i$
j=1
while (Soll fortgesetzt werden und j $\leq$ n?)
{
Versuche Veranstaltung l_j^3 einzuplanen: rufe Schedule(l_j^3, param) auf
Frage ab, ob Prozess abgebrochen werden soll
Nehme nächste ungeplante Veranstaltung: j=j+1
}

Nehme nächste Woche: i=i+1
}
Die erstmögliche Tage sind nun nicht mehr der erste Tag des Planungszeitraumes - setze die Variablen auf diesen zurück
Setze $lastUsedDay_{ed_i} = day_1$ und $lastUsedWish_{ed_i}^{Ressource}$=1
}

Das Einplanen einer Veranstaltung

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

schedule($e_{current}$, $param$)
{
if (Veranstaltung $e_{current}$ bereits eingeplant?)
Lösung gefunden

Ermittle Definition $ed_{current}$ von der Veranstaltung $e_{current}$

if (($param \cap \{SCHEDULETIME\} = \emptyset$) $\vee$ ($param \cap \{PRIORITY\_TIME\} \neq \emptyset$)?)
{
Wenn Wunschdozenten und Wunschräume vorhanden sind, dann gibt es auch bevorzugte Dozenten und Räume, dies wurde vorher sichergestellt
Hole $PL_{ed_{current}}^{Zeit} = \{pl_1, \ldots, pl_n \,|\, pl_1, \ldots, pl_n \in W_{ed_{current}}^{Zeit}\}$
i=$lastUsedWish_{ed_{current}}^{Zeit}$
while (i $\leq$ n ?)
{
if (pl_i^{Zeit} ist Zeitpunkt?)
{
Versuche zu dem Zeitpunkt einzuplanen
Breche ab, wenn Lösung gefunden
setze $lastUsedWish_{ed_{current}}^{Zeit} = i$
}
elseif (pl_i^{Zeit} ist Zeitspanne?)
{
Ermittle *Startzeit*, *Endzeit*
while (*Startzeit* $\leq$ *Endzeit* ?)
{
Versuche, zum Zeitpunkt Startzeit einzuplanen
Breche ab, wenn Lösung gefunden
setze $lastUsedWish_{ed_{current}}^{Zeit} = i$
Erhöhe *Startzeit*
}
}
Nimm nächste Wunschzeit: $i = i + 1$ }
}

Der Punkt wird erreicht, wenn die Wunschzeiten zu keinem Resultat geführt haben
if ($param \cap \{SCHEDULETIME\} = \emptyset$?)
Breche ab, ohne eine Lösung gefunden zu haben

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 $firstDay < lastUsedDay_{ed_{current}}$?)
{
erster Tag ist gleich dem Tag, an dem zuletzt
eine Veranstaltung dieses Typs eingeplant wurde
$firstDay = lastUsedDay_{ed_{current}}$
}
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 Möglichen, außerdem sind sie indizierbar
if ($param \cap \{PRIORITY_{ROOM}\} \neq \emptyset$?)
{
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
$lastUsedRessource_{ed_{current}}^{Ressource} = lastWish_{ed_{current}}^{Ressource}; Ressource \in \{Dozent, Raum\}$
}
elseif(($(param \cap \{USEPOSSIBILITIES\} \neq \emptyset) \wedge (param \cap \{USEALL\} = \emptyset)$)?)
{
Setze letztmögliche Ressourcen mit dem Index der letzten bevorzugten Ressource gleich
$lastUsedRessource_{ed_{current}}^{Ressource} = lastPrefered_{ed_{current}}^{Ressource}; Ressource \in \{Dozent, Raum\}$

}
else
{
Setze letztmögliche Ressourcen mit dem Index der letzten möglichen Ressource gleich
$lastUsedRessource_{ed_{current}}^{Ressource} = lastPossible_{ed_{current}}^{Ressource}; Ressource \in \{Dozent, Raum\}$
}

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 ($firstDay \leq day_o$?)
{
Ermittle den letzten Tag $dayOfWeek_{last}^{week_j}$ der aktuellen Woche $week_j$

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

Erstelle Liste DL^{week_j}
$dayOfWeek^{week_j} = dayOfWeek_{first}^{week_j}$

while ($firstDay \leq dayOfWeek_{last}^{week_j} \wedge firstDay \leq day_o$?)
{
Ermittle Variable $c_{firstDay}^{ed_{current}}$, die die Anzahl _
der identischen Veranstaltungen an dem Tag wiedergibt
Füge am Ende der Liste Tupel $dc_{firstDay}^{ed_{current}} = \left(firstDay, c_{firstDay}^{ed_{current}}\right)$ _
mit Tag und ermitteltem 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 ($i \leq \left|DL^{week_j}\right|$?)
{
Nehme-Tupel $\left(dc_k^{ed_{current}}\right)_i = \left(day_v, c_v^{ed_{current}}\right)$
!!freelist(e_i, day_v, $Kurs$, FT_v^{Kurs})**!!**
Ermittlt Liste FT_v^{Kurs} mit freien Zeiträumen für den _ Kurs/-gruppe der Veranstaltung an Tag day_v.
if ($\left|FT_v^{Kurs}\right| \neq 0$?)
{
Prüfe, ob es Zeitabschnitte gibt, an dem ein Dozent und Raum frei ist
$x = 1$
while ($x \leq \left|FT_v^{Kurs}\right|$?)
{
if(Zeitabschnitt ft_v^{Kurs} nicht lang genug für Veranstaltung ?)
Fahre fort mit dem nächsten Zeitabschnitt: $x = x + 1$

Probiere jede Ressource mit hoher Priorität

Erster Index von Element mit hoher Priorität ist 1

$highRes = 1$
while ($highRes \leq lastUsedRessource_{ed_{current}}^{highPrio}$?)
{
!!freelist($\left(ft_v^{Kurs}\right)_x$, $pl_{highRes}^{highPrio}$, day_v, $Kurs$, $FT_v^{highPrio}$**)!!**
Erstellt Liste $FT_v^{highPrio}$ **mit freien Zeitabschnitten, die die Schnittmenge von Zeitabschnitt** $\left(ft_v^{Kurs}\right)_x$ **und freien Zeiten der Ressource** $pl_{highRes}^{highPrio}$ **enthält.**
if ($\left|FT_v^{highPrio}\right| \neq 0$?)
{
Prüfe jeden Zeitabschnitt y und versuche, freie Ressource niedriger Priorität zu finden
$y = 1$
while($x \leq \left|FT_v^{highPrio}\right|$?)
{
Probiere jede Ressource mit niedriger Priorität
Erster Index von Element mit niedriger Priorität ist 1
$lowRes = 1$
while($lowRes \leq lastUsedRessource_{ed_{current}}^{lowPrio}$?)
{
Der Ort beider Ressourcen muss identisch sein
if($ort\left(pl_{highRes}^{highPrio}\right) = ort\left(pl_{lowRes}^{lowPrio}\right)$?)
{
!!freelist($\left(ft_v^{highPrio}\right)_y$, $pl_{lowRes}^{lowPrio}$, day_v, y, $FT_v^{lowPrio}$**)!!**
Erstellt Liste $FT_v^{lowPrio}$ **mit freien Zeitabschnitten, die die Schnittmenge von Zeitabschnitt** $\left(ft_v^{highPrio}\right)_y$ **und freien Zeiten der Ressource** $pl_{lowRes}^{lowPrio}$ **enthält.**
if($\left|FT_v^{lowPrio}\right| \neq 0$?)
{
Zeitpunkt zum Einplanen gefunden
Erster Wert in der List entspricht dem Begin der Veranstaltung
!!einplanen($\left(ft_v^{lowPrio}\right)_1$, e_i**)!!**
Plant zum Zeitpunkt $\left(ft_v^{lowPrio}\right)_1$ **ein.**
Lösung gefunden
}
}
Nehme nächste Ressource: $lowRes = lowRes + 1$
}
Nehme nächsten Zeitabschnitt: $y = y + 1$
}
}
Nehme nächste Ressource: $highRes = highRes + 1$
}
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 ($Tagday_{current}nichtgefunden?$)
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 Ressourcenlisten sind vorsortiert: erst kommen die Wünsche, dann die
Bevorzugten und als letztes die Möglichen, ausserdem sind sie indizierbar
if ($param \cap \{PRIORITY_ROOM\} \neq \emptyset$?)
{
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
$lastUsedRessource_{ed_{current}}^{Ressource} = lastWish_{ed_{current}}^{Ressource}; Ressource \in \{Dozent, Raum\}$
}
elseif(($(param \cap \{USEPOSSIBILITIES\} \neq \emptyset) \wedge (param \cap \{USEALL\} = \emptyset)$) ?)
{
Setze letztmögliche Ressourcen mit dem Index der letzten Bevorzugten gleich
$lastUsedRessource_{ed_{current}}^{Ressource} = lastPrefered_{ed_{current}}^{Ressource}; Ressource \in \{Dozent, Raum\}$
}
else
{
Setze letztmögliche Ressourcen mit dem Index der letzten Möglichen gleich
$lastUsedRessource_{ed_{current}}^{Ressource} = lastPossible_{ed_{current}}^{Ressource}; Ressource \in \{Dozent, Raum\}$
}

Schleife der eigentlichen Planung
Probiere jede Ressource mit hoher Priorität

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

while($highRes \leq lastUsedRessource_{ed_{current}}^{highPrio}$?)
{
if(**!!isfreeressource(**$pl_{highRes}^{highPrio}$**,** $((ft, ft+l)$**, Result)!!**
(Ressource $pl_{highRes}^{highPrio}$ **im Zeitraum** $(ft, ft+l)$ **verfügbar ?)**
{
Probiere jede Ressource mit niedriger Priorität
Erster Index von Element mit niedriger Priorität ist 1
$lowRes$?1
while($lowRes \leq lastUsedRessource_{ed_{current}}^{lowPrio}$?)
{
Der Ort beider Ressourcen muss identisch sein
if($ort\left(pl_{highRes}^{highPrio}\right) = ort\left(pl_{lowRes}^{lowPrio}\right)$?)
{
if(**!!isfreeressource(**$pl_{lowRes}^{lowPrio}$**,** $(ft, ft+l)$**) !!**
(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 gefüllt, 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

annehmen, ohne sich mit einer anderen Teilliste zu vermischen. In der Teilliste der Wünsche behalten diese ihre Ordnung bei, da die Wünsche initial priorisiert sind und damit eine gewünschte Reihenfolge haben. Mit der Umsortierung an eine zufällige Position in zwei der drei Teillisten ohne Vermischung von Wünschen, bevorzugten und möglichen Ressourcen wird der beschriebene Effekt reduziert.

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 Engpassanalyse 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 haben 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 einsortiert. Je niedriger der Wert der Priorität ist, desto weiter vorn wird das Element einsortiert. 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 hoch gesetzt (zum Beispiel um die Hälfte des Wertebereichs der Prioritäten), so dass diese Veranstaltungen höher priorisiert sind als alle anderen. Dies ergibt dann folgendes:

$$prio^{ed_i} = min\left(lastPossible_{ed_i}^{Dozent}, lastPossible_{ed_i}^{Raum}\right) + \begin{cases} PL_{ed_i}^{Zeit} \neq \emptyset : 30000 \\ PL_{ed_i}^{Zeit} = \emptyset : 0 \end{cases}$$

Sortierung der Tage einer Woche innerhalb der Planung

Ziel der Planung ist es, Veranstaltung so früh wie möglich einzuplanen. Leider hat dies auch den unangenehmen Nebeneffekt, dass es zu lokalen Häufungen von gleichen Veranstaltungen kommen kann. Dies resultiert unter anderem auch daraus, dass meist in den Listen der ungeplanten Veranstaltungen gleiche Veranstaltungen direkt aufeinander folgen. Diese Häufung kann zwar durchaus Sinn haben, aber gerade an Schulen ist damit die Planungsqualität niedriger.
Um dem Problem vorzubeugen, wird in der automatischen Planung wochenweise 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 $c_{ed_{current}}^{day_i}$ kleiner oder größer als $MaxEventsPerDay$ ist, werden die Tage entweder in der Reihenfolge der Tage oder in der Reihenfolge der Anzahl der Veranstaltungen sortiert. lhs und rhs sind Tupel und definiert als $lhs_i^{ed_{current}} = \left(\begin{array}{cc} day_i & c_i^{ed_{current}} \end{array} \right)$ und $rhs_j^{ed_{current}} = \left(\begin{array}{cc} day_j & c_j^{ed_{current}} \end{array} \right)$. Dann gilt also:

$$\left(\begin{array}{l} \left(lhs_i^{ed_{current}}\right)_2 < MaxEventsPerDay \wedge \\ \left(\begin{array}{l} \left(rhs_j^{ed_{current}}\right)_2 \geq MaxEventsPerDay \vee \\ \left(lhs_i^{ed_{current}}\right)_1 < \left(rhs_j^{ed_{current}}\right)_2 \end{array} \right) \end{array} \right) \vee$$

$$\left(\begin{array}{l} \left(lhs_i^{ed_{current}}\right)_2 \geq MaxEventsPerDay \wedge \\ \left(\begin{array}{l} \left(lhs_i^{ed_{current}}\right)_2 < \left(rhs_j^{ed_{current}}\right)_2 \vee \\ \left(\begin{array}{l} \left(lhs_i^{ed_{current}}\right)_2 = \left(rhs_j^{ed_{current}}\right)_2 \wedge \\ \left(lhs_i^{ed_{current}}\right)_1 < \left(rhs_j^{ed_{current}}\right)_2 \end{array} \right) \end{array} \right) \end{array} \right)$$

$$\rightarrow lhs_i^{ed_{current}} < rhs_j^{ed_{current}}$$

Damit wurde erreicht, dass die Veranstaltungen so früh wie möglich eingeplant werden können (Reduzierung der Freistunden) und keine lokalen Häufungen von gleichen Veranstaltungen entstehen.

Kapitel 6

Modellierung von Stundenplanungsproblemen

6.1 Methoden:

In dem folgenden Abschnitt soll die Modellierung von n-dimensionalen Ressourcenplanungsproblemen mit n-1 zweidimensionalen Räumen erläutert werden. Für die Lösung von Ressourcenproblemen in zweidimensionalen Räumen gibt es effektive Algorithmen, um in kurzer Zeit optimale Positionierungen und Platzierungen von Körpern im zweidimensionalen Raum vorzunehmen. Mit diesen Algorithmen können zweidimensionale Platzierungsproblem gelöst werden. Sie sind zum Beispiel im globalen Diffn - Constraint diffn() der französichen Firma Cosytec integriert.
Zur Lösung dreidimensionaler Platzierungsproblem gibt es derzeit einen Ansatz, der in der Literatur veröffentlicht [4] wurde. Im globalen Diffn - Constraint (diffn()) der französischen Firma Cosytec in der dreidimensionalen Version ist ein Algorithmus integriert, der den Unterschied in einer der drei Dimensionen als Unterschied bei der Positionierung und damit der überlappungsfreien Positionierung des Körpers im dreidimensionalen Raum akzeptiert (siehe Abbildung 6.1).

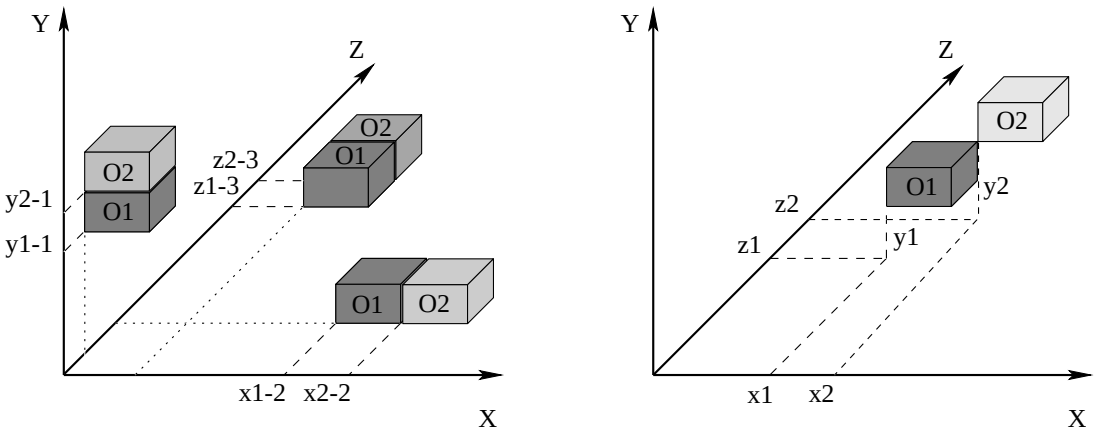

Abbildung 6.1: Eindimensionale und dreidimensionale Verschiedenheit beim Diffn - Constraint

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 zweidimensionale 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 (xi, yi, zi) 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 Quaders(Würfels) umfasst alle Außenflächen des Körpers.
Der Ursprungspunkt eines Objekt im dreidimensionalen Raum kann auch mit Punkten (P1, P2) auf den zweidimensionalen Hüllenflächen beschrieben werden. Auf jeder zweidimensionalen Hüllenfläche benötigt man zwei Koordinaten ((xi, zi) oder (yi, zi)), um einen Punkt zu beschreiben. Auf den sechs Hüllenflächen existieren dann sechs Punkte, auf jeder Hüllenfläche ein Punkt, mit denen der Ursprungspunkt eines Objektes im dreidimensionalen Raum eindeutig definiert werden kann. Es werden nicht alle sechs Hüllenflächen benötigt, da gegenüberliegende Hüllenflächen gleichgroß sind. Um den Ursprungspunkt eines Objektes im dreidimensionalen Raum mit den Punkten auf den zweidimensionalen Hüllenflächen eindeutig beschreiben zu können, benötigt man nur drei Flächen der Hülle des Köpers mit jeweils einem Punkt mit zwei Koordinaten auf der zweidimensionalen Hüllenfläche. In der Abbildung 6.2 wird dieses Vorgehen zur Positionsbestimmung des Ursprungs von Objekten veranschaulicht. Die drei Dimensionen werden mit X, Y und Z bezeichnet.

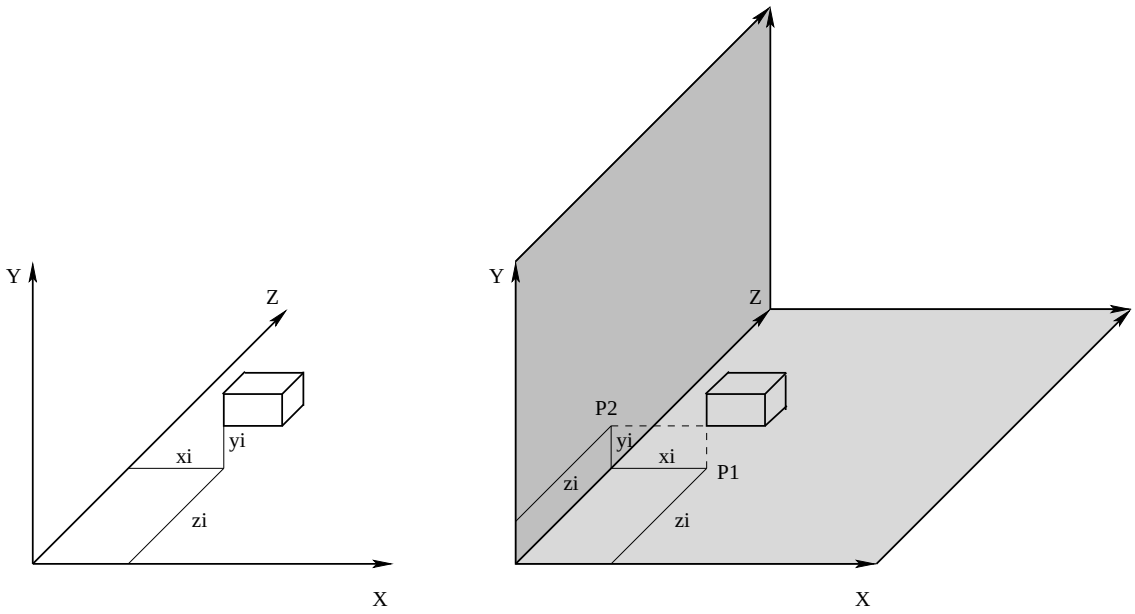

Abbildung 6.2: Ressourcendimensionen X, Y und Z

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

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

Die reduzierte Hülle HR(X,Y,Z) ergibt sich aus der Hülle H(X,Y,Z), wenn die einander gleichen Flächen entfernt werden.

$$HR(X,Y,Z) = \{X \text{ x } Y,\ Y \text{ x } Z,\ X \text{ x } Z\}$$

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

$$HRV(X,Y,Z) = \{X \text{ x } Y,\ X \text{ x } Z\}$$

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

Für den Beweis benötigt man Verfahren aus der darstellenden Geometrie. Die darstellende Geometrie untersucht und verwendet Abbildungen des dreidimensionalen Raumes auf ein ebenes Zeichenfeld. Um die konstruktiven Methoden der Planimetrie übernehmen zu können, benutzt man Abbildungsverfahren, bei denen Geraden des Raumes auch Geraden im Zeichenfeld entsprechen. Dies nennt man Projektion.

Eine der benutzten Projektionen ist die **Normalprojektion** oder **Orthogonalprojektion**. Bei der Abbildung der Normalprojektion verlaufen die zueinander parallelen Projektionsstrahlen senkrecht zur Bildebene E. Die stark reduzierte Anschaulichkeit wird auf zwei Wegen kompensiert.
Bei dem ersten Weg beziffert man markante Punkte oder Linien des dargestellten Objekts mit den Höhenknoten 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 Projektionseinrichtungen und die Bildebenen, welche 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 $E1*$ und $E2$ 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 `erstprojizierendes Parallelstrahlenbündel` s1 senkrecht zu $E1*$ auf $E1*$ und ein `zweitprojizierendes Parallelstrahlenbündel` s2 senkrecht zu $E2$ auf $E2$ abgebildet. Es entstehen auf diese Weise in zwei zunächst aufeinander senkrecht stehenden Ebenen zwei Normalprojektionen von einem Gegenstand. In der Ebene $E1*$ liegt der `Grundriss` und in der Ebene $E2$ liegt der `Aufriss`. Eine Konvention besteht darin, die Grundrisstafel horizontal und die Aufrisstafel vertikal anzunehmen. Für die weitere Bearbeitung legt man die Aufrisstafel in das Zeichenfeld und dreht anschließend die Grundrisstafel um die waagerechte `Rissachse` x_{12} in die vorgelegene Ebene. Nach dem Aufdrehen der Bildebenen in das Zeichenfeld kommt der Aufriss des im ersten Quadranten angebrachten Körpers über und der Grundiss unter der Rissachse zu liegen. Der Grund- und Aufriss eines Punktes P liegen auf einer zur Rissachse senkrechten Geraden. Diese wird als `Ordnungslinie` oder `Ordner` bezeichnet. Der Grundriss P' und der Aufriss P'' eines Punktes P befinden sich in `Mongescher Lage`, die auf den wissenschaftlichen Begründer 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 Projektionsstrahl` 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 Bildebenen [80].

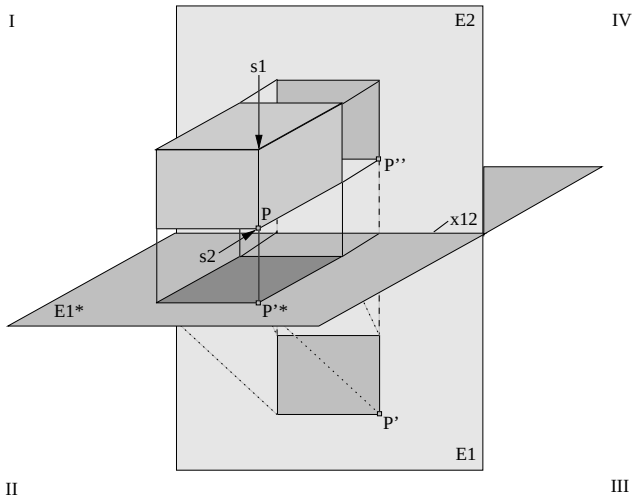

Abbildung 6.3: Zweitafelprojektion eines räumlichen Objekts in den zugeordneten Normalrissen

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.

Bezogen auf einen Quader oder Würfel bedeutet dies: liegt ein Körper so zu den Bildebenen, dass zwei Körperseiten zu diesen parallel sind, so werden diese zwei Seiten kongruent auf Grund- und Aufriss abgebildet. Nach der Aussage über die Zweitafelprojektion wird der Körper durch diese Abbildung vollständig beschrieben.

Zusammenfassung: Liegt nun ein Körper in einem dreiachsigen 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.

Satz: Seien c $\in Z$ 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:

$$(A(c) \wedge \forall(n \in Z(c))\ (A(n) \rightarrow A(n+1))) \rightarrow \forall(n \in Z(c))\ A(n).$$

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].

Beweis: Die Anzahl der Dimensionen wird mit AnR bezeichnet und die Anzahl der zweidimensionalen Flächen wird mit AnHRV bezeichnet.

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

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 entsprach, wird ein n+1 dimensionaler Körper also vollständig durch n Bildebenen dargestellt.

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

Induktionsschluss:
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:

$$H(V,X,Y,Z) = \{V \times X,\ X \times Y,\ Y \times Z,\ V \times Y,\ V \times Z,\ V \times X,\ X \times Y,\ Y \times Z,\ V \times Y,\ V \times Z,\ X \times Z,\ X \times Z\}$$
$$HR(V,X,Y,Z) = \{V \times X,\ X \times Y,\ Y \times Z,\ V \times Y,\ V \times Z,\ X \times Z\}$$
$$HRV(V,X,Y,Z) = \{V \times X,\ V \times Y,\ V \times Z\}$$

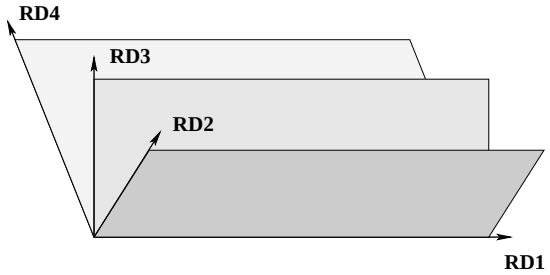

Abbildung 6.4: Visualisierung der zweidimensionalen Flächen für den vierdimensionalen Raum

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 dem Beispielproblem der Erstellung von Stundenplänen für Weiterbildungseinrichtungen handelt es sich um ein vierdimensionales Ressourcenplanungsproblem. Es gibt die vier Ressourcendimensionen Zeit, Dozent, Raum und Kurs. Die Ressourcendimension Zeit ist die Ressorcendimension, von der die anderen Ressourcendimensionen Dozent, Raum und Kurs abhängen, da die Zuordnung eines Dozents zu einem Raum und zu einem Kurs immer von der Zeit bedingt ist. Die Ressourcendimension Zeit wird im Folgenden mit RD1 bezeichnet, die Ressourcendimension Dozent mit RD2, die Ressourcendimension Raum mit RD3 und die Ressourcendimension Kurs mit RD4. Die Ressourcendimension Kurs (RD4) kann als eine kombinierte oder n-fach aufgelöste Ressourcendimensionen modelliert werden. Die Form der Modellierung ist abhängig von dem zur Realisierung ausgewählten Constraintsystem. Zum Beispiel ist die Propagation beim globalen Constraint diffn() vom Constraintsystem CHIP bei mehr als zwei Dimensionen sehr gering. Deshalb werden bei CHIP vorwiegend zweidimensionale Modellierungen genutzt und die einzelnen Kurse und deren Gruppen in mehreren Dimensionen modelliert (n-fach aufgelöste Ressourcendimension).
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 mitte) 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) zusammensetzt, mit n $\in$ 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 einführt, 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 Abbindung 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.

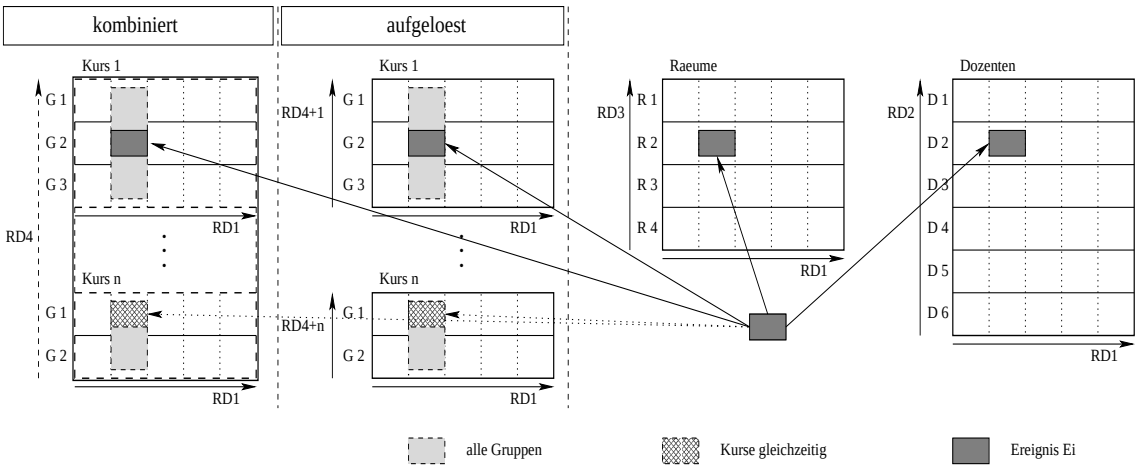

Abbildung 6.5: Ressourcendimensionen RD1, RD2, RD3, RD4 und RD4+n

Für die Modellierung des Stundenplanungsproblems für Weiterbildungseinrichtungen wird die Variante der aufgelösten Kurs-Ressourcendimension bei der Verwendung von herkömmlichen Constraintsystem, wie CHIP von der französischen Firma Cosytec, genutzt. Dies ist notwendig, weil die Propagation in den globalen Constraints diffn() mit mehr als zwei Dimensionen sehr gering ist. Bei der Realisierung mit dem neuen Constraintlöser CS_{fd-MR} wird die kombinierte Variante der Kurs-Ressourcendimension RD4 angewandt. Die kombinierte Variante der Kurs-Ressourcendimension wird bei dem Constraintlöser CS_{fd-MR} zur Reduzierung des Speicherverbrauchs und zur Beschleunigung des Lösungsprozesses eingesetzt.
Im folgenden Abschnitt wird die Modellierung des Stundenplanungsproblems für Weiterbildungseinrichtungen bei Anwendung der Constraints zur Definition des Constraintnetzes 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 Constraintlöser CS_{fd-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

Zur Abbildung von Ressourcen mit ihren Wertebereichen der Ressourcendomänen werden Domänenvariablen benutzt. Die Domänenvariable definiert in ihrer Zuordnung von realen Werten zu diskreten Werten die Genauigkeit der Abbildung der realen Umwelt. Im Bereich der Stundenplanung ist eine minutengenaue Abbildung für die meisten Abbildungen ausreichend, da der Beginn und die Dauer von Lehrveranstaltungen, Pausen und Wegezeiten im Allgemeinen minutengenau angegeben werden. Hingegen ist im Verkehrsbereich eine sekundengenau Abbildung unablässig, weil die Geschwindigkeiten und damit die Zustandwechsel viel schneller sind.
Der Planungshorizont erstreckt sich bei Stundenplänen in Schulen über eine Woche und in Weiterbildungseinrichtungen meist über ein bis zwei Jahre. Bei einer minutengenauen 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

= 1068480 diskrete Werte benutzt. Bei einer sekundengenauen Modellierung im Verkehrsbereich ergeben sich bei einer Abbildung eines Tages 60 Sekunden * 60 Minuten * 24 Stunden = 86400 diskrete Einheiten und für eine Woche 86400 Tagessekunden * 7 Tage = 604800 diskrete Einheiten. Die Anzahl der notwendigen diskreten Einheiten zur Abbildung einer Wochen bei sekundengenauer Abbildung ist um den Faktor 60 größer als bei minutengenauer Abbildung im Bereich der Stundenplanung. Dies lässt bei gleichem Speicherverbrauch einen längeren Planungshorizont bei der minutengenauen als bei der sekundengenauen Planung zu.
Zur Modellierung des Stundenplanungsproblems wird eine minutengenaue Abbildung gewählt, um die kurzen Pausen von ca. 5 Minuten und die geringen Wegezeiten zwischen den verschiedenen Veranstaltungsorten abbilden zu können. Die kleinste diskrete Einheit der Domäne sind die Minuten, die zu Stunden aggregiert und danach zu Wochentagen und zu Wochen zusammengefasst werden. Diese Domäne eignet sich zum gleichzeitigen Planen von Vorlesungen und Praktika, da sich kurze Veranstaltungen mit zum Beispiel 45 Minuten und auch mehrtägige Praktika abbilden lassen. Die zu planenden Veranstaltungen können eine unterschiedliche Länge haben, womit sich auch Doppel- oder Mehrfachstunden mit und ohne Pause modellieren lassen. In dieser Domäne werden die Pausen durch Blockierungen von Zeiträumen modelliert.

6.3.2 Einschränkung der "Nichtzeiten" - Feiertage, Urlaub, Pausen

In Stundenplänen, die über mehrere Wochen oder Jahre reichen, muss auch der unterschiedlichen Unterrichtsdauer in den einzelnen Wochen Rechnung getragen werden, die durch Feiertage und Urlaubstage verursacht werden. An diesen Tagen findet kein Unterricht statt. Diese Bedingung kann mit zwei Ansätzen modelliert werden. Die Zeiträume der Feier- und Urlaubstage können durch Pseudoveranstaltungen blockiert werden oder die Werte werden aus der Domäne entfernt.
Im Programmiersystem CHIP erfolgt die Modellierung der Feiertage durch die Entfernung der Domänenwerte. Hierfür kann das Notin-Constraint verwendet werden. Man gibt bei diesem Constraint die betroffene Domänenvariable, den Start- und den Endwert an. Die zwischen diesen beiden Werten liegenden Werte der Domäne werden dann entfernt. Die andere Möglichkeit der Blockierung der Feiertags- und Urlaubszeitabschnitte lässt sich mit dem diffn()-Constraint durch das Einfügen von zusätzlichen Rechtecken als Pseudoveranstaltungen realisieren.
Beide Varianten unterscheiden sich nicht hinsichtlich ihrer Funktionalität. Wohingegen es Unterschiede in der Nutzung des benötigen Speichers für die gleiche Aufgabe gibt (siehe folgende Tabelle). Als Testbeispiel diente die Domäne über ein Jahr und die entsprechenden Feiertag und dreißig Urlaubstage.

Speicherverbrauch bei Modellierung mit Diffn-Constraint und Notin-Constraint

diffn()- gegen notin()-Constraint ausgetauscht, für das Beispiel getestet				
Kriterium	Constraint		Einsparung notin() zu diffn()	
	diffn()	notin()	absolut	in Prozent
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
Gesamt Speicher	6195 kB	1741 kB	- 4454 kB	- 72
Prozesszeit	480 ms	460 ms	- 20 ms	- 4

Aus der Tabelle ist ersichtlich, dass ein Entnehmen von Domänenwerten mit dem notin()-Constraint weniger Speicher verbraucht als das Einfügen zusätzlicher Pseudoveranstaltungen für die Abbildung der Feier- und Urlaubstage. Für das Entfernen von Domänenwerten werden nur 7 % des globalen Speicher benötigt, im Vergleich mit dem Hinzufügen von Pseudoveranstaltungen für das gleiche Planungsproblem. Auf den gesamten Speicherverbrauch bezogen werden nur noch 28 % benötigt, womit eine Ersparnis von 72 % erreicht wird. Bei gleichem zur Verfügung stehenden Speicherplatz können bei der Modellierung der Nichtzeiten mit dem notin()-Constraint etwa 3 mal größere Planungsprobleme

gelöst werden. Aufgrund dieser Tatsache wird die Modellierung der Feier- und Urlaubstage im Programmiersystem CHIP mit notin()-Constraints umgesetzt.
Im neuen Constraintlöser CS_{fd-MR} stehen für die Modellierung der Feiertage ein notin()-Constraint und das globale Constraint diffn2D() zur Verfügung. In diesem System ist aber die Domänenvariable anders realisiert. In der Domänenvariable werden nur die Bereiche angeben, in denen Domänenwerte vorhanden sind. Ein Entfernen von Domänenwerten aus der Domäne führt zur Aufteilung des zusammenhängenden Domänenbereichs, wodurch sich der Speicherplatzbedarf der Domäne vergrößert. Das Hinzufügen von Pseudoveranstaltungen wird auch als ein zusammenhängender Bereich mit Anfangs- und Endwert in einer Datenstruktur gehalten. Somit fallen für die beide Modellierungen die gleichen Speicheraufwände an. Zur Sicherung der Vergleichbarkeit wird hier die Modellierung mit notin()-Constraints gewählt.

Die Pausenzeiten können genauso wie die Feier- und Urlaubstage modelliert werden. Wenn keine Pausenaufsicht zu planen ist, können die Pausenzeiten aus der Domäne mit dem notin()-Constraint entfernt werden. In dem Fall, dass in den großen Hofpausen oder den Freistunden beaufsichtigt werden muss, bietet sich die Variante des Einlegens von Pseudoveranstaltungen an. Die Pseudoveranstaltung wird dann zu einer Veranstaltung; die von allen Kursen besucht werden kann und zu der als Aufsichtspersonal ein oder mehrere Personen zugeordnet sind.

6.3.3 Überschneidungsfreiheit

In Stundenplänen ist immer zu gewährleisten, dass sich die Stunden nicht überschneiden, die Dozenten und Räume nicht doppelt oder überschneidend belegt sind. Diese Überschneidungsfreiheit wird durch das diffn()-Constraint in der Programmiersprache CHIP und durch das globale Constraint diffn2D() im neuen Constraintlöser CS_{fd-MR} gewährleistet. In beiden Ausprägungen der globalen Constraints wird durch das Einlegen von Rechtecken in zweidimensionale Platzierungsräume die Überlappungsfreiheit gesichert. Beim Verletzen der Überlappungsfreiheit wird das globale Constraint und die damit ausgedrückte Bedingung nicht eingehalten. Eine Überlappung ist gegeben, wenn sich zwei Rechtecke an irgendeiner Stelle überschneiden.

6.3.4 Reihenfolgebeziehungen

Die Modellierung der Lehrpläne mit den darin enthaltenen spezifischen Abfolgen von Veranstaltungen, beginnend mit der Basisveranstaltung, gefolgt von der Aufbauveranstaltung und endend mit der Spezialisierungsveranstaltung, wird mit Hilfe des Reihenfolgeconstraints RC1 realisiert.
Mit dem Reihenfolgeconstraint wird die festgelegte Abfolge der Veranstaltung gesichert. Im automatischen Planungsmodus wird dieses Constraint immer eingehalten, wohingegen im interaktiven Modus die Möglichkeit besteht, auch einzelnen Veranstaltungen mit einander zu vertauschen. Diese Möglichkeit der Vertauschung ist im Fall von Krankheit und Vertretung notwendig und auch von den Bildungsträgern gefordert. Wenn dieses nicht nutzbar wäre, müssten erst alle nachfolgenden Veranstaltungen einzeln verschoben werden, um einen entsprechenden Freiraum zu schaffen.
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 Constraintlöser CS_{fd-MR} ist das Reihenfolgeconstraint auch vorhanden.

Kapitel 7

Anwendungsbeispiel

7.1 Stundenplanung für Kurse

Die Stundenpläne von Weiterbildungs- und schulischen Einrichtungen unterliegen vielerlei Restriktionen, die je nach Institution stark variieren und individuelle Anpassungen verlangen. Es müssen Dozenten und Räume auf Veranstaltungen aufgeteilt werden, so dass es weder bei den entsprechenden Kursen, den Dozenten oder den Räumen zu zeitlichen Überschneidungen kommt. Dazu kommen noch viele andere Randgrößen, die beachtet werden müssen. Das Erstellen funktionsfähiger Pläne erstreckt sich in der Praxis über einen längeren Zeitraum. Um diesen Zeitaufwand und die damit verbundene mögliche Fehleranfälligkeit zu reduzieren, soll ein Planungssystem zur Stundenplanerstellung verwendet werden. Aus der Sicht des Anwenders, dem Stundenplaner, soll das Planungssystem folgende Eigenschaften besitzen und nachfolgenden Anforderungen genügen:
Das Planungssystem soll mit Hilfe geeigneter Verfahren konsistente Pläne mit all ihren Restriktionen in möglichst kurzer Zeit automatisch erstellen können. Der Anwender soll nur noch gezwungen sein, all die Restriktionen zu definieren, die der automatischen Planung zu Grunde liegen. Diese Phase der Definition der Problemdomäne umfasst die Eingabe der Ressourcen, wie Orte, Dozenten, Räume, Fächer und Kurse und den darauf basierenden Kursgruppen, sowie letztendlich auch die Definition der Veranstaltungen. Zahlreiche Kombinationsmöglichkeiten verschiedenster Restriktionen erlauben die Definition unterschiedlichster Pläne zahlreicher Institutionen. So werden neben schulspezifischen Dingen, wie Blockstunden, Kursgruppen und Teilungsunterricht auch weiterbildungsspezifische Dinge wie Praktika und Abhängigkeitsbeziehungen in Bezug auf die Reihenfolge von Veranstaltungen unterstützt. Ziel ist, es neben hoher Variabilität auch die Kombination von interaktiver und automatischer Planung zu ermöglichen, so dass der Anwender in der Lage ist, den Planungsprozess in der Weise zu manipulieren, dass er bestimmte Randbedingungen von Hand setzen kann. Im Folgenden wird vor allem auf die automatische Planung eingegangen.
Die eigentliche Planung soll mit Hilfe constraintbasierter Verfahren erfolgen. Die Problemdomäne umfasst folgende vier Dimensionen: Räume, Dozenten, Zeit und Kurse/Kursgruppen. Als Grundlage der Planung dienen die Definitionen der Veranstaltung. Veranstaltungen sind einem bestimmten Kurs oder einer Kursgruppe zugeordnet. Der Anwender kann außerdem Wünsche und bevorzugte Ressourcen bezüglich der Räume, Dozenten und der Zeit festlegen, die berücksichtigt werden sollen.
Die Veranstaltung besitzt somit vier Hauptabhängigkeiten. Die Abhängigkeit von der Zeit, dem Raum, dem Dozenten und der Gruppe des Kurses oder dem Kurs direkt. Die beteiligten Ressourcen stehen auch untereinander in weiteren Abhängigkeiten. So muss zum Beispiel der Dozent auch das Fach unterrichten könne, zu welchem die Veranstaltung gehört. In der Abbildung 7.1 sind die Abhängigkeiten innerhalb des Planungsproblems dargestellt.
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 undenkbar. Dort können maximal Doppelstunden gebildet werden. Außerdem sollte die

Anzahl der Freistunden minimal gehalten werden.

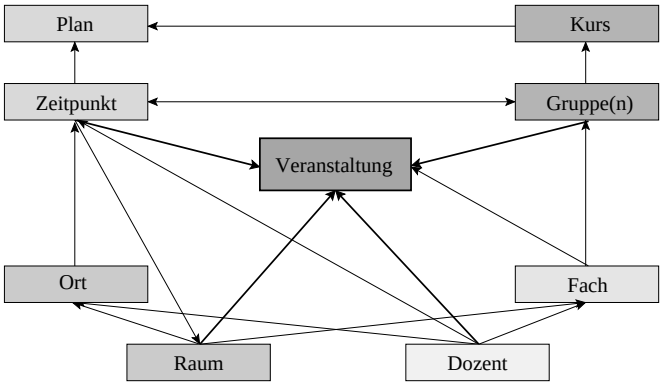

Abbildung 7.1: Graphische Interpretation der Abhängigkeiten

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.
Die einzelnen Dimensionen haben spezielle Eigenschaften und Grenzen. Die Zeitdimension ist durch ihr Maximum begrenzt. Die Ressourcendimension der Räume stellt einen Engpass da. Es sind nur eine begrenzte Anzahl von Räumen vorhanden, die unterschiedlich ausgestattet sind. Die Ressourcendimension der Dozenten stellt in dem Beispiel zur Planung von Kursen an einer Weiterbildungseinrichtung keinen Engpass da. Es sind genug Dozenten für die einzelnen Veranstaltungen auf dem Markt vorhanden, die je nach Bedarf rechtzeitig akquiriert werden können. Die Ressourcendimension der Gruppen der Kurse oder dem Kurs selbst ist eine Organisationsressource. Die Teilnehmer der Kurse werden den Kursen und innerhalb der Kurse einzelnen Gruppen zugeordnet.

Problembeschreibung

Ein Stundenplan ist eine Zuordnung von Veranstaltungen zu Zeitpunkten bzw. Zeitintervallen, Räumen, Dozenten und Gruppen. Bei der Zuordnung zu den Zeitpunkten werden problemspezifische Bedingungen beachtet. Solche Bedingung sind zum Beispiel:

- Pausen zwischen zwei Veranstaltungen,
- Wegezeiten 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 verschieden 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 weiche 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 Constraintlöser 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 Performanz 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 Constraintlöser für Multiressourcenprobleme CS_{fd-MR} aus dem gleichnamigen Kapitel in dieser Arbeit. Im Planungssystem Kursplan 1.0 wurde die Benutzeroberfläche in der Programmiersprache .NET und der Constraintlöser 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 Constraintlöser 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

Die Randbedingungen, die harten und weichen Constraints des Planungsproblems werden mit den Mitteln der Constraintverarbeitung modelliert. Die beiden Systeme beinhalten für Stundenplanungs-, Zuordnungs- und die Ressourcenprobleme in der Funktion gleichmächtige Constraints. In den folgenden Tabellen wird dargestellt, welches Constraint in dem jeweiligen System zur Abbildung der einzelnen Randbedingungen und Constraints benutzt wurde.

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 Domänenvariablen. Entweder beinhaltet die Domänenvariable nur einen Wert, wenn eine eindeutige Wertzuordnung erfolgen soll, zum Beispiel bei 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		
Bedingung	Constraint ConTime 1.0	Constraint Kursplan 1.0
R1	Domänenvariablen	Domänenvariablen
R2	Domänenvariablen	Domänenvariablen
R3	Domänenvariable mit einem Wert	Domänenvariable mit einem Wert
R4	Domänenvariablen	Domänenvariablen
R5	Domänenvariablen	Domänenvariablen
R6	Domänenvariablen	Domänenvariablen

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

Harte- und weiche Constraints

Die harten Constraints gelten immer und dürfen nicht verletzt werden. Die weichen Constraints hingegen dürfen auch nicht erfüllt oder verletzt sein. Damit nicht bei jeder Verletzung eines weichen Constraints alle anderen Constraints wieder neu abgesetzt werden müssen, werden die weichen Constraints im Suchprozess behandelt. So wird zum Beispiel die Erfüllung eines Zeit-, Raum- oder Dozentenwunsches durch eine Wertzuweisung im System realisiert, wenn die gewünschte Ressource zu einem bestimmten Zeitpunkt verfügbar ist (C9 - C11).

Zur Modellierung der harten Constraints gehört auch die Festlegung der Domänengröße, zum Beispiel bei der Abbildung der Genauigkeit (C1) und bei der Abbildung von speziellen Ressourcenausprägungen bei Dozenten und Räumen (C6, C7). Die Modellierung von gesperrten 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 überlappungsfreie, eineindeutige Zuordnung sichern sollen, werden mit dem globalen Constraint diffn() oder diffn2D() realisiert (C3 - C5, C8).

Die Abbildung von Reihenfolgebeziehungen wird mit dem in beiden System neu implementierten Reihenfolgeconstraint RC1 oder RC2 umgesetzt (C2). Bei der Abbildung von verschiedenen Veranstaltungen mit unterschiedlicher Anzahl und damit unterschiedlich langen Zeitabschnitten wird das Reihenfolgeconstraint RC1 verwendet. Wenn identische Veranstaltungen mit einer konstanten Anzahl und damit über einen konstanten Zeithorizont abzubilden sind, wird das Reihenfolgeconstraint RC2 benutzt. Die Modellierung von Veranstaltungen zur gleichen Zeit, in gleichen Räumen oder mit gleichen Dozenten wird mit dem Gleichheitsconstraint = realisiert. Bei dem Vorhandensein von einzelnen speziellen Abfolgen von Veranstaltungen, die nur zwischen zwei einzelnen Veranstaltungen gelten, wird das Constraint < benutzt.

Modellierung der Constraints		
Bedingung	verwendetes Constraint in ConTime 1.0	verwendetes Constraint in Kursplan 1.0
C1	Domänengröße	Domänengröße
C1.1	notin()	notin()
C1.2	notin()	notin()
C2	Domänenvariable / RC1 / RC2 / = / <	Domänenvariable / RC1 / RC2 / = / <
C3	diffn()	diffn2D()
C4	diffn()	diffn2D()
C5	diffn()	diffn2D()
C6	Domänengröße	Domänengröße
C7	Domänengröße	Domänengröße
C8	diffn()	diffn2D()
C9	Suche	Suche
C10	Suche	Suche
C11	Suche	Suche

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 Systen **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 durchgeführen Tests mit den beiden Planungssystem **ConTime 1.0** und **Kursplan 1.0** erläutert.

Kapitel 8

Ergebnisse und Bewertung

8.1 ConTime 1.0

Die Ergebnisse der Experimente werden in diesem Abschnitt dargestellt. Es wurden verschiedene praktische Fälle des Planungsproblems an einer Weiterbildungseinrichtung untersucht. In allen Fällen sind die Kurse in einem Planungshorizont von 75 Wochen entsprechend der vorgegebenen Ressourcenwünsche einzuplanen. Die Anzahl der Backtrackingschritte wurde für die Suche nicht begrenzt. Die Rechenzeiten wurden auf einem Personalcomputer mit einer Taktfrequenz von 700MHz und einem AMD-K7-Prozesser mit 128 MByte Hauptspeicher in Sekunden ermittelt. Es wird in den Tabellen die Zeit für die Suche der Lösung angegeben. Die Planungsprobleme wurde entsprechend ihrer Größe auf mehrere Tabellen aufgeteilt. Die Tabellen haben eine laufende Nummer in der ersten Spalte zur Nummerierung der einzelnen Testfälle. In der nächsten Spalte 'Problem' sind die Kursabkürzungen, die Anzahl der Kurse, die Anzahl der Lehrveranstaltungen in den Kursen und die Anzahl der Wochen angegeben, in welchen die Lehrveranstaltungen verplant werden. In der nächsten Spalte 'Backtracking' wird die Zeit für das Constraintabsetzen und die Zeit für das Finden der Lösung angegeben. In der vorletzten Spalte 'LA/RC' ist die Zeit für die Lösungsfindung von Look ahead mit dem Reihenfolgeconstraint RC1/RC2 und in der letzten Spalte 'Faktor' ist der Faktor verzeichnet, der angibt, um wieviel schneller die Domänenzerlegung gegenüber dem Backtracking ist. Die Zeit für die Lösungsfindung beinhaltet die Zeit für das Absetzen der Constraints und die Zeit für die Suche. In der ersten Tabelle sind die kleinen Planungsprobleme mit einer bis maximal 700 einzuplanenden Einheiten dargestellt.

	Problem				*Backtracking*		*Domänenzerlegung*	
	Kurse	*Anzahl*	*Einheiten*	*Wochen*	*C.-Absetzen[sec]*	*Suche[sec]*	*LA/RC[sec]*	*Faktor*
1	t	1	1	1	0	0.1	9	–
2	8	1	72	4	0	2	9	–
3	3	1	74	6	0	9	13	–
4	1	1	176	10	0	47	24	1.96
5	9	1	214	12	15	60	21	2.86
6	2	1	224	10	16	92	28	3.29
7	1,2	2	400	19	22	174	46	3.78
8	1,2,9	3	614	22	36	344	64	5.38

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

Lösungszeit mit Backtracking und der Domänenzerlegung mit Look ahead und dem Reihenfolgeconstraint RC1/RC2 angegeben. Je größer das Problem wird, desto größer wird der Unterschied zwischen den beiden Lösungsmethoden.
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

	Problem				*Backtracking*		*Domänenzerlegung*	
	Kurse	*Anzahl*	*Einheiten*	*Wochen*	*C.abset.[sec]*	*Suche[sec]*	*LA/RC[sec]*	*Faktor*
9	4	1	751	36	216	1750	60	29.16
10	5	1	776	39	190	2470	53	46.60
11	1, 4	2	927	39	229	2084	82	25.41
12	2, 5	2	1000	44	215	3071	82	37.45
13	4, 5	2	1527	55	439	6580	130	50.62
14	1, 2, 4, 5	4	1927	57	500	9367	197	47.55
15	1, 2, 3, 4, 5, 7, 8, 9	7	2359	75	–	–	253.322	–

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 Säulendiagrammen in Abbildung 8.1 visualisiert. Bei größeren Problemen aus dem praktischen

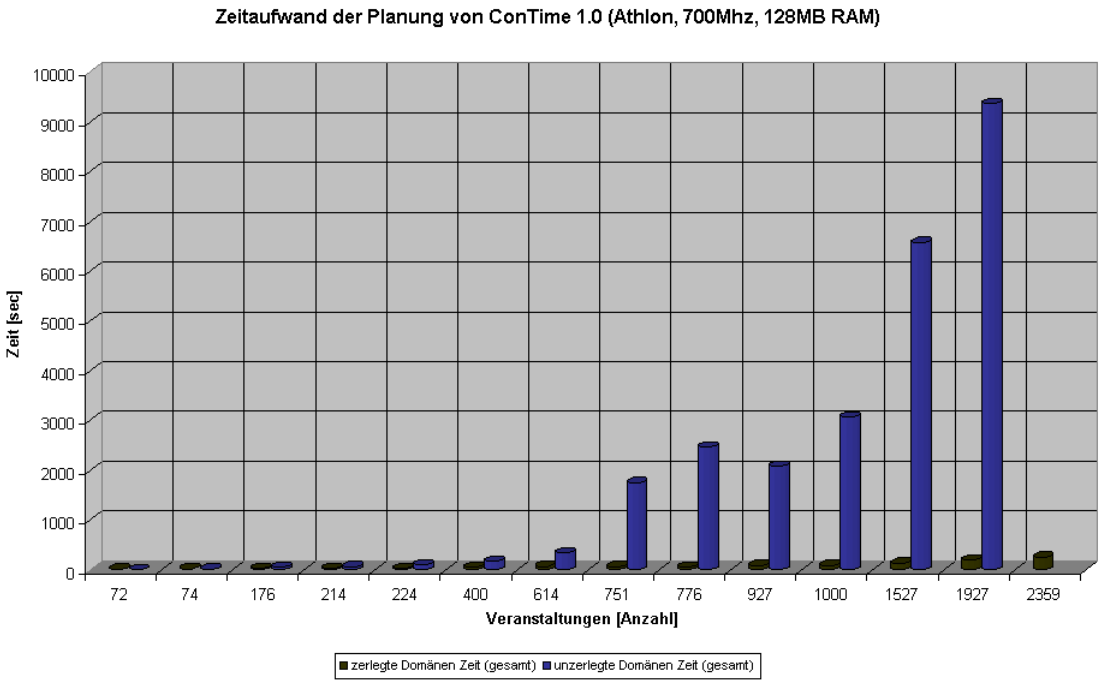

Abbildung 8.1: Darstellung der Messergebnisse von ConTime 1.0 im Säulendiagramm des Athlon - Prozessors

Beispiel zeigt sich, dass der Unterschied zwischen den Zeiten der Domänenzerlegung und den Zeiten des Backtrackings bei steigender Problemgröße zu nimmt. Der ermittelte Faktor aus den Lösungszeiten der beiden Methoden liegt bei diesen Problemen im zweistelligen Bereich von ca. 25-fach bis 50-fach. Ersichtlich ist die Abhängigkeit des Faktors von der Anzahl der beteiligten Kurse und Einheiten. Die Beispiele in denen der Kurs 4 ohne Kurs 5 beteiligt ist, liegt der Faktor zwischen ca. 25 bis 29. Bei den

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 Wertespannen (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 vermitteln. 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 ergänzend Trendlinien der Form $f(x) = \alpha x^{\beta}$ erfasst. Die Trendlinie der Form

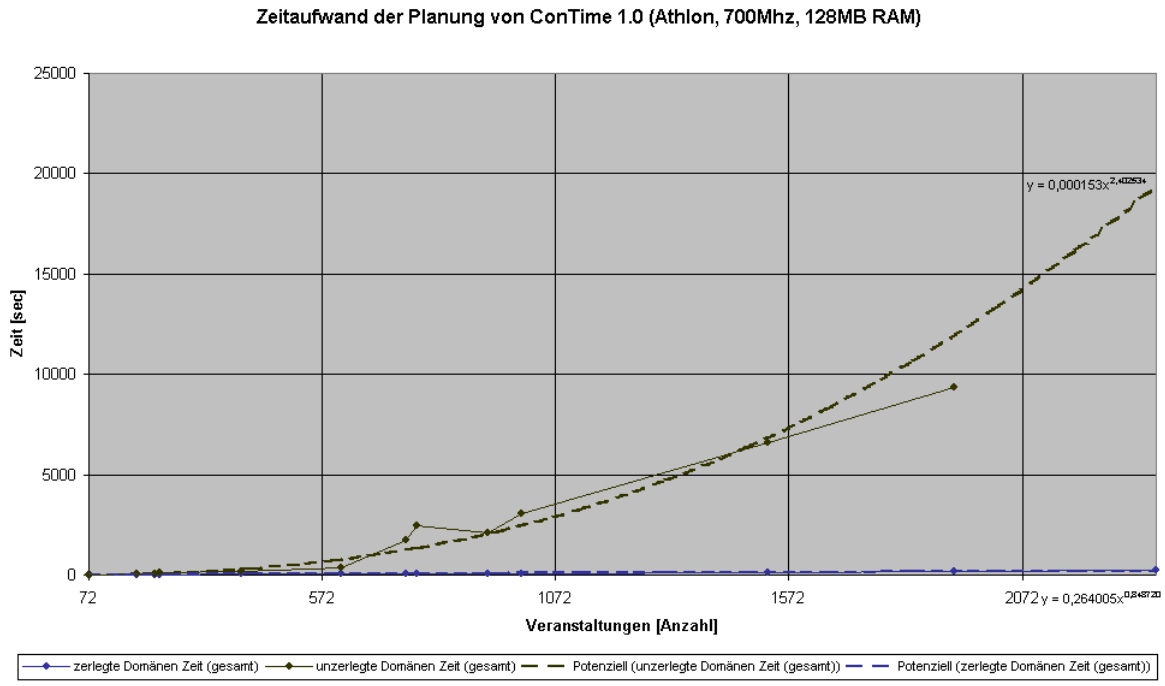

Abbildung 8.2: Darstellung der Messergebnisse von ConTime 1.0 Liniendiagramm

$f(x) = \alpha x^{\beta}$ 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 α mit 0,000153 sehr gering. Wohingegen der β-Wert viel stärker mit 2,402534 ins Gewicht fällt. Bei der Funktion für die Domänenzerlegung sind der α- und der β-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 (bezogen 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-Prozesser 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.

			mit Ressourcenwünsche				
			zerlegte Domänen		*unzerlegter Domänen*		
	Kurse	*Veranst.*	*Zeit[sec]*	*verplant*	*Constraints[sec]*	*Zeit[sec]*	*verplant*
1	8	72	6,749	72	0,832	1,473	72
2	7	72	6,92	72	1,362	1,983	72
3	3	74	8,582	74	1,673	7,951	74
4	6	144	11,725	72	2,774	6,49	144
5	1	176	12,258	176	7,941	42,591	176
6	9	214	19,327	211	0	7,134	0
7	2	224	15,702	224	10,185	71,383	224
8	1, 2	400	24,215	400	18,707	136,276	400
9	1, 2, 9	614	43,272	611	0	127,664	0
10	4	751	38,775	751	209,982	1649,982	751
11	5	776	40,048	776	370,652	2140,487	776
12	1, 4	927	53,006	927	402,3	2249,155	927
13	2, 5	1000	56,832	1000	325,969	2324	1000
14	4, 5	1527	98,612	1527	1190,863	6642	1527
15	1, 2, 4, 5	1927	130,668	1927	730	8290	1927
16	1, 2, 3, 4, 5, 7, 8, 9, 10	2359	233,035	2315			
			ohne Ressourcenwünsche				
			zerlegter Domänen		*unzerlegter Domänen*		
	Kurse	*Veranst.*	*Zeit[sec]*	*verplant*	*Constraints[sec]*	*Zeit[sec]*	*verplant*
1	8	72	6,76	72	0,851	1,462	72
2	7	72	6,97	72	1,322	1,953	72
3	3	74	7,06	74	1,673	3,005	74
4	6	144	10,1	72	2,814	6,459	144
5	1	176	8,532	176	7,951	17,765	176
6	9	214	19,974	211	0	6,889	0
7	2	224	10,335	224	10,225	26,969	224
8	1, 2	400	14,841	400	18,707	50,904	400
9	1, 2, 9	614	27,7	611	0	8,462	0
10	4	751	31,896	751	211,084	997,234	751
11	5	776	30,844	776	363,802	1338,554	776
12	1, 4	927	42,39	927	611,979	1814,598	927
13	2, 5	1000	40,318	1000	496,363	1530,51	1000
14	4, 5	1527	72,053	1527	1927,241	4196	1527
15	1, 2, 4, 5	1927	111,841	1927	2854	5670	1927
16	1, 2, 3, 4, 5, 7, 8, 9, 10	2359	142,685	2317			

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 Planungsystems nicht mehr als 1,6 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 $f(x) = \alpha x^{\beta}$ (potentiel) dargestellt. Zusätzlich

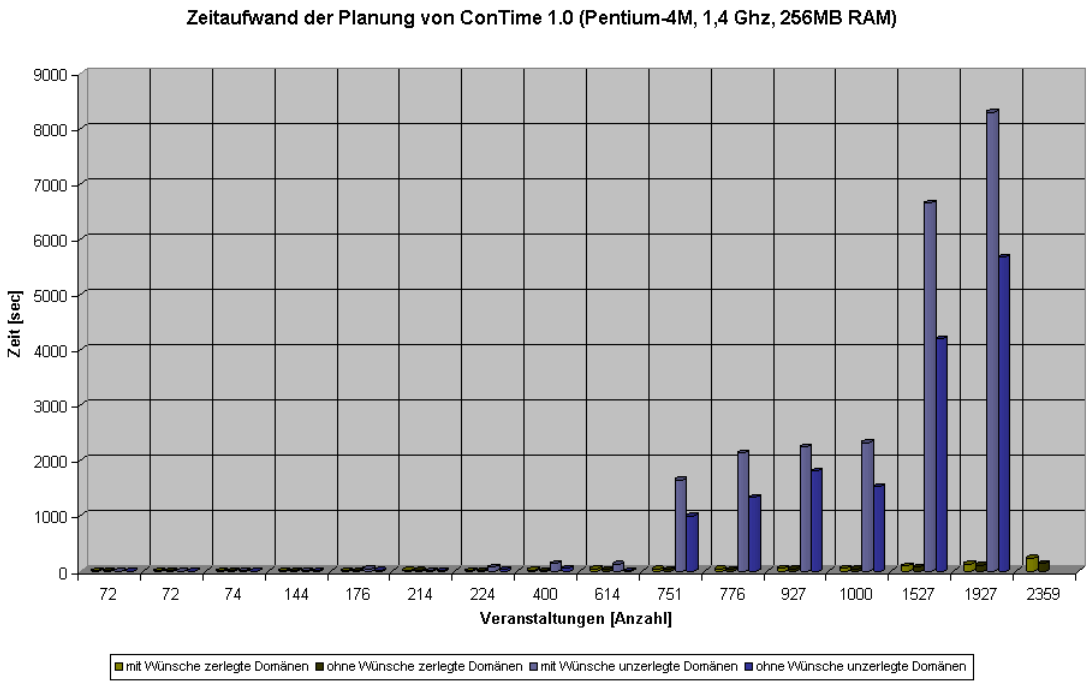

Abbildung 8.3: Darstellung der Messergebnisse von ConTime 1.0 im Säulendiagramm des Pentium - Prozessors

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 $y = 0,000033x^{2,585829}$ 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 (potentiel) $y = 0,1778x^{0,8088}$ (polynom) $y = 0,00024x^2 + 0,034103x$ und für die Domänenzerlegung (zerlegter Domänen) ohne Wünsche die Funktionen (potentiel) $y = 0,1477x^{0,8816}$ (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 α- und β-Werten der Funktionen und ein nur wenig geringerer Aufwand in den Abarbeitungszeiten.
Wenn die potentielle Funktion für die Domänenzerlegung des Pentium-Prozessors mit der potentielle 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.
Somit kann festgestellt werden, dass sich die Unterschiede in der Leistung der verwendeten Testhardware (Taktfrequenz und RAM) nur geringfügig auf die Zeiten der Problemlösung auswirken. Die Domänenzerlegung ist gegenüber dem Backtracking wesentlich im Vorteil in Bezug auf die Bearbeitungszeit. Die Lösungsqualität ist bei beiden Varianten identisch.
Die polynomen Funktionen werden beim Vergleich der Systeme im übernächsten Abschnitt betrachtet.

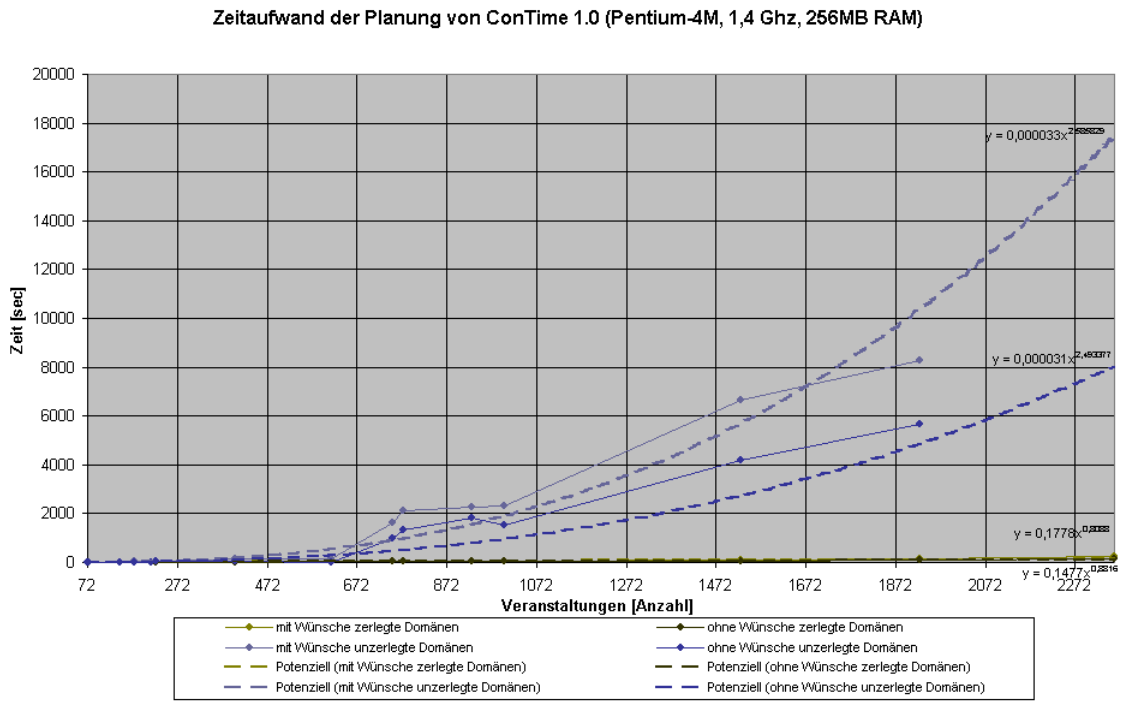

Abbildung 8.4: Darstellung der Messergebnisse von ConTime 1.0 im Liniendiagramm

8.2 Kursplan 1.0

Im folgenden Abschnitt werden die Testergebnisse des auf dem neuen Constraintlöser CS_{fd-MR} basierenden Planungssystems **Kursplan 1.0** beschrieben.
Da viele verschiedene Größen Einfluss auf das Laufzeitverhalten des Constraintlöser CS_{fd-MR} haben, lässt sich der Zeitaufwand nur in Abhängigkeit von zahlreichen Größen berechnen. Wichtig ist vor allem die Frage nach dem Worst- und dem Best-Case. Der Worst-Case tritt genau dann ein, wenn keine Ressourcenkombination gefunden werden kann, die das Einplanen einer Veranstaltung ermöglicht. Besonders negativ kann sich im Laufzeitverhalten die Zerteilung eines Tages in zahlreiche Zeitabschnitte auswirken. Dies kann zum Beispiel aufgrund zahlreicher Pausen zu treffen. Im Worst-Case sind die Zeitabschnitte zwischen den Pausen zwar groß genug, dass eine Veranstaltung theoretisch eingeplant werden könnte, steht dann aber die beanspruchte Ressource, zum Beispiel der Raum, mit hoher Priorität nicht zur Verfügung, wird die schnelle Planung durch das Testen der Ressourcen mit niedriger Priorität scheitern. Ist der Tag dagegen nicht zergliedert und führt die erste Kombination gleich zum Erfolg, so liegt der Best-Case vor. Außerdem hängt der Aufwand stark von der Reihenfolge der Veranstaltungen und Ressourcen ab, was bereits im Abschnitt zu den Heuristiken beschrieben wurde. In der nachfolgenden Testreihe wurden der Zeitaufwand und die Ressourcen konstant gehalten und nur die Anzahl der Veranstaltungen und Kurse variiert.
Zur Ermittlung des Zeitaufwandes wurde das gleiche Planungsproblem einer Weiterbildungseinrichtung verwendet, wie zum Test des Planungssystems **ConTime** im vorherigen Kapitel. Dieser Plan umfasst 75 Wochen, 8 Kurse und insgesamt 2359 zu verplanende Veranstaltungen. Um verschiedene Testszenarien zu simulieren, wurde die Anzahl der Kurse und deren Kombination verändert. Daraus resultiert eine Variation der Anzahl der zu verplanenden Veranstaltungen. Die Zeiten wurden zum einem auf einem Laptop mit Pentium 4M, 1,4 Ghz, 256 RAM und einem normalen Desktop-PC, 700 Mhz, 256 RAM ermittelt, jeweils unter Microsoft Windows 2000 und Microsoft .Net Framework 1.1. Bei dieser Testreihe blieben Ressourcen und Länge des Planungszeitraumes unverändert. Wie man den Tabellen in 8.4 und 8.5 entnehmen kann, nimmt der Aufwand mit zunehmender Anzahl der Ressourcen erwartungsgemäß stetig zu.

		nur Ress.-wünsche		*Ress.-wünsche/-präferenz*		*alle sonst. Möglich.*	
Variante	*Veranst.*	*Zeit[sec]*	*verplant*	*Zeit[sec]*	*verplant*	*Zeit[sec]*	*verplant*
Kurs 8	72	2	0	3	72	3	72
Kurs 7	72	2	0	3	72	3	72
Kurs 3	74	2	0	2	36	3	74
Kurs 6	144	3	0	3	0	4	144
Kurs 1	176	3	0	4	88	5	176
Kurs 9	214	5	0	6	106	7	214
Kurs 2	224	4	0	4	99	5	223
Kurs 1, 2	400	6	0	7	187	11	399
Kurs 1, 2, 9	614	8	0	10	293	15	613
Kurs 4	751	11	0	13	489	17	751
Kurs 5	776	13	0	16	503	18	775
Kurs 1, 4	927	14	0	18	577	21	927
Kurs 2, 5	1000	14	0	20	602	24	998
Kurs 4, 5	1527	22	0	31	992	34	1526
Kurs 1, 2, 4, 5	1927	28	0	38	1179	48	1919
Kurs 1, 2, 3, 4, 5, 7, 8, 9, 10	2359	42	0	54	1465	82	2190

Tabelle 8.4: Messergebnisse von Kursplan 1.0 und vom 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.
Bei der 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.
Bei der dritten Variante der Planung werden die Wünsche, die präferierten und die sonst verfügbaren Ressourcenausprägungen unter Einhaltung der definierten Randbedingungen geplant. Einige Veranstaltungen konnten auch bei Verwendung aller möglichen Ressourcen nicht eingeplant werden, da in den vorhandenen Testdaten zum angegebenen Fach kein Dozent gefunden werden konnte, der dieses Fach unterrichtet. Die Bedingung, dass die Dozenten das Fach der Veranstaltung unterrichten müssen, konnte nicht erfüllt werden. Dies tritt bei jeweils einer Veranstaltung im Kurs 2 und 5 auf. Zusätzlich kann es bei zu stark besetzten und in der Zeit begrenzten Stundenplänen vorkommen, dass aus Kapazitätsgründen bei Erfüllung der geforderten Randbedingungen keine passenden Zeiträume mehr vorhanden sind, um die Veranstaltung einzuplanen. Dies ist bei den letzten beiden Testvarianten der Fall. Hier sind zu viele Veranstaltungen vorhanden, die in dem vorgegeben Zeitraum und mit den vorhanden Räumen nicht zu platzieren sind. Es werden die einplanbaren Veranstaltungen eingeplant und die überzähligen

			nur Ress.-wünsche		*Ress.-wünsche/-präferenz*		*alle sonst. Möglich.*	
	Kurse	*Veranst.*	*Zeit[sec]*	*verplant*	*Zeit[sec]*	*verplant*	*Zeit[sec]*	*verplant*
1	8	72	1	0	1	72	1	72
2	7	72	1	0	1	72	1	72
3	3	74	1	0	1	36	1	74
4	6	144	1	0	1	0	1	144
5	1	176	1	0	2	88	2	176
6	9	214	1	0	2	106	2	214
7	2	224	1	0	2	99	3	223
8	1, 2	400	2	0	3	187	5	399
9	1, 2, 9	614	3	0	5	293	8	613
10	4	751	3	0	4	489	6	751
11	5	776	3	0	5	503	7	775
12	1, 4	927	5	0	7	577	10	927
13	2, 5	1000	5	0	8	602	11	998
14	4, 5	1527	5	0	10	992	14	1526
15	1, 2, 4, 5	1927	6	0	12	1179	20	1918
16	1, 2, 3, 4, 5, 7, 8, 9, 10	2359	7	0	17	1465	35	2186

Tabelle 8.5: Messergebnisse von Kursplan 1.0 und vom System Pentium 4M, 1,4 Ghz, 256 MB RAM

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).

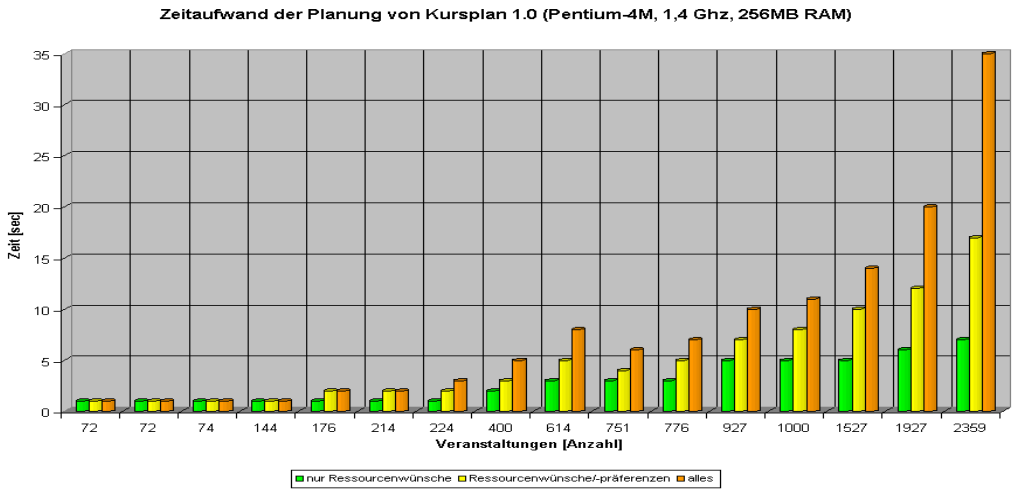

Abbildung 8.5: Darstellung der Messergebnisse von Kursplan 1.0 im Säulendiagramm

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 ergänzend Trendlinien der Form $f(x) = \alpha x^{\beta}$ und $f(x) = \gamma x^2 + \delta x$ erfasst worden.

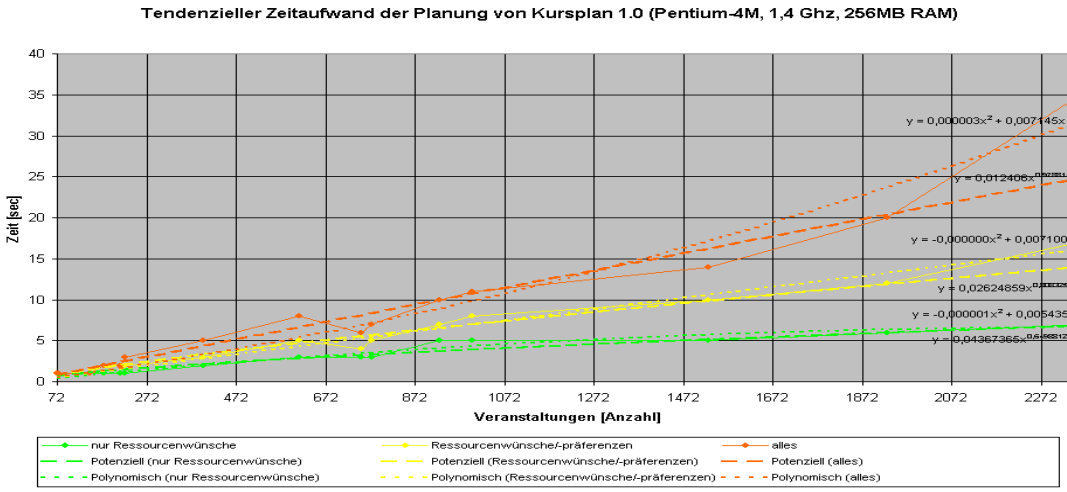

Abbildung 8.6: Darstellung der Messergebnisse von Kursplan 1.0 im Liniendiagramm

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 ($f(x) = \alpha x^{\beta}$) 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-Prozesser haben bei der Planung mit Ressourcenwünschen die Funktion (potentiell) $y = 0,03849891x^{0,88205175}$ (polynom) $y = -0,000002x^2 + 0,021074x$, für die Planung mit Ressourcenwünsche/-präferenzen die Funktion (potentiell) $y = 0,04422115x^{0,88185490}$ (polynom) $y = 0,000002x^2 + 0,017077x$ und für die Planung mit alle Ressourcenwünsche und -präferenzen zusammen mit den zusätzlichen Ressourcen die Funktion (potentiell) $y = 0,05350473x^{0,89013974}$ (polynom) $y = 0,000007x^2 + 0,016203x$.
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) $y = 0,02624859x^{0,80832558}$ (polynom) $y = 0x^2 + 0,007100x$ und für die Planung mit alle Ressourcenwünsche und -präferenzen zusammen mit den zusätzlichen Ressourcen die Funktion (potentiell) $y = 0,012406x^{0,978814}$ (polynom) $y = 0,000003x^2 + 0,007145x$.
Bei den potentiellen Funktionen der Trendlinien sind der α und β-Wert bestimmend für den Funktionsverlauf. Hingegen ist bei den polynomialen Funktionen der Trendlinien der γ-Wert sehr klein, kann vernachlässigt werden und somit hat der quadratische Anteil fast keinen Einfluss auf die Funktion. Der δ-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 Domänenzerlegung (zerlegter Domänen) vergleicht, hat man für die Planung mit Wünschen die Funktionen $y = 0,1778x^{0,8088}$ und $y = 0,04367365x^{0,64988127}$.

Beim Vergleich der Ergebnisse der beiden Planungsysteme **Kursplanung 1.0** und **ConTime 1.0**, der im folgenden Kapitel vertieft wird, ergibt sich ein Unterschied im α-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 β-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 $y = 0,00024x^2 + 0,034103x$ und $y = -0,000001x^2 + 0,005435x$. Hier zeigt sich derselbe Trend: beide Planungssysteme haben einen geringen γ-Wert, der sich ungefähr um den Faktor 240 voneinander unterscheidet und beim δ-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 Constraintsolver CS_{fd-MR} ist in dieser Problemklasse effizienter in der Problemlösung als der Constraintlöser 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_{fd-MR} in C^{++} implementiert. Interessant ist hierbei das Zeitverhalten beider Planungssysteme (siehe Abbildung 8.7).

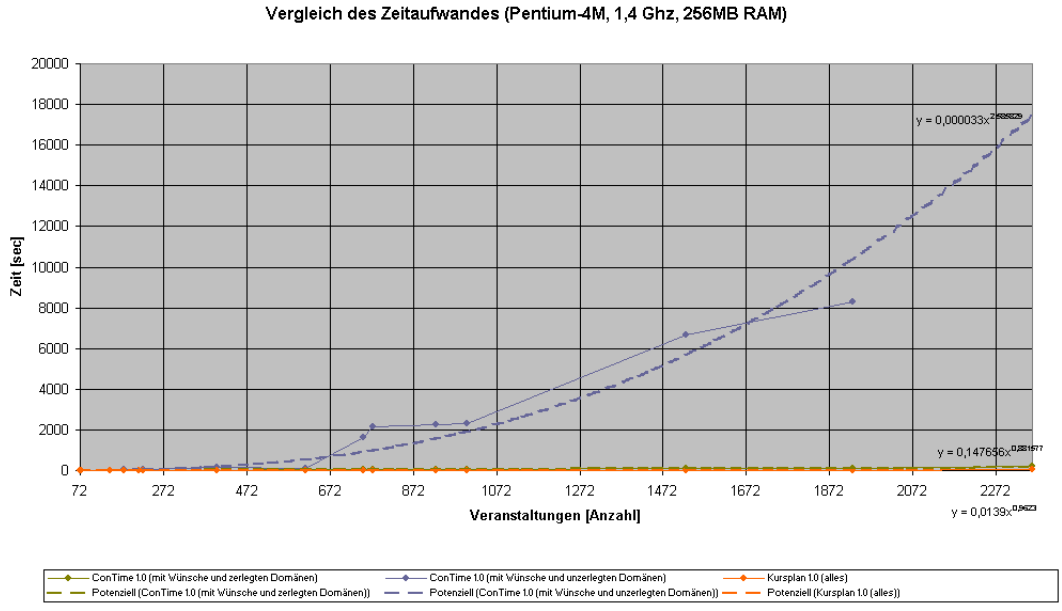

Abbildung 8.7: Vergleich ConTime 1.0 und Kursplan 1.0 mit uneingeschränkter Y-Achse

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 Programmiersystem CHIP auf dem Testsystem zu benutzen, 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 1927 Veranstaltungen. Bei der Planung mit unzerlegten Domänen stellt der enorme Speicherbedarf und der große Zeitbedarf des Planungssystems **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 Planungsvariante "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.

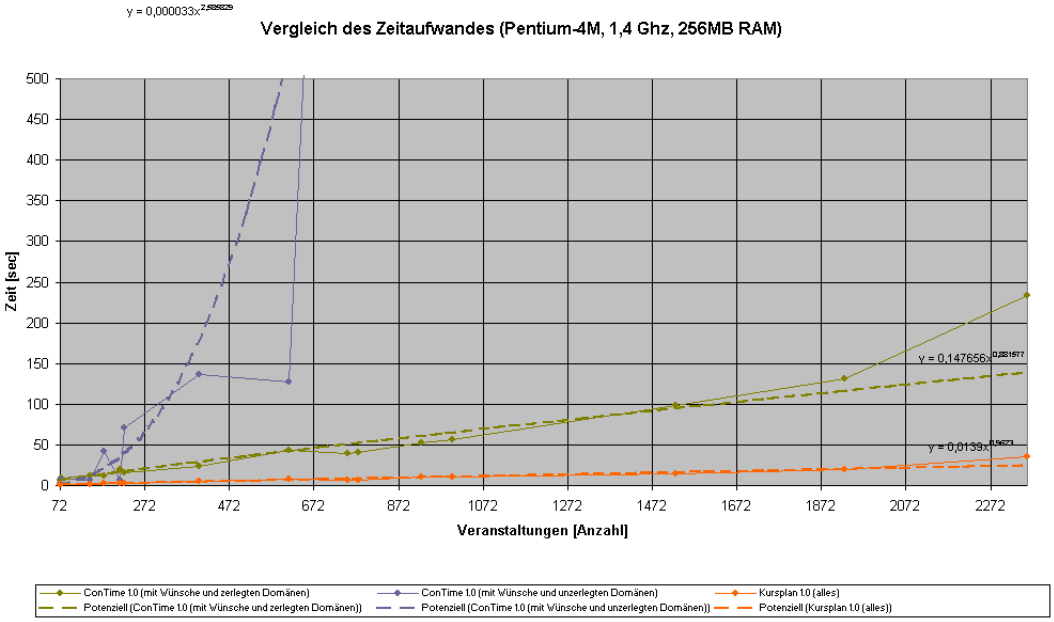

Abbildung 8.8: Vergleich ConTime 1.0 und Kursplan 1.0 mit eingeschränkter Y-Achse

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 steileren 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 $y = 0,000033x^{2,585829}$ beschrieben werden, für das System **ConTime 1.0** mit Wünschen und zerlegten Domänen mit der Funktion $y = 0,147656x^{0,881577}$ und für das System **Kursplan 1.0**

mit allen Wünschen mit der Funktion $y = 0,0139x^{0,9623}$.
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_{fd-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 Speicherverbrauchs für die interaktive Planung auch größerer Probleme geeignet.

Kapitel 9

Ausblick:

In diesem Kapitel soll auf die möglichen Erweiterungen und Anwendungsbereiche des Constraintsolvers CS_{fd-MR} eingegangen werden. In der Abbildung 9.1 sind die Komponenten des Constraintsolvers dargestellt. In einer Fortführung des Aufbaus des Constraintsolvers könnten die weiß hinterlegten Komponenten hinzugefügt werden. Zu diesen Komponenten gehören das Constraint `element` und die globalen Constraints `cycle`, `sequence` und `among`. Außerdem könnte der Constraintsolver um die Methode `Branch and bound` erweitert werden, mit deren Hilfe optimale Lösungen gefunden werden können.

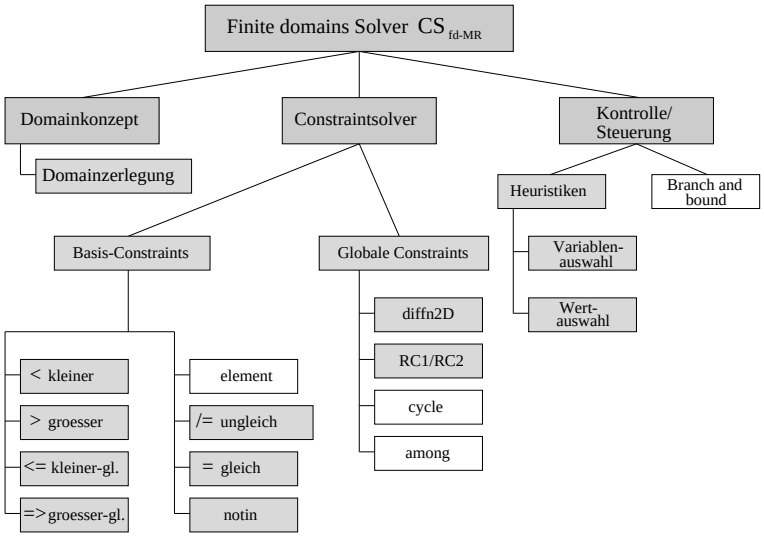

Abbildung 9.1: Erweiterte Komponenten des Constraintsolvers CS_{fd-MR}

In Anwendungen, die mit großen Datenmengen unter Echtzeitbedingungen arbeiten, wie zum Beispiel bei der Simulation in Eisenbahnnetzen, könnte die Integration des Constraintsolvers 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 Constraintsolvers würde kurze Rechenzeiten und ein effizientes Management der Speichernutzungen ermöglichen.
Um den Constraintsolvers 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 Optimierungsmethode

Die bisher integrierten Basisconstraints und die globalen Constraints `diffn2D` und `RC1/RC2` ermöglichen das Lösen von Ressourcen- und Platzierungsproblemen. Zu dieser Klasse von Problemen gehören zum Beispiel die Stundenplanung, die Weiterbildungsplanung, die Trassenplanung, die Fahrplan- und Betriebssimulation. Mit den zusätzlichen Constraints `cycle` und `among` ließen sich dann zum Beispiel Probleme, wie Tourenplanung, Zugumlaufplanung und Zugbildungsplanung realisieren. Diese Probleme gehören zu der Klasse der Logistik- und Personaleinsatzprobleme.
Das Constraint `cycle` ist ein weiteres globales Constraint und beruht auf Ideen der Graphentheorie und Kombinatorik. Dieses Constraint lässt sich für Planungsprobleme im Transportbereich anwenden, wenn Routen zu planen sind, die an mehreren verschiedenen Orten einen oder mehrere Kunden besuchen sollen. Diese Planungsproblem korrespondiert, im mathematischen Sinne, mit dem Finden eines oder mehrerer Zyklen in einem direkten Graphen. Das Constraint wird im Logistikbereich eingesetzt und eignet sich zum Lösen von Tourenplanungen, der Planung von Handelsreisen (travelling salesman) Problemen (TSP) und Personaleinsatzplanungen mit "rotierenden" Crews wie zum Beispiel bei Straßenbahnen und Flugzeugen.
Mit dem Constraint `among` können Bedingungen von Folgen und Teilfolgen von Domänenvariablen ausgedrückt werden. Es kann die Bedingung dargestellt werden, das die Werte einer Menge kleiner und größer eines festen Wertes sein sollen. Diese Werte sollen sich in einer Teilfolge befinden die eine feste vorgebene Länge haben. Dieses Constraint kann für Zeitplanungen engesetzt werden, bei denen es kleine Spannen der Überlappung in zeitlicher Hinsicht gibt. Zum Beispiel bei der groben Planung von Arbeitsaufgaben. Die ein Arbeitsaufgabe wird noch zu Ende bearbeitet während schon die nachfolgende Arbeitsaufgabe parallel begonnen wird. Der Hauptvorteil dieses Constraints liegt in der Behandlung überlappender Ressourcennutzungen.
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.
Die neuen Techniken der mehrdimensionalen Multiressourcenplanung wurden bisher nur für Probleme angewendet, bei der nur eine Ressourcendimension einen Engpass darstellt. Als nächsten Schritt könnten die Techniken auf Probleme angewendet werden, bei denen zwei oder mehr Ressourcendimensionen durch Restriktionen begrenzt sind und optimal ausgenutzt werden müssen. Bei diesen Problemen wird ein erhöhter Aufwand in der Findung einer Lösung notwendig sein, da der Lösungsraum größer ist.

9.2 Simulation in Eisenbahnnetzen

Der Constraintlöser CS_{fd-MR} könnte für die Simulation in Eisenbahnnetzen eingesetzt werden. Hierfür stünden die Vorschläge für die Modellierung aus den Veröffentlichungen [13], [57], [58] zum Förderprojekt[1] "Simulation von Trassen-Slots und Zuglagen in Eisenbahnnetzen" (SIMONE) zur Verfügung.
Die Grundlage für die Modellierung im Eisenbahnbetrieb bilden 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 der 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 Solvern 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).

[1] Förderprojekt des BMBF Förderkennzeichen: 19 P 0021

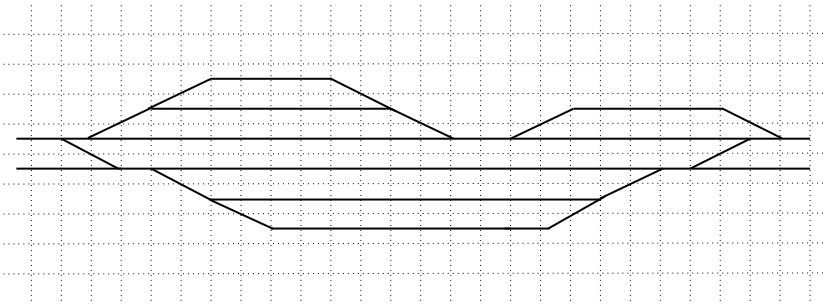

Abbildung 9.2: Diskretisierung der Infrastruktur in Spurplangenauigkeit

Infrastruktur/Topologie
Die Infrastruktur wird zur Sicherung eines konfliktfreien Fahrens in der Spurplangenauigkeit betrachtet. Das heißt, die Gitternetzlinien auf der X-Achse sollten genau an charakteristischen Infrastrukturelementen, wie Signalen, Weichen, Geschwindigkeitstafeln, Steigungsänderungen, Tunnelein- und -ausfahrten, Bogenein- und -ausfahrten, Signal- und Fahrstraßenzugschlussstellen liegen. Bei einem feinen Gitterliniennetz kann mit einem kontinuierlichen Abstand der Gitterlinien zueinander gearbeitet werden. Wohingegen bei einem groben Gitterliniennetz ein diskontinuierlicher Abstand der Gitterlinien zu einander gewählt werden muss. Mit dieser feinen Abbildung der Infrastruktur lässt sich das Fahren im relativen und absoluten Bremswegabstand abbilden, da die Abstände der Gitterlinien unter den Zuglängen und den Bremswegabständen liegen. Des weiteren lassen sich Durchrutschwege und Überlängen von Zügen bei Ein- und Ausfahrten aus Bahnhöfen betrachten.

Züge
Die auf der vorhandenen Infrastruktur fahrenden Züge werden in Ihrem Zeitverlauf auf der z-Achse abgebildet, die senkrecht auf der Ebene aus x- und y-Achse steht. Die Züge werden in die Infrastruktur durch Aneinanderreihungen von Quadern eingelastet. Die Quader sind zu Teilbelegungsabschnitten und diese Teilbelegungsabschnitte sind in übergeordneten Belegungsabschnitten zusammengefasst. Die Vorbelegung der Strecke erfolgt mit den Belegungsabschnitten und die Teilfreigabe erfolgt mit den Teilbelegungsabschnitten. Bei beiden Arten der Belegung werden durch Aneinanderreihungen von Quadern in die Infrastruktur eingelegt. Die Quader füllen die Flächen aus, die zwischen jeweils den begrenzenden sechs Gitternetzlinien der drei Achsen liegen. Eine Fläche wird in einer Dimension immer von zwei Gitternetzlinien begrenzt.

Zur Realisierung dieser Modellierung mit dem Constraintlöser CS_{fd-MR} könnte das globale Constraint **diffn2D()** und die enthaltenen Basisconstraints $=$ und $\geq$ verwendet werden. Es müßte untersucht werden, ob mit dem Constraintlöser CS_{fd-MR} die Realisierung der Fahrplan- und Betriebssimulation und die Trassenplanung von Zügen effizient möglich ist. Hierzu sollte in verschiedenen Testreihen die Verarbeitungszeit von großen Netztopologien mit einer großen Anzahl von Zügen bei der Steigerung des Simulations-/ Planungshorizonts betrachtete werden. Im weiteren Verlauf der Erstellung eines Softwaretools für die Trassenplanung, Fahrplan- und Betriebssimulation müßten die Zusammenhänge zwischen der Infrastrukturgröße, der Anzahl der zu simulierenden Züge, der Abbildungsgenauigkeit, der Lösungsqualität und der Bearbeitungszeit genauer untersucht werden. Der Zusammenhang zwischen der Infrastrukturgröße und der Anzahl der Züge in Bezug auf den benutzten Speicher sollten ebenfalls Gegenstand der Untersuchungen sein. Meist steigt die Abarbeitungszeit sehr großer Probleme bei einem hohen Speicherbedarf der Anwendung ab einem bestimmten Wert der Speichernutzung sprunghaft an, da Auslagerungen der Daten aus dem Hauptspeicher auf die Platte erfolgen müssen.

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 spezialisierten Dozenten. Die Daten umfaßten nahezu alle bei einer Kursplanung vorkommenden Bedingungen.

Bei der Lösung von großen Multiressourcenproblemen fallen die Defizite der Constraint-basierten Programmierung wesentlich ins Gewicht, da sie eine zeitnahe und interaktive Nutzung von Constraintsystemen erschweren. Zu ihnen gehören, neben dem mit der Problemgröße exponentiell steigenden Lösungsaufwand, die begrenzte Speichernutzung, die eingeschränkte Variablenanzahl und der automatische Wechsel zur komplexitätsreduzierenden Intervallpropagation mit ausschließlicher Einschränkung der Werte an den Intervallgrenzen der Domänenvariablen.

Mit dem in dieser Arbeit erläuterten Constraintlöser für Multiressourcenprobleme konnte der Lösungsaufwand, der bei den zur Zeit verfügbaren kommerziellen Constraintlösern mit der Problemgröße exponentiell ansteigt, für eine wesentlich erweiterte Problemgröße signifikant verringert werden. In der Arbeit wurde an einem realen Anwendungsbeispiel gezeigt, dass die starke Verringerung des Anstiegs des Lösungsaufwands auch auf große Pobleme realisierbar wird durch eine Problemzerlegung des mehrdimensionalen Multiressourcenproblems in mindestens einer Dimension, durch die Einführung einer zentralen Konsistenzsicherung und Bereichsverwaltung in speziellen Wertepräsentationen der Domänenvariablen und durch die Implementierung von speziellen globalen Constraints für die Modellierung von Ressourcen und Reihenfolgebeziehungen.

In der vorliegenden Arbeit wurden die Eigenschaften des Constraintslösers für mehrdimensionale Multiressourcenproblme am Problem der mehrjährigen Kursplanung für Weiterbildungseinrichtungen vorgestellt. Die Implementierung des Constraintlösers 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 Eisenbahnnetzen erfolgreich getestet worden.

Abbildungsverzeichnis

Tabellenverzeichnis

Index

Literaturverzeichnis

[1] A. Aggoun, D. Chan, P. Dufresne, E. Falvey and H. Grant. *ECL^{i}PS^e* User Manual. Release 5#3. 2002.

[2] Ausschuss für wirtschaftliche Fertigung e.V. (Hrsg.). Integrierter EDVEinsatz in der Produktion. Computer Integrated Manufacturing - Begriffe, Definitionen, Funktionszuordnungen. Empfehlung, Eschborn, 1985.

[3] S. Abdennadher and M. Marte. University timetabling using constraint handling rules. In O. Ridoux, editor, *Proc. JFPLC'98, Journées Francophones de Programmation Logique et Programmation par Constraintes*, pages 39 –49, Paris, 1998. Hermes.

[4] A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *J. Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[5] F. Azevedo and P. Barahona. Timetabling in constraint logic programming. In *Proc. World Congress on Expert Systems*, 1994.

[6] R. Bartak. On-line Guide to Constraint Programming. Prague,1998, *http://kti.mff.cuni.cz/ bartak/constraints/.*

[7] J. Beck. Dynamische Bewertung von Ressourcenbelegungsplänen durch Simulation. Diplomarbeit, Universität Würzburg, 1998.

[8] E. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *J. Mathematical and Computer Modelling*, 20(12):97–123, 1994.

[9] C. Bessiere. Arc-consistency and arc-consistency again. Artificial Intelligence, 65(1):179–190, 1994.

[10] C. Bessiere, E.C. Freuder and J.-R.Regin. Using constraint metaknowledge to reduce arc consistency computation. Artificial Intelligence, 107:125–148, 1999.

[11] J. Blazewicz. Scheduling in Computer and Manufacturing Systems. SpringerVerlag, Berlin, 1993.

[12] P. Boizumault, Y. Delon, and L. Peridy. Constraint logic programming for examination timetabling. *J. Logic Programming*, 26(2):217–233, 1996.

[13] M. Bolemant, D. Matzke. Modellierung und Simulation von Fahrplänen für den Eisenbahnverkehr mit constraint-basierten Verfahren. ASIM 2002. SCM - The Society for Modeling and Simulation International, ISBN 3-936150-19-2, 2002.

[14] E. Burke and M. Carter, editors. *Practice and Theory of Automated Timetabling II*, volume 1408 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 1998.

[15] E. Burke, D. Eiiliman, P. Ford, and R. Weare. Examination timetabling in British universities: A survey. In *[16]*, pages 76–90, 1996.

[16] E. Burke and P. Ross, editors. *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 1996.

[17] C. Busch. Spezifikation eines kooperativen Werkzeugs zur Lösung komplexer Zuordnungsprobleme. Diplomarbeit, Universität Würzburg, 1997.

[18] Y. Caseau and F. Laburthe. Improved CLP scheduling with task intervals. In Pascal van Hentenryck, editor, Proceedings of the Elevnth International Conference on Logic Programming,ICLP'94, pages 369–383, MIT Press, 1994.

[19] P. Codognet and D. Diaz. Yet another local search method for constraint solving. In K. Steinhöfel, editor, Proc. SAGA 2001, LNCS 2264, pages 73–89. Springer, 2001.

[20] T. B. Cooper and J. H. Kingston. The complexity of timetable construction problems. In *[16]*, pages 183–295, 1996.

[21] Dialog Computer Systeme GmbH. On-line Beschreibung für die Stundenplansoftware gp-Units. Berlin, 2004, *http:www.dcs.de/produktbereiche/stundenplan.html.*

[22] W. Dilger, S. Kassel, O. Schumann and S. Schumann. Repräsentation von Planungswissen für verteilte PPSSysteme. In C. Klauck and J. Müller, editors: Künstliche Intelligenz und Verteilte PPS-Systeme, Universität Bremen. Beiträge der 1. Bremer KI Pfingstworkshops, Bericht Nr. 5/95, 1995.

[23] W. Dilger and S. Kassel. Sich selbst organisierende Produktionsprozesse als Möglichkeit zur flexiblen Fertigungssteuerung. In J. Müller, editor: Verteilte Künstliche Intelligenz, pages 347–356, Mannheim. BI Wissenschaftsverlag, 1993.

[24] M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Int. Conf. Fifth Generation Computer Systems (FGCS'88)*, pages 693–702, Tokyo, 1988.

[25] W. Domschke and A. Drexl. Einführung in Operations Research. Springer-Verlag, 1991.

[26] H. El Sakkout. Improving Backtrack Search: Three Case Studies of Localized Dynamic Hybridization. PhD thesis, Imperial College, London, June 1999.

[27] H. El Sakkout and M. Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. Constraints, 5(4):359–388, 2000.

[28] F. Focacci, F. Laburthe and A. Lodi. Local search and constraint programming. In Tutrial notes at CP 2001, 2001. Also published in [33].

[29] T. Frühwirth and S. Abdennadher. Constraint-Programmierung. Springer-Verlag, 1997.

[30] T. Frühwirth, A. Herold, V. Küchenhoff, T. Le Provost, P. Lim, E. Monfroy and M. Wallace. Constraint Logic Programming. An Informal Introduction. Technical Report ECRC-93-5, European Computer-Industry Research Centre Munich, 1993.

[31] C. Gervet. Interval propagation to reason about sets: Definition and implementation of a practical language. Constraints, 1(3):191–244, 1997.

[32] U. Geske, H.-J. Goltz, U. John, D. Matzke, A. Wolf. Constraint-basiert Planung und Simulation von Multiressourcen Problemen. GMD Report 28, 1998.

[33] F. Glover and G. Kochenberger, editors. Handbook on Metaheuristics. Academic Publishers, 2001.

[34] H.-J. Goltz. Reducing domains for search in CLP(FD) and its application to job-shop scheduling. In U. Montanari and F. Rossi, editors, *Principles and Practice of Constraint Programming – CP'95*, volume 976 of *Lecture Notes in Computer Science*, pages 549–562, Berlin, Heidelberg, 1995. Springer-Verlag.

[35] H.-J. Goltz und D. Matzke. Beschreibung des Demonstrationsprogrammes zur Anwendung der Constraintmethode auf ein einfaches, ideallisiertes Simulationsproblem. Technischer Report, Frauenhofer FIRST, PlanT, 2002.

[36] C. Guéret, N. Jussien, P. Boizumault, and C. Prins. Building university timetables using constraint logic programming. In *[16]*, pages 130–145, 1996.

[37] R. Hackstein. Produktionsplanung und steuerung (PPS). VDI Verlag, Düsseldorf, 2. Auflage, 1989.

[38] H. Havermann. Der Leitstand als Integrator in einem CIMKonzept. HMD, 27(151):37–50, 1990.

[39] M. Henz and J. Würtz. Using Oz for college timetabling. In *[16]*, pages 162–177, 1996.

[40] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge (Mass.), London, 1989.

[41] P. van Hentenryck, Y. Deville and C.-M. Teng. A generic arc-consistency algorithm and its specializations. Artificial Intelligence, 57: 291–321, 1992.

[42] P. van Hentenryck and T. le Provost. Incremental search in constraint logic programming. New Generation Computing, 9(3&4):257–275, 1991.

[43] P. Hofstedt und A. Wolf. Einführung in die Constraint-Programmierung. Scriptum zur Vorlesung, TU Berlin, FG Übersetzerbau und Programmiersprachen, 2002.

[44] ILOG. CPLEX. www.ilog.com/products/cplex/, 2001.

[45] ILOG. Constraintsolving in ILOG. White Paper.

[46] O. Kamarainen and H. El Sakkout. Local probing applied to scheduling. In P. van Hentenryck, editor, Principles and Practice of Constraint Programming-CP2002, Springer LNCS 2470, pages 155–171, 2002.

[47] M. Kambi and D. Gilbert. Timetabling in constraint logic programming. In *Proc. 9th Symp. on Industrial Applications of PROLOG (INAP'96)*, Tokyo, Japan, 1996.

[48] H. Kernler. PPS der 3. Generation - Grundlagen, Methoden, Anregungen. Hüthig Verlag, Heidelberg, 1993.

[49] K. Kurbel. Produktionsplanung und steuerung: Methodische Grundlagen von PPSSystemen und Erweiterungen. Oldenbourg Verlag, München, Wien, 1993.

[50] K. Kurbel und J. Meynert. Flexibilität in der Fertigungssteuerung durch Einsatz eines elektronischen Leitstands. ZwF, 83(12):581–585, 1988.

[51] G. Lajos. Complete university modular timetabling using constraint logic programming. In *[16]*, pages 146–161, 1996.

[52] C. Le Pape and P. Baptiste. Resource constraints for preemptive job-shop scheduling. Constraints, 3(4):263–287, 1998.

[53] T. Le Provost. Approximate Generalised Propagation. ESPRIT Project Deliverable CORE-93-7, also as CHIC-WP5-D.5.2.3.3., ECRC GmbH, January 1993.

[54] T. Le Provost and M. Wallace. Domän-independent propagation (or Generalised Propagation). In Proceedings of the International Conference on Fifth Generation Computer Systems (FGC$S^{(}$)92), pages 1004–1011, June 1992.

[55] J.W. Lloyd. Foundations of Logic Programming. Spriger-Verlag, 1997.

[56] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. Artifical Intelligence, 25(1), 1984.

[57] D. Matzke und M. Bolemant. Modellierungssprache für die Disposition in Eisenbahnnetzen. 4. Braunschweiger Symposium. Automatisierungs- und Assistenzsysteme für Transportmittel, Fortschritt-Berichte VDI, Reihe12 Verkehrstechnik/Fahrzeugtechnik, Nr. 525, ISBN 3-18-352512-7, 2003.

[58] D. Matzke und M. Bolemant. Modellierung innovativer Systemtechniken der Zugbeeinflussung mit constraint-logischer Programmierung. ASIM 2003. SCM - The Society for Modeling and Simulation International, 2003.

[59] R. Mohr and T.C. Henderson. Arc and path consistency revised. Artificial Intelligence, pages 225–233, 1986.

[60] D. Matzke und A. Pohlmann. CBSIM. Manual, 2002.

[61] K. Marriott and P.J. Stuckey. Programming with Constraints. An Introduction. The MIT Press, 1998.

[62] M. Musiol. Implementierung von parallelem Divide-and-Conquer auf Gittertopologien. Diplomarbeit, Universität Passau, 1996.

[63] H. Nadzeyka und B. Schnabel. Untersuchungen über die Fertigungsdurchlaufzeiten in der MaschinenbauIndustrie. REFANachrichten, 28(5):267–271, 1975.

[64] K. Neumann. Operations Research Verfahren, Bd.I. Carl Hanser Verlag, 1975.

[65] U. Nilsson and J. Maluszy*n*ski. Logic, Programming and Prolog. John Wiley & Sons Ltd., 1995.

[66] F. Puppe. Problemlösungsmethoden in Expertensystemen. Springer, Heidelberg, Berlin, 1990.

[67] J.-C. R*e*gin. A filtering algorithm for constaints of difference in CSP's. IN Proceedings of the National Confernce on Artificial Intelligence, pages 362–367, 1994.

[68] G. Ringwelski and D. Matzke. Integration lokaler Suche in CHIP. Technischer Report, Frauenhofer FIRST, PlanT, 2003.

[69] G. Rößling. Quicksort nach Divide and Conquer. 1999, *http://www.animal.ahrgr.de/Anims/de/quicksortDAndC.aml.*

[70] F.J. Schlichte und J. Dicks: Höchster. Planungsbedarf für die Fertigung - Terminsteuerung mit dem klassischen Produktionsleitstand. HMD, 27(151):103–110, 1990.

[71] SICStus Prolog User's Manual. Release 3#5. Swedish Institute of Computer Science. Kista, Sweden, 1996.

[72] Scientia GmbH. On-line Beschreibung für die Stundenplansoftware S-Plus. Köln, 2004, *http:www.scientia.de/produkte/splus.htm.*

[73] Stüber Software. On-line Beschreibung für die Stundenplansoftware daVinci. Koblenz, 2004, *http:www.stueber.de/daVinci/davinci.html.*

[74] P. van Hentenryck. Constraint Satisfaction in Logic Programming. Logic Programming Series. MIT Press, Cambridge, MA, 1989.

[75] J. Wellner and W. Dilger. MAPS - A Multi-Agent Production Planning System. In A. Holsten, G. Joeris, C. Klauck, M. Klusch, H.-J. Müller, and J. Müller, editors: Intelligent Agents in Information and Process Management, Workshop at the 22nd German Annual Conference on Artificial Intelligence in Bremen (KI98), Universität Bremen - TZIBericht Nr. 9, 1998.

[76] J. Wellner and W. Dilger. MAPS - A Multi-Agent Production Planning System. Akzeptiert für den Workshop Planen und Konfigurieren 99 während der XPSTagung im März 1999 in Würzburg.

[77] G.M. White and J. Zhang. Generating complete university timetables by combining tabu search with constraint logic. In *[14]*, pages 187–198, 1998.

[78] H.P. Wiendahl. Belastungsorientierte Fertigungssteuerung - Grundlagen, Verfahrensaufbau, Realisierung. Hanser Verlag, München, Wien, 1987.

[79] SchulSoftWare A. Tillmann. On-line Beschreibung für die Stundenplansoftware Winschule. Berlin, 2004, *http:www.winschule.de.*

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

Anhang A

Funktionen des globalen Ressourcen-Constraints diffn2D

Konstanten

- $CS-RESSOURCE-STATE-FREE$ entspricht der -1, kennzeichnet einen Wert als unbenutzt
- $CS-RESSOURCE-STATE-NOT-AVAILABLE$ entspricht der -2, kennzeichnet einen Wert als nicht verfügbar

Konstanten zum Laufzeit-Status

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

Definition und Initialisierung

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

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

- **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.

- **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 einzuhaltenden 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 einzuhaltenden 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):
  Die Statuswerte, die nur temporär in Bezug auf Distanzen als neutral betrachtet werden sollen, können mit dieser Funktion wieder freigegeben werden. Allerdings bewirkt diese Funktion nicht, dass bereits getätigte Einfügungen von Rechtecken, die nach dem Entfernen des Status nicht mehr den Restriktionen entsprechen, entfernt werden. Die Änderung wird nur in Bezug auf Änderungen am Lösungsraum berücksichtigt.

Passage

- bool MakePassageList(const ValueList* values, PassageList* passages):
  Einige Funktionen bekommen einen Zeiger auf eine ValueList und füllen diese entsprechend mit Werten. Manchmal ist es aber von Vorteil, wenn man nicht die freien Spalten/Zeilen zurückbekommt, sondern eine Liste mit Abschnitten, die den Beginn und die Länge von Freiräumen beschreiben. Genau für diesen Zweck gibt es diese Funktion. Sie gibt zu einer ValueList die passende PassageList zurück. Wichtig dabei ist, dass die Werte in der ValueList aufsteigend sortiert sind.

Nutzung bekannt geben(Lock)

- bool Lock(bool canOverwrite=false):
  Sie kennzeichnet alle Werte des Feldes mit dem Wert $CS-RESSOURCE-STATE-NOT-AVAILABLE$. Ein weiteres Änderung von Werten ist dann nicht mehr möglich.

- 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):
  Sie kennzeichnet alle Werte innerhalb des angegebenen Rechteckes mit dem Wert $CS-RESSOURCE-STATE-NOT-AVAILABLE$. Die weiteren Änderungen von Werten innerhalb des Rechteckes sind dann nicht mehr möglich.

- bool Lock(DWORD firstX, DWORD firstY, DWORD width, DWORD height, short state, bool canOverwrite=false):
  Sie kennzeichnet alle Werte innerhalb des Rechteckes mit dem übergebenen Wert. Die weiteren Änderung von Werten innerhalb des Rechteckes ist dann nicht mehr möglich.

Verschieben

In X-Richtung.

- bool ChangeX(DWORD rowXOld, DWORD rowXNew, short state):
  Verschiebt die mit dem Wert gekennzeichneten Werte aus der Reihe rowXOld in die Reihe rowXNew, falls möglich. In der Reihe rowXOld wird für jeden Wert der Wert $CS-RESSOURCE-STATE-FREE$ gesetzt.

- 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 $CS - RESSOURCE - STATE - FREE$ gesetzt.

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

In Y-Richtung.

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

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

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

Vertauschen

- bool Cross(short state1, short state2):
  Vertauscht zwei Werte miteinander.

Nutzung aufheben (UnLock)

- bool UnLock():
  Gibt alle Werte mit f(x, y) ≥ 0 des gesamten Lösungsbereichs frei und kennzeichnet sie mit $CS - RESSOURCE - STATE - FREE$.

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

- bool UnLock(DWORD firstX, DWORD firstY, DWORD width, DWORD height):
  Gibt alle Werte mit f(x, Y) ≥ 0 innerhalb des übergebenen Rechteckes frei und kennzeichnet sie mit $CS - RESSOURCE - STATE - FREE$.

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 im Bereich von x bis x+length zurück, die mit $CS - RESSOURCE - STATE - FREE$ gekennzeichnet sind.

- void getFreeY(DWORD x, DWORD length, list<DWORD>* value, const list<short>* neutrals):
  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):
  Prüft, ob das angegebene Rechteck mit $CS - RESSOURCE - STATE - FREE$ gekennzeichnet ist.

- 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):
  Sie teilt den übergebenen Bereich in als mit $CS - RESSOURCE - STATE - FREE$ 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. Dadurch ist height immer

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):
  Sie teilt den übergebenen Bereich in als mit einem der Werte 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. Dadurch ist height immer 1 und width entspricht der Länge des freien Bereiches.

- void IntersectX(DWORD firstX, DWORD firstY, DWORD width, DWORD height, list<*CS − Rectangle*>* value, DWORD minLength):
  Sie teilt den übergebenen Bereich in als mit $CS - RESSOURCE - STATE - FREE$ 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.

- void IntersectX(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. Dadurch ist height immer 1 und width entspricht der Länge des freien Bereiches.

- void IntersectY(DWORD firstX, DWORD firstY, DWORD width, DWORD height, list<*CS − Rectangle*>* value):
  Sie teilt den übergebenen Bereich in als mit $CS - RESSOURCE - STATE - FREE$ 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. Dadurch ist width immer 1 und height entspricht der Länge des freien Bereiches.

- void IntersectY(DWORD firstX, DWORD firstY, DWORD width, DWORD height, list<*CS − Rectangle*>* value, const list<short>* neutrals):
  Sie teilt den übergebenen Bereich in als mit einem der Werte 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. Dadurch ist width immer 1 und height entspricht der Länge des freien Bereiches.

- void IntersectY(DWORD firstX, DWORD firstY, DWORD width, DWORD height, list<*CS − Rectangle*>* value, DWORD minLength):
  Sie teilt den übergebenen Bereich in als mit $CS - RESSOURCE - STATE - FREE$ 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. Dadurch ist width immer 1 und height entspricht der Länge des freien Bereiches.

- 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 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 erfolglosen 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

```
bool Schedule(void)
```

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

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

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

```
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.

```
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.

```
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.

```
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.

```
bool UnSchedule()
```

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

```
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 (0x1)

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_SCHEDULETIME (0x2)

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 (0x4)

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

SCHEDULER_FLAG_USEPOSSIBILITIES (0x8)

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 (0x10)

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 (0x20)

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 (0x40)

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 (0x80)

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 (0x100)

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.

```
SCHEDULER_FLAG_LOG_DETAILED (0x200)
```

Für dieses Flag gilt das gleiche wie für SCHEDULER_FLAG_LOG_STANDARD. Die generierten Nachrichten geben detailliertere Auskünfte über den aktuellen Planungsprozess. Wenn das Flag gesetzt wird, muss nicht mehr SCHEDULER_FLAG_LOG_STANDARD gesetzt werden.

```
SCHEDULER_FLAG_DEFAULT (MAX_WORD)
```

Alle vorangegangenen Flags werden gesetzt, wenn dieses Flag gesetzt wird. Nähere Informationen zu den einzelnen Flags stehen bei den entsprechenden Flags.

```
SCHEDULER_LOG_FUNC getLogEvent(void)
void setLogEvent(SCHEDULER_LOG_FUNC value)
```

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

```
SCHEDULER_EVENT_PROCESSED_FUNC -
getSchedulerProcessedEvent(void)
void setSchedulerProcessedEvent -
(SCHEDULER_EVENT_PROCESSED_FUNC value)
```

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

```
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.

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

Diese Eigenschaft erlaubt festzulegen, unter welcher Nummer die Veranstaltung eingeplant werden soll. Unterschiedliche Nummern könnte man zum Beispiel verwenden, um zusätzliche Eigenschaften von geplanten Veranstaltungen zu codieren. Interessant wären hier zum Beispiel die Vertretungsstunden.

Anhang D

Stundenplanungssysteme

Die beiden Stundenplanungssysteme **ConTime 1.0** in der Programmiersprache CHIP und **Kursplan 1.0** auf Basis des Constraintlöser CS_{fd-MR} wurden ähnlich realisiert und bestehen aus einer Datenhaltungs-, Visualisierungs- und Planungskomponente.

Die Datenhaltungskomponente liest die mit einem graphischen Editor erstellten Problemdefinitionen aus einer Datei ein und überprüft hierbei die Konsistenz der eingelesenen Daten. Gleichzeitig wird eine interne Objektstruktur angelegt, in der die eingelesenen Daten abgelegt werden.
In der Visualisierungskomponente können die Daten in unterschiedlichen Ansichten angezeigt werden. Es stehen Kurs-, Raum- und Dozentenansichten zur Verfügung, in denen die einzelnen Kurse und Dozenten mit ihren Veranstaltungen und die einzelnen Räume mit ihren Belegungen dargestellt werden. Für die interaktive Nutzung der Visualisierungskomponente gibt es die Möglichkeit sich freie Kapazitäten für Räume und Dozenten sowie für Zeitpunkte und Zeitintervalle anzeigen zu lassen. Hierbei werden die freien Zeiträume in den entsprechenden Kursen beachtet und nur die freien Kapazitäten zu den freien Zeiten in den Kursen angezeigt.
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.

Bei der Realisierung des Systems **Kursplan 1.0** mit dem neuen Constraintlöser CS_{fd-MR} wurde ein gemischter Ansatz für die Implementierung des Systems gewählt. Die Komponenten der Datenverwaltung und der Planung, welche einen hohen Rechenaufwand besitzen und viele Aktionen auf dem Speicher ausführen, wurden in dem maschinennahen Code C^{++} implementiert.
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.
Die Visualisierungskomponente, die zur Steuerung des Systems und zur Anzeige der Daten dient, wurde im interpretierten Code $.NET$ implementiert. In der Programmiersprache $.NET$ stehen hierfür viele Funktionen zur Verfügung, die eine nutzerfreundliche und schnelle Realisierung einer Visualisierung erlauben.

In der nachfolgenden Abbildung D.1 sind die Hauptkomponenten der beiden Systeme für die Stundenplanung auf der Basis der kommerziellen Programmierssystems CHIP und des neuen Constraintlösers CS_{fd-MR} dargestellt.

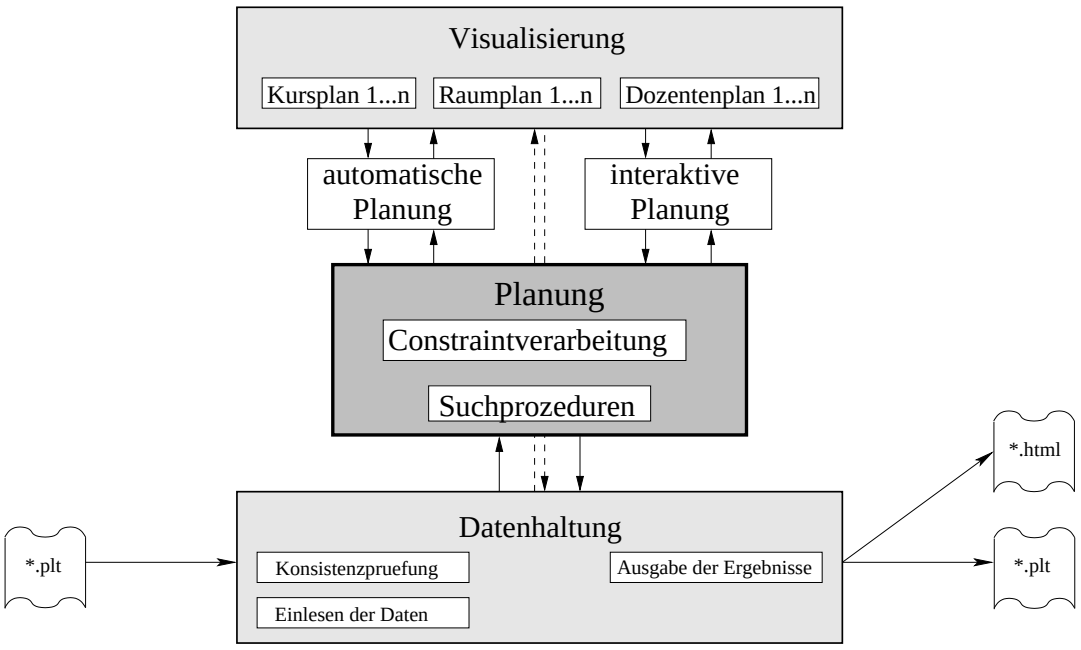

Abbildung D.1: Komponenten der beiden Stundenplanungssysteme mit CHIP und dem Constraintlöser CS_{fd-MR}

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 naheliegend, 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 Progammiersprache CHIP wäre der Austausch der Daten über die C-Schnittstelle der Programmiersprache CHIP notwendig. Die C-Schnittstelle stellt aber bei großen Datenmengen einen Engpass da 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 Datenmengen 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

Die Diskussion der Ergebnisse erfolgte im Kapitel "Ergebnisse und Bewertungen" anhand einiger ausgewählter Diagramme. Hier werden nun die Diagramme der einzelnen Planungssysteme nach den verwendeten Prozessoren in eigenständigen Unterkapiteln dargestellt.

Begonnen wird mit den Diagrammen des Planungssystems **ConTime 1.0**, welches die constraintlogische Programmiersprache CHIP der französichenFirma Cosytec nutzt.
Die ersten beiden Diagramme E.1 zeigen die Ergebnisse des Athlon-Prozessors mit 700 MHz Taktfrequenz und 128 MB Hauptspeicher. Zur besseren Sichtbarkeit des starken Anstiegs des Berechnungsaufwandes werden zwei Säulendiagramme genutzt. Im oberen Säulendiagramm wird die y-Achse beschränkt, damit die Werte des Lösungsprozesses mit zerlegten Domänen sichtbar werden. In dem unteren Säulendiagramm sind die Daten von beiden Varianten des Lösungsprozesses, von unzerlegten Domänen und von der zerlegten Domänen, insgesamt zu sehen. Zur Bestimmung der Tendenz und der Angabe einer Trendfunktion wurden die Ergebnisse als Liniendiagramm E.2 dargestellt. Hierbei erfolgte wieder die Unterteilung in ein Liniendiagramm für den Lösungsprozess mit zerlegten Domänen und ein gemeinsames Liniendiagramm mit zerlegten und unzerlegten Domänen. Die Tendenz des Bearbeitungsaufwandes wurde mit einer potentiellen Funktion genähert. Die nächsten vier Diagramme E.3, E.4 zeigen die Säulen- und Liniendiagramme mit begrenzter und vollständiger y-Achse für einen Pentium4-mobil-Prozesser mit 1,4 GHz Taktfrequenz und 256 MB Hauptspeicher. In den Säulendiagrammen E.3 sind die Daten sowohl für zerlegte und unzerlegte Domänen zu sehen, als auch in Kombination mit angebenden Wünschen für Zeitpunkte-, Räume- und Dozentenwünsche. Es ergeben sich vier Säulen im Säulendiagramm. In der ersten Säule ist der Berechnungsaufwand für zerlegte Domänen mit Wünschen, in der zweiten Säule der Berechnungsaufwand für zerlegte Domänen ohne Wünsche, in der dritten Säule die unzerlegten Domänen mit Wünschen und in der vierten Säule die unzerlegten Domänen ohne Wünsche dargestellt. In den Liniendiagrammen E.4 sind die Daten für zerlegte und unzerlegte Domänen und mit und ohne Beachtung der Wünsche zusehen. Zusätzlich wurde mit potenziellen und polynomen Funktionen versucht, den Trend zu nähern. Die polynomen Funktionen kommen nur bei den zerlegten Domänen zur Anwendung und sind nur im oberen Diagramm mit angegeben.

Als nächstes werden die Diagramme des Planungssystems **Kursplan 1.0**, welches den Constraintlöser für Multiressourcenprobleme CS_{fd-MR} aus dem gleichnamigen Kapitel dieser Arbeit nutzt, jeweils für die beiden oben genannten Athlon- und Pentium-Prozessoren vorgestellt. Hier wird der Aufwand für die Berechnung in Säulendiagrammen und Liniendiagrammen zusammen jeweils für den Athlon-Prozessor E.5 und für den Pentium-Prozessor E.6 dargestellt. In dem Säulendiagramm wird in der ersten Säule der Berechnungsaufwand mit Ressourcenwünschen, in der zweiten Säule mit Ressourcenwünschen/-präferenzen und in der dritten Säule mit beiden zusammen dargestellt. Im Liniendiagramm wird der Trend für die drei Varianten des Planungsprozesses jeweils mit potentiellen und mit polynomen Funktionen genähert.

Als letztes werden die Ergebnisse beider Planungssysteme in gmeinsamen Diagrammen dargestellt. Hierfür wurden Liniendiagramme (siehe Diagramme E.7) gewählt, weil sich hier auch die Tendenz der Abarbeitungszeiten gut ablesen lässt. Für den Vergleich wurden die Zeiten mit Wünschen auf den zerlegten und unzerlegten Domänen des Systems **ConTime 1.0** gewählt, die den Zeiten des Systems **Kursplan 1.0** mit der Beachtung der Wünsche und Präferenzen der Ressourcen gegenüber gestellt wurden. Da bei diesen Varianten in beiden Systemen der größte Planungsaufwand anfällt, sind sie gut vergleichbar. Alle Daten wurden auf einem Pentium-Prozessor ermittelt und werden mit einer reduzierten und vollständigen y-Achse dargestellt.

E.1 ConTime 1.0

AMD-Prozessor

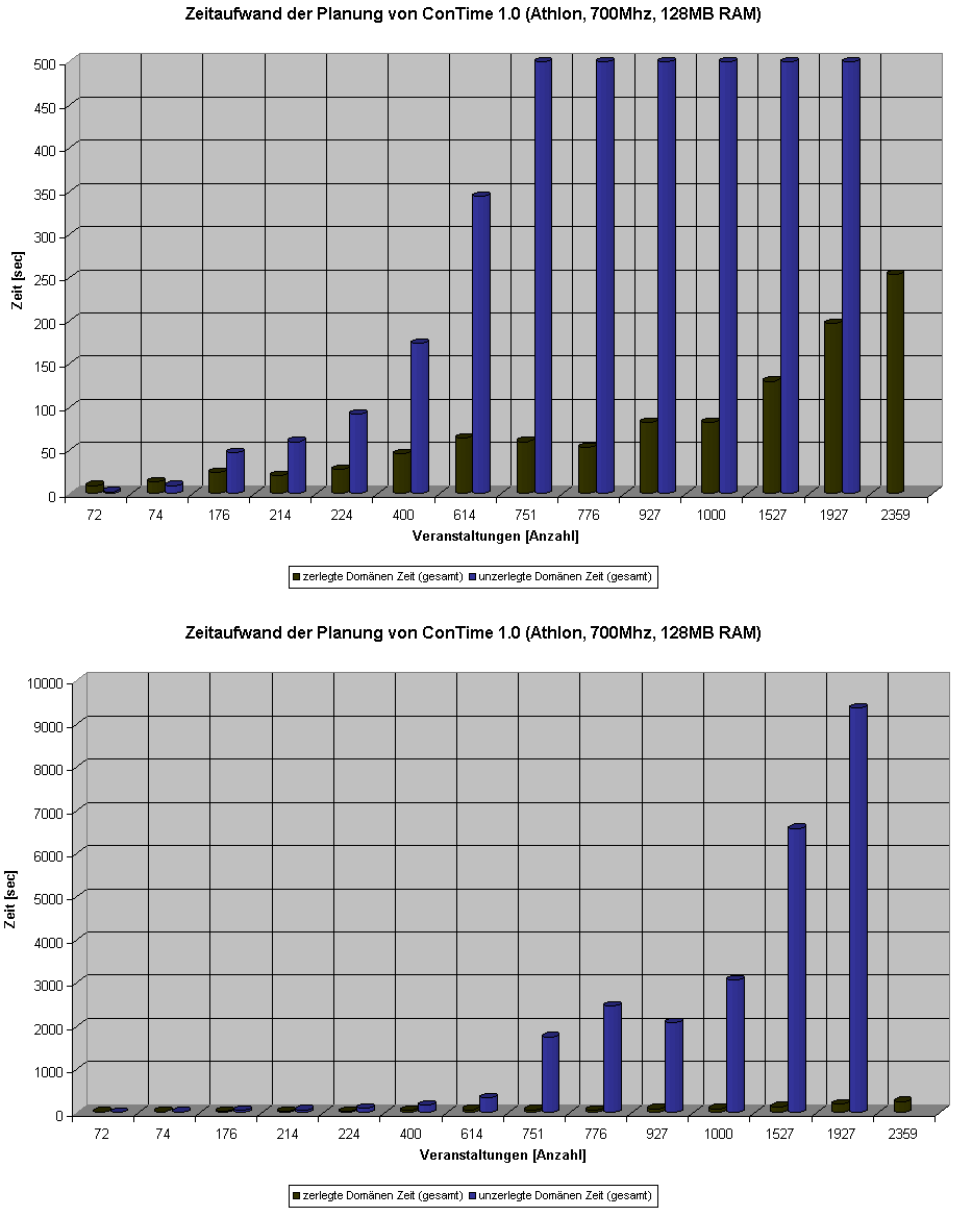

Abbildung E.1: Darstellung der Messergebnisse von ConTime 1.0 im Säulendiagramm mit eingeschränkter (oben) und maximaler (unten) Y-Achse

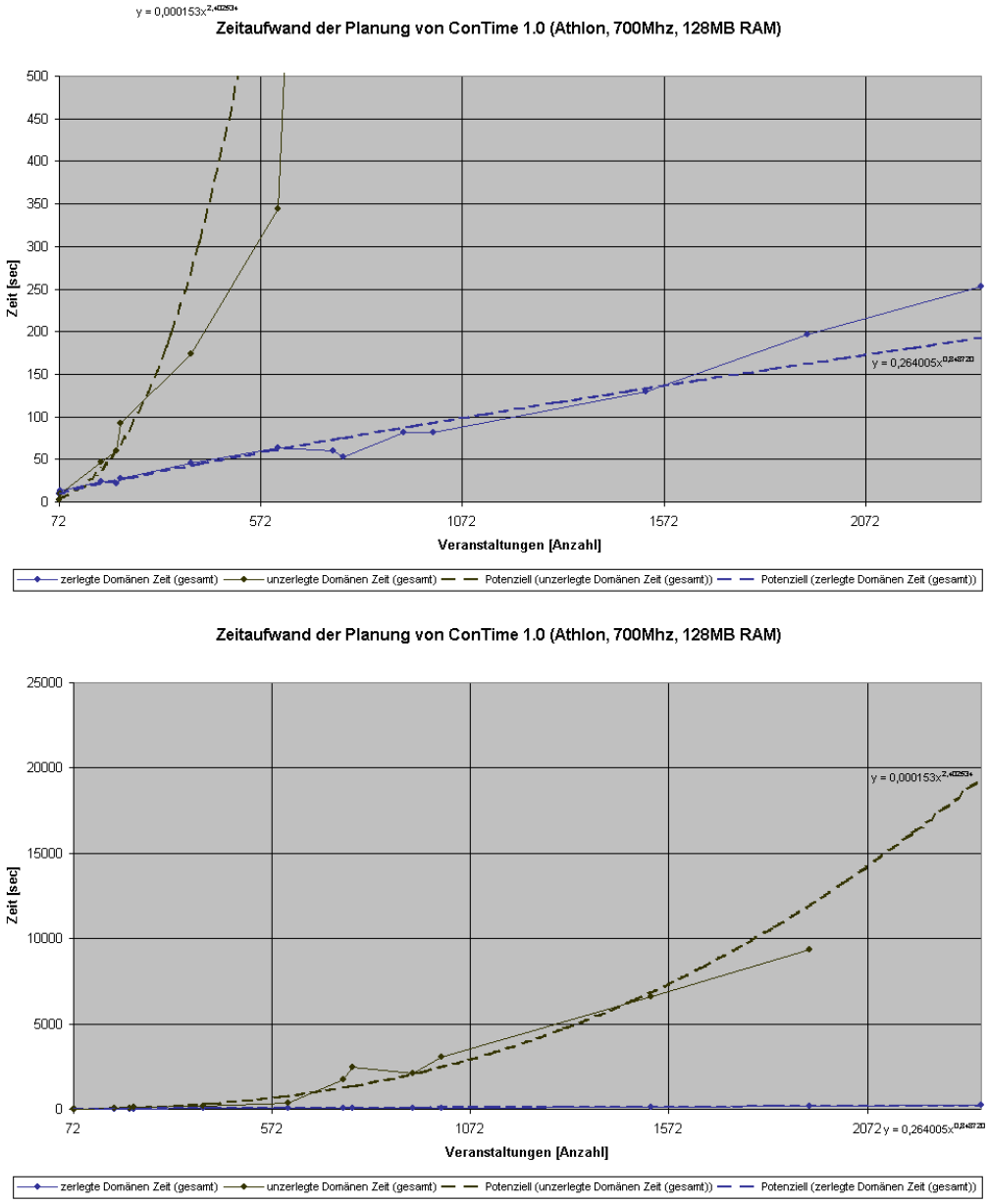

Abbildung E.2: Darstellung der Messergebnisse von ConTime 1.0 im Linienndiagramm mit eingeschränkter (oben) und maximaler (unten) Y-Achse

Pentium-Prozessor

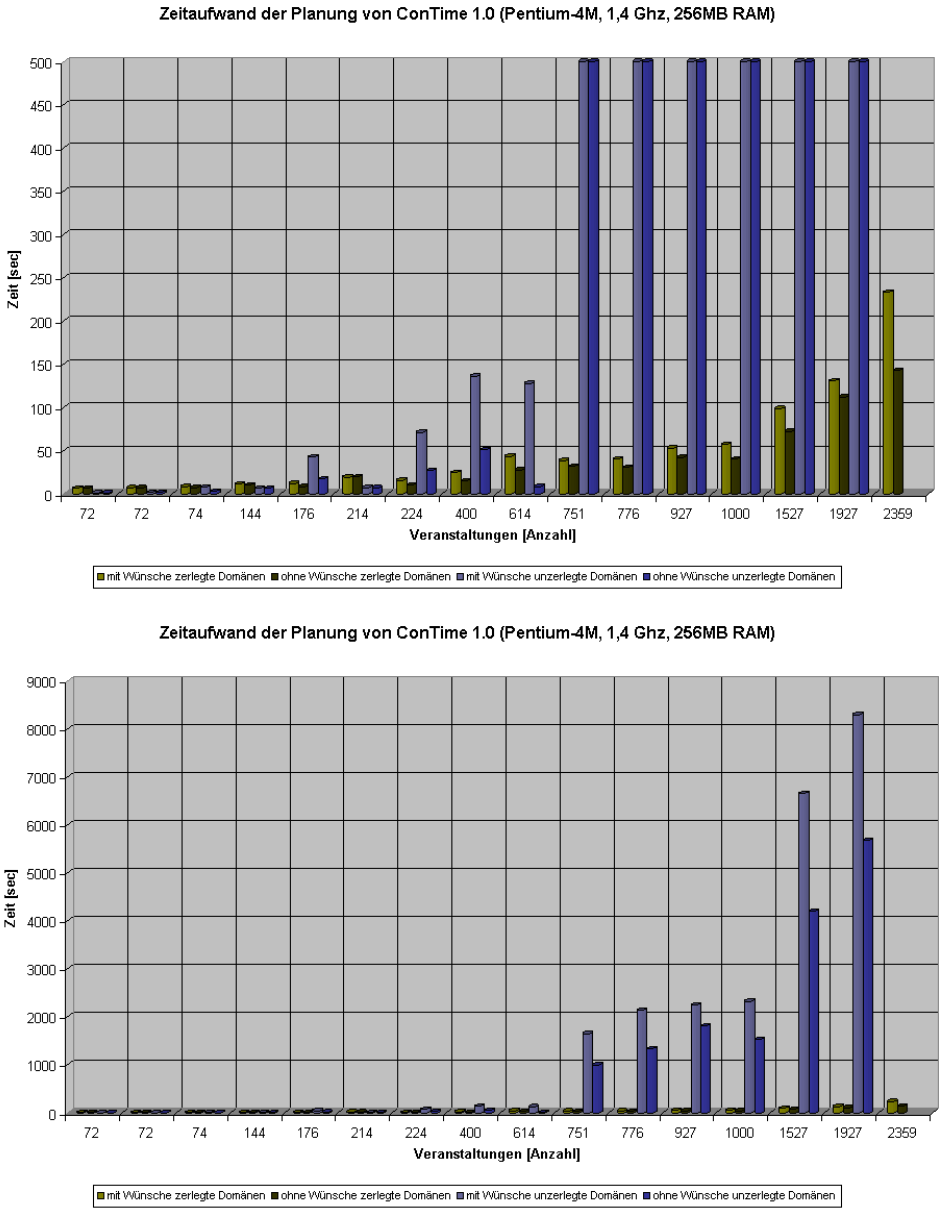

Abbildung E.3: Darstellung der Messergebnisse von ConTime 1.0 im Säulendiagramm mit eingeschränkter (oben) und maximaler (unten) Y-Achse

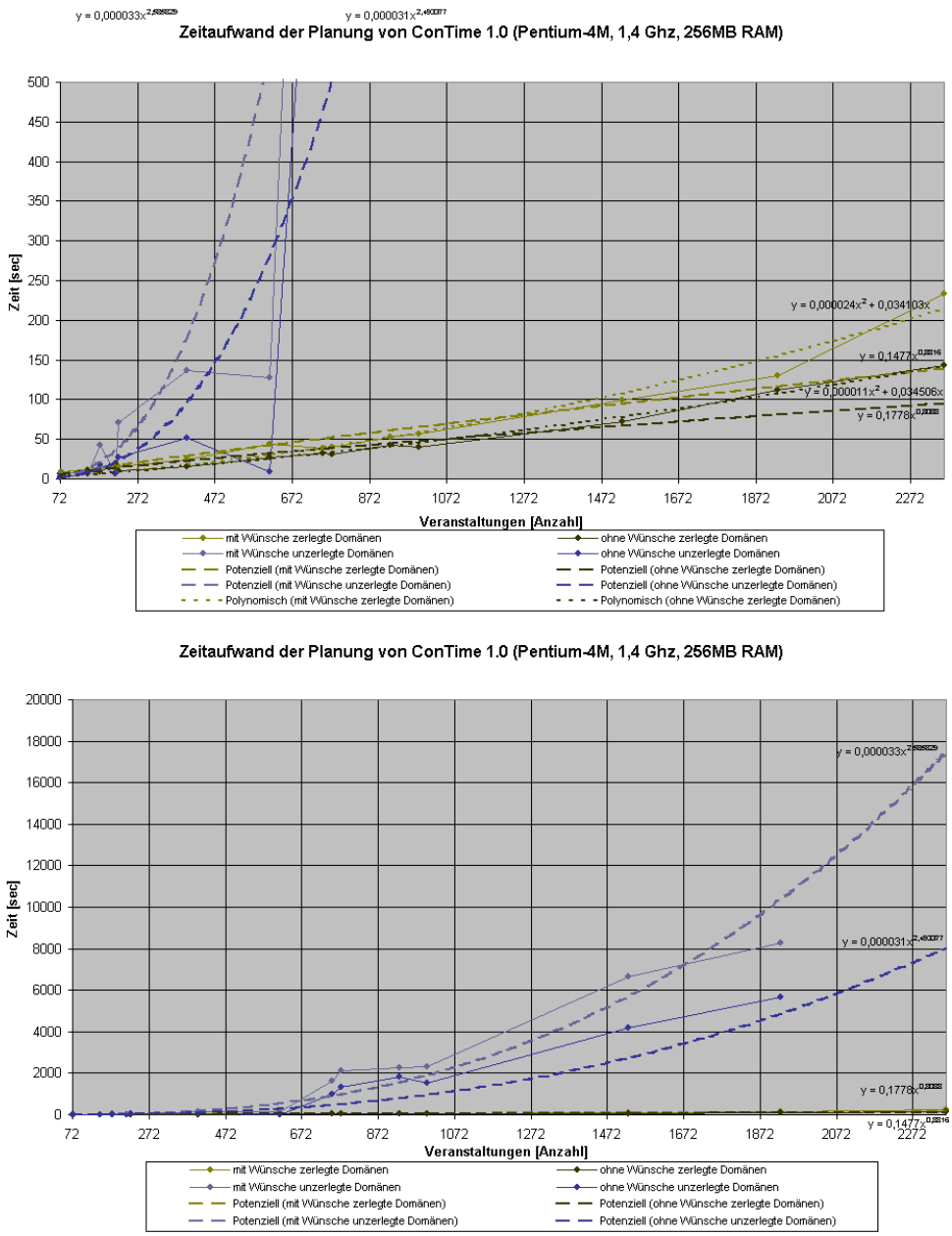

Abbildung E.4: Darstellung der Messergebnisse von ConTime 1.0 im Liniendiagramm mit eingeschränkter (oben) und maximaler (unten) Y-Achse

E.2 Kursplan 1.0

AMD-Prozessor

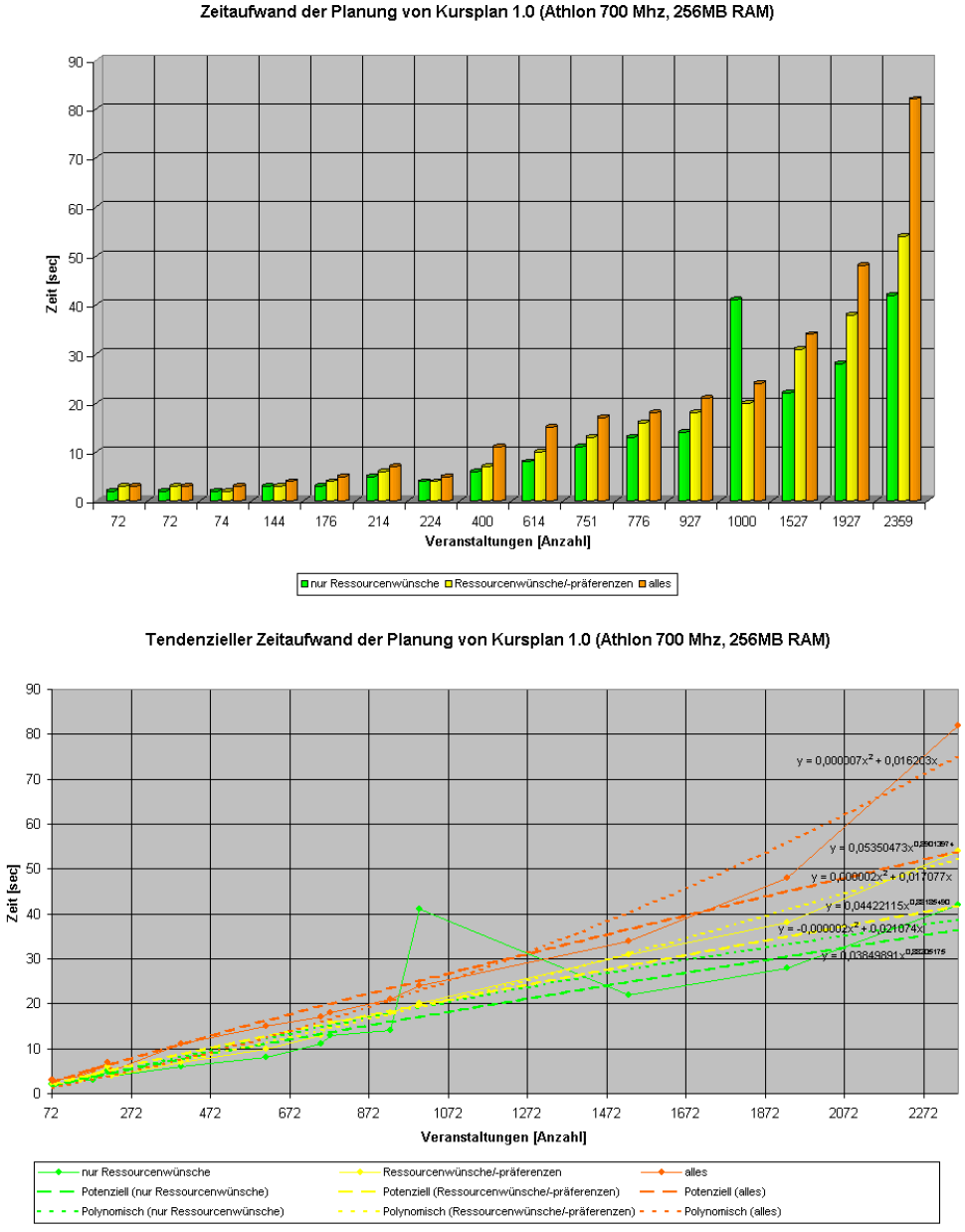

Abbildung E.5: Darstellung der Messergebnisse von Kursplan 1.0 (Athlon - Prozessor 700 Mhz, 256 MB RAM)

Pentium-Prozessor

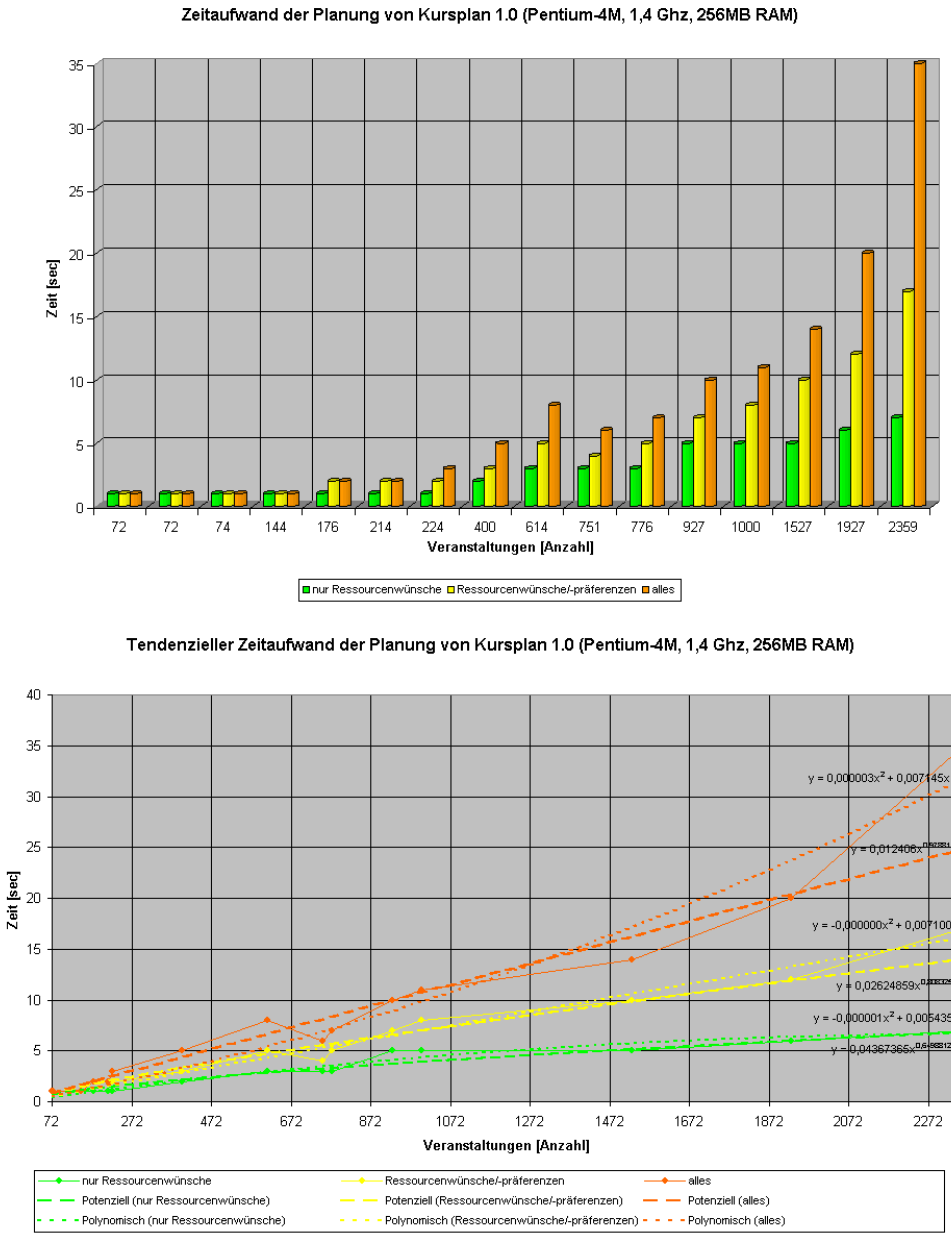

Abbildung E.6: Darstellung der Messergebnisse von Kursplan 1.0 (Pentium-4M - Prozessor, 1,4 Ghz, 256 MB RAM)

E.3 Vergleich von ConTime 1.0 und Kursplan 1.0

Pentium-Prozessor

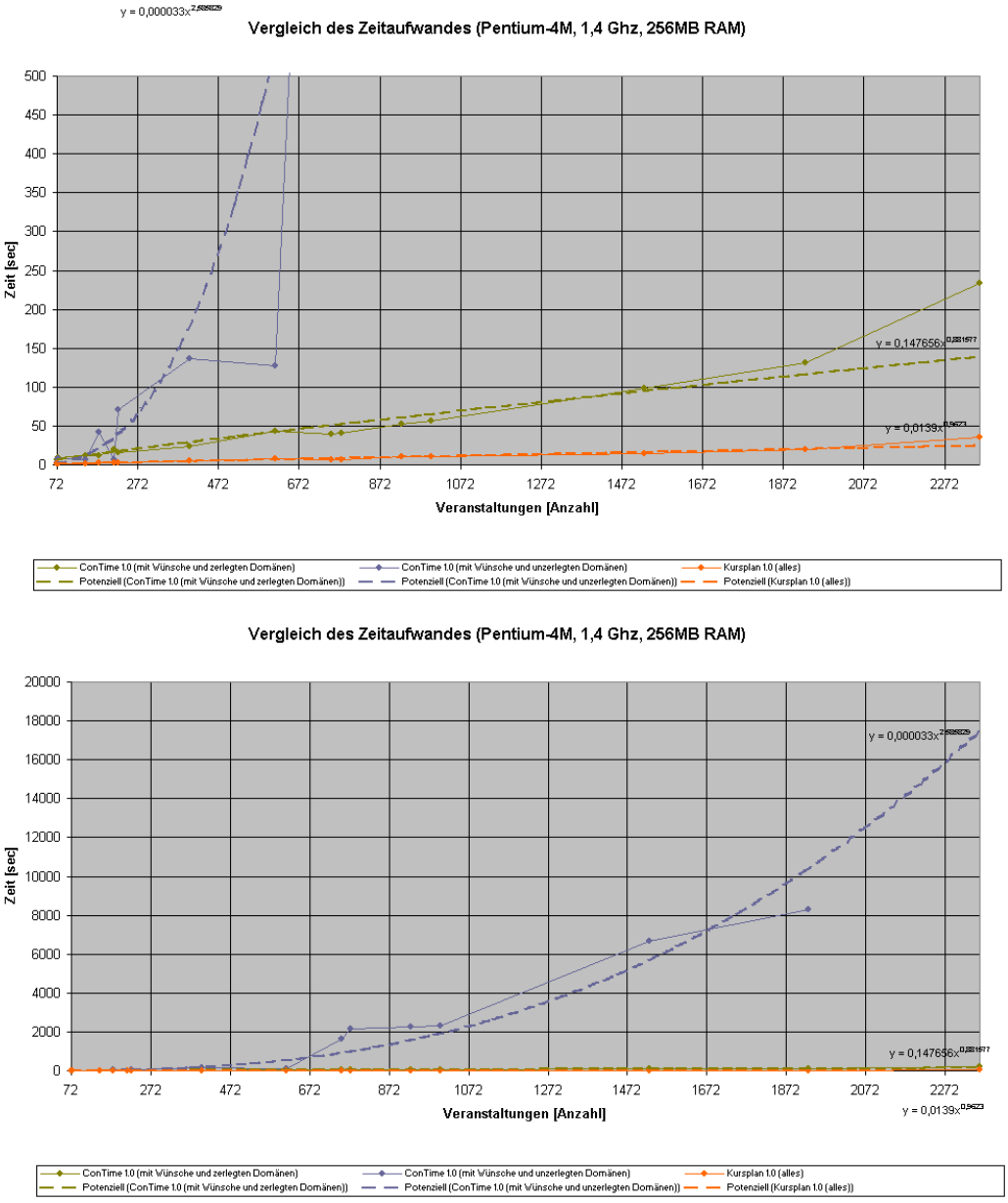

Abbildung E.7: Vergleich ConTime 1.0 und Kursplan 1.0 mit eingeschränkter (oben) und maximaler (unten) Y-Achse