Douglas Cunningham, Petra Hofstedt, Klaus Meer, Ingo Schmitt (Hrsg.): INFORMATIK 2015
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2015

Divide & Conquer – Komplexe Constraint-Probleme durch Aufteilen lösen

Peter Sauer[1] Denny Schneeweiß[1], Petra Hofstedt[1]

Abstract: Die Constraint-Programmierung bietet uns hervorragende Möglichkeiten um komplizierte Probleme zu modellieren und zu lösen. Manchmal sind die zu lösenden Probleme jedoch zu komplex, sodass sie mit den üblichen Mitteln der Constraint-Programmierung nicht in annehmbarer Zeit gelöst werden können. Eine Möglichkeit den Lösungsprozess für solche Probleme zu beschleunigen ist die Aufteilung in mehrere kleinere Teilprobleme. Nicht alle Probleme sind gleich gut geeignet durch Aufteilen schneller gelöst zu werden. Einige erfordern einen erheblichen Mehraufwand in der Modellierung, bei anderen ergibt sich eine geeignete Aufteilung nahezu von selbst. Anhand von zwei Beispielen aus der Praxis beschreiben wir mögliche Strategien um komplexe Constraint-Probleme durch Aufteilen schneller zu lösen.

Keywords: Constraint-Programmierung, CSP

1 Einleitung

In der Constraint-Programmierung werden Probleme mit Hilfe von Bedingungen (Constraints) modelliert und anschließend von einem geeigneten Constraint-Solver gelöst. Hierbei stehen dem Constraint-Solver neben allgemeinen Such- und Lösungsstrategien zur Beschleunigung des Lösungsprozesses zusätzlich sogenannte globale Constraints mit spezialisierten Algorithmen zu Verfügung. Da Constraint-Probleme jedoch sehr komplex sein können, ist die Berechnung von Lösungen in adäquater Zeit jedoch auch dann nicht gesichert. Eine weitere Möglichkeit den Lösungprozess zu beschleunigen, ist die Aufteilung des Constraint-Problems in Teilprobleme welche potentiell auch parallel behandelt werden können und die anschließende Kombination der Teillösungen. Dieses Vorgehen führt i.d.R. jedoch zu weiteren Constraints und weiterem Aufwand, um die Konsistenz der entstehenden Teillösungen garantieren zu können.

Unser Ziel ist es, geeignete Aufteilungen für schwere Constraint-Probleme zu finden, um eine Lösung in adequater Zeit bestimmen zu können.

Wir betrachten im Folgenden die Behandlung und Lösung komplexer Planungsprobleme durch ihre Aufteilung in Teilprobleme und deren Lösung. In Abschnitt 2 stellen wir für zwei Planungsprobleme jeweils deren Struktur, Umfang und mögliche Aufteilungen vor: Abschnitt 2.1 diskutiert die Planung von Prüfungsterminen an einer Universität, Abschnitt 2.2 skizziert ein Dienstplanungsproblem.

[1] Brandenburgische Technische Universität Cottbus – Senftenberg, Lehrstuhl Programmiersprachen und Compilerbau, Postfach 101344, 03013 Cottbus, {peter.sauer, schneden, petra.hofstedt}@b-tu.de

Da solche, aus der Aufteilung eines Planungsproblems resultierende, Teilprobleme i.d.R. nicht voneinander unabhängig sondern globalen Bedingungen unterworfen sind, ist eine nebenläufige oder parallele Behandlung und Lösung zwar möglich, aber mit weiteren Bedingungen und u.U. erheblichem Koordinationsaufwand zur Kombination der Teillösungen verbunden. Berechnet man beispielsweise Teildienstpläne für Zeiträume oder Mitarbeitergruppen, so müssen diese aufeinander abgestimmt werden. Mehrere separat arbeitende Constraint-Solver müssen also Teil- und Zwischenergebnisse austauschen und auf Widersprüche und Konflikte reagieren können.

Haben wir eine Aufteilung eignet sich als Framework zur Formalisierung und Lösung solcher Constraint-Probleme die *Verteilte Constraint-Programmierung* [YH00, Sa12]. Verteilte Constraint-Speicher halten dabei die Variablen und Constraints eines Teilproblems und errechnen hierfür zulässige (Teil-)Lösungen. Abhängigkeiten zwischen Teilproblemen werden in einer Hierarchie der Speicher abgebildet. Teil- und Zwischenergebnisse sowie Konflikte werden in Form von Constraints zwischen den Speichern entsprechend der Hierarchie ausgetauscht. In Abschnitt 3.1 präsentieren wir kurz verteilte und nebenläufige Constraint-Speicher. Anschließend illustrieren wir in Abschnitt 3.2 ihre Verwendung zur Lösung komplexer Constraint-Probleme durch Aufteilung.

2 Aufteilung in Teilprobleme

Nicht alle Probleme eignen sich gleich gut, um durch Aufteilen schneller gelöst zu werden. Bei der Planung von Prüfungsterminen an einer Universität liegt eine geeignete Aufteilung in Teilprobleme sehr nahe. Im Gegensatz dazu lässt sich für die Erstellung von Rahmendienstplänen so leicht keine geeignete Aufteilung in Teilprobleme finden. In den beiden folgenden Abschnitten spezifizieren wir die beiden genannten Probleme und deren Anforderungen genauer und stellen jeweils Strategien zu deren Aufteilung in mehrere Teilprobleme vor.

2.1 Planung von Prüfungsterminen an einer Universität

Das Problem der Prüfungsplanung an der BTU Cottbus – Senftenberg (am Zentralcampus) ist von vornherein hierarchisch strukturiert, so dass es sich für eine nebenläufige oder parallele Lösung hervorragend eignet. Im Folgenden stellen wir das Problem, seine Struktur und den Lösungsprozess detaillierter dar.

Problemstellung: Das hier beschriebene Problem der Prüfungsplanung umfasst die in Cottbus zentral geplanten (schriftlichen) Prüfungen. Dies betrifft insbesondere Prüfungen der Module der Bachelor-Studiengänge, da die Studienpläne in den Masterstudiengänge i.d.R. individuell von den Studierenden (in Absprache mit ihrem Mentor) zusammengestellt werden. Nicht berücksichtigt werden also Prüfungen in den Masterstudiengängen, die mündlich durchgeführt werden und deren Termine von den Prüfern oder Planern in einem zweiten Planungsschritt nach der zentralen Prüfungsplanung festgelegt werden.

Die zentrale Prüfungsplanung umfasst ca. 500 Prüfungen über 36 Studiengänge je Semester. Der verfügbare Planungszeitraum gliedert sich in zwei zweiwöchige Zeiträume (im Folgenden Zeitraum A und B genannt), die am Anfang bzw. Ende der vorlesungsfreien Zeit bzw. der sogenanten Semesterpause liegen. Es stehen 60 Räume bzw. Hörsäle unterschiedlicher Größen zwischen 10 und 500 Plätzen zur Verfügung.

Für die Festlegung der Prüfungstermine gibt es diverse Nebenbedingungen und Wünsche, die bei der Planung zu berücksichtigen sind. Dies umfasst nicht nur notwendige Raumgrößen oder die Prüfungsdauer (zzgl. Vor- und Nachbereitungszeiten), sondern auch potentielle Kollisionen von Prüfungen innerhalb von Studiengängen, die Einplanung von freien Tagen zwischen Prüfungen eines Studiengangs bis hin zu individuellen Wünschen einzelner Prüfer.

Strukturierung des Lösungsprozesses: Eine Aufteilung des Lösungsprozesses des Planungsproblems in drei Phasen bietet sich an:

1. Zuordnung der Prüfungen auf konkrete Prüfungstage („Tagesplanung“),
2. Zuordnung der Prüfungen auf konkrete Zeitblöcke der im vorherigen Schritt fixierten Tage („Blockplanung“) und
3. Zuordnung von Räumen, wobei Tag und Block der Prüfungen schon feststehen („Raumplanung“).

Eine Zerlegung des Problems in dieser Form ist notwendig, da das Gesamtproblem sehr komplex ist und die Lösung damit in vertretbarer Zeit nicht garantiert werden kann. Auch unterstützt eine solche Aufteilung die Planung von Problemen gleicher Struktur mit deutlich größeren Umfang.

Gleichzeitig ist zu berücksichtigen, dass eine solche Problemzerlegung i.d.R. zu einer Backtracking-artigen Berechnung führen kann. Das heißt, eine bereits erzeugte Lösung für ein Teilproblem der Ebene i (z.B. Ebene $i = 1$, d.h. Tagesplanung) kann in höheren Ebenen $j > i$ (z.B. $i = 2$, d.h. Zeitplanung) zu einem Konflikt führen, so dass auf Ebene i eine neue Lösung erzeugt werden muss. Sind die Bedingungen im Planungsproblem schwach und die Domänen ausreichend groß, können durch die Aufteilung des Lösungsprozesses Lösungen u.U. ohne Backtracking erzeugt werden.

Constraints für unser Problem (nachfolgend detaillierter beschrieben) können jeweils eindeutig einer Planungsphase zugeordnet werden. Das heißt, die Zerlegung des Problems in Phasen impliziert keine zusätzlichen Constraints. Mit dieser Aufteilung gehen durch den phasenweisen Lösungsprozess auch keine Lösungen verloren. Eine solche Aufteilung kann allerdings nicht für alle Probleme gefunden werden, was wir in Abschnitt 3.2 diskutieren.

Wir diskutieren nun die einzelnen Lösungsphasen und damit verbundenen Constraint-Probleme.

(1) Tagesplanung

Die Tagesplanung regelt die Zuordnung aller Prüfungen auf konkrete Tage im Planungszeitraum. Wichtige komplexe Constraints auf dieser Ebene sind durch die Zuordnung von Prüfungen zu Fachsemestern der einzelnen Studiengängen gegeben: Da es sich bei den zu planenden Prüfungen um Bachelor- und damit häufig um Grundstudiumsmodule (wie Mathematik-, BWL- oder Programmierungsgrundlagen) handelt, sind diese oft mehreren, teilweise auch sehr vielen Studiengängen gleichzeitig zugeordnet.

Wir berücksichtigen folgende Constraints:

- Für jedes Fachsemester eines Studiengangs (im Folgenden "FS-Studiengang") darf höchstens eine Prüfung pro Tag stattfinden. Variante: Es muss mindestens ein (oder x) freie Tage zwischen zwei Prüfungen eines FS-Studiengangs liegen.
- Prüfungen eines FS-Studiengangs dürfen nicht mit Prüfungen des gleichen Studiengangs des vorangehenden Studienjahrs kollidieren (Berücksichtigung von Studierenden, die Prüfungen wiederholen müssen).
- Nur Prüfungen mit dem Wunsch 'Sonnabend', sollen auch am Sonnabend geschrieben werden. Prüfungen mit dem Wunsch Zeitraum A oder Zeitraum B (s. oben) sollen auch in diesem Zeitraum stattfinden.

Einige dieser Constraints können in Varianten des Problems auch als Optimierungskriterium eingesetzt werden. Beispielsweise kann die Maximierung der Anzahl freier Tage zwischen Prüfungen eines FS-Studiengangs angestrebt werden.

Eine weitgehend gleichmäßige Verteilung aller Prüfungen über den gesamten Zeitraum erleichtert die Lösungsfindung in den Ebenen (2) und (3) (Block- und Raumplanung).

(2) Blockplanung

Auf der Ebene der Blockplanung gehen wir davon aus, dass die Prüfungen (in Ebene (1), d.h. Tagesplanung) bereits konkreten Tagen zugeordnet wurden. Nun werden konkrete Zeitblöcke fixiert.

Jeder Prüfungszeitraum, d.h. A und B, umfasst 12 Tage (2 Wochen ohne Sonntag), täglich jeweils von 7:30 bis 19:00 Uhr. Übliche Prüfungsdauern liegen zwischen 60 und 180 Minuten, wobei Zeiten der Vor- und Nachbereitung hinzugerechnet werden müssen. Jeder Prüfungstag wird in 23 Böcke zu je 30 Minuten aufgeteilt, um eine gute Ausnutzung von Zeiten und Räumen gewährleisten zu können.

Da die Prüfungen bereits konkreten Tagen zugeordnet wurden, sichern die Constraints dieser Ebene vor allem die Bereitstellung von hinreichend langen, zusammenhängenden Zeitblöcken für jede Prüfung ab.

Auch für Ebene (2) Blockplanung gilt: Eine weitgehend gleichmässige Verteilung aller Prüfungen über den betreffenden Prüfungstag erleichtert die Lösungsfindung bzw. Planung

auf der nächsten Ebene (3) Raumplanung. Problemvarianten erlauben hierbei aber noch eine Bevorzugung bestimmter Blöcke, so wird — soweit möglich — der erste und letzte Prüfungsblock vermieden.

(3) Raumplanung

Die Raumplanung geht von einer konsistenten Zuordnung von Tagen und Zeiträumen je Prüfung aus und ordnet den Prüfungen Räume zu.

Constraints dieser Ebene berücksichtigen folgende Bedingungen:

- Die Summe der Plätze soll mindestens der der erwarteten Teilnehmer entsprechen.
- Gleichzeitig soll die Platzanzahl die Teilnehmerzahl nicht um Größenordnungen übertreffen.
- Die Einplanung mehrerer Räume für eine Prüfung wird unterstützt (notwendig bei Prüfungen mit besonders hohen Teilnehmerzahlen). Werden mehrere Räume für eine Prüfung gleichzeitig gebucht, so gilt:
  - Hörsäle sollen nicht mit kleinen Räumen kombiniert werden.
  - Es sollen so wenige Räume wie möglich pro Prüfung gewählt werden.
- Räume dürfen nicht zu einer Zeit mehrfach verplant werden.

Wurden in der Raumplanung für die gegebene Zuordnung von Tagen und Zeiträumen passende Räume gefunden, so haben wir eine Gesamtlösung gefunden, welche dem Nutzer präsentiert werden kann.

2.2 Dienstplanung einer Feuerwehrleitstelle

Problemstellung: Dieses Problem umfasst die automatische Rahmendienstplanung im Schichtbetrieb am Beispiel der *Regionalleistelle Berlin-Brandenburg* in Cottbus, die für das gesamte Gebiet Südbrandenburgs und den Flughafen Berlin-Brandenburg International mit insgesamt ca. 650.000 Einwohnern zuständig ist.

Ein Rahmendienstplan wird viele Monate im Voraus erstellt und ordnet die Mitarbeiter tageweise in verschiedene Schichten ein, wobei planbare Fehlzeiten wie Urlaub oder Weiterbildungen noch nicht berücksichtigt werden. Dies geschieht später im Grunddienstplan, für den der Rahmendienstplan die Basis bildet. Kurzfristige Änderungen durch krankheitsbedingte Fehlzeiten oder Schichttausch werden in der Tagesdienstplanung berücksichtigt, welche den letzten Schritt der Dienstplanung bildet. An dieser Stelle soll nur die Rahmendienstplanung betrachtet werden, da diese langfristig erstellt werden muss und durch den großen Betrachtungszeitraum und die Vielzahl von Constraints eine hohe Komplexität besitzt. Die Erstellung der weiterführenden Grund- bzw. Tagespläne hat dann eine wesentlich geringere Komplexität, da die Erfüllung der wichtigsten Rahmenbedingungen bereits vom Rahmendienstplan gewährleistet wird.

Planungsanforderungen

Für die Rahmendienstplanung müssen eine ganze Reihe organisatorischer Anforderungen der Leitstelle wie auch arbeitsrechtliche Rahmenbedingungen und arbeitswissenschaftliche Erkenntnisse beachtet werden, die wir im Anschluss näher erläutern werden. Ziel der Dienstplanung ist die Sicherstellung der vollen Einsatzbereitschaft der Leitstelle durch notwendige Personalstärken in den verschiedenen Schichten. Um diese auch langfristig zu garantieren, muss die Dienstplanung jedoch auch auf die physiologischen, sozialen und psychischen Auswirkungen der Arbeitsbelastung im Schichtbetrieb Rücksicht nehmen und diese Faktoren in die Planung mit einfließen lassen.

In der Regionalleistelle existieren insgesamt neun Schichttypen. Neben Frühschicht (6:00-14:00 Uhr), Spätschicht (14:00-22:00 Uhr) und Nachtschicht (22:00-6:00 Uhr) gibt es drei weitere sogenannte Springerschichten die parallel dazu laufen und zur Abdeckung von Fehlzeiten durch Krankheit oder Urlaub dienen. Fällt ein Mitarbeiter in den normalen Schichten aus, kann ein Springer direkt übernehmen. Weiterhin existieren noch zwei Tagesdienstschichten die 8:00-16:00 Uhr bzw. 11:00-19:00 Uhr laufen sowie eine Praktikumsschicht (7:00-18:00 Uhr) für Ausbildungstätigkeiten und das Training von Rettungstätigkeiten.

Constraints: Die grundlegenden Constraints für die Dienstplanung ergeben sich aus einer Bedarfstabelle der Leitstelle (siehe Tabelle 1), in der für jeden Schichttyp und Wochentag die zur Einsatzbereitschaft notwendige Mitarbeiterzahl (Schichtstärke) verzeichnet ist. Die Leitstelle verfügt über ca. 40 Mitarbeiter.

	Mo	Di	Mi	Do	Fr	Sa	So
Frühschicht	7	7	8	8	8	7	7
Spätschicht	6	6	7	6	7	6	6
Nachtschicht	5	5	5	5	5	5	5
Praktikum	1	1	1	1	1	0	0
Tagesdienst 1	1	1	0	0	0	0	0
Tagesdienst 2	0	0	0	0	0	0	0
Springer Früh	2	2	2	2	2	2	2
Springer Spät	2	2	2	2	2	2	2
Sprinter Nacht	2	2	2	2	2	2	2

Tab. 1: Zur Einsatzbereitschaft notwendige Schichtstärken der Regionalleitstelle Lausitz für eine Belegschaft mit 38 Mitarbeitern [Hä13].

Die Dienstplanung muss diese Schichtstärken für die jeweiligen Tage garantieren, wobei eine Reihe weiterer Constraints hinzukommen, die sich aus den gesetzlichen Regelungen zum Arbeitsschutz ergeben:

- Der Gesetzgeber gibt im Arbeitszeitgesetz für Schichtarbeit vor, dass diese acht Stunden in der Regel nicht überschreiten darf.

- Weiterhin müssen Ruhezeiten von mindestens elf Stunden zwischen zwei Schichten eingehalten werden, was Auswirkungen auf die Plangestaltung hat. Auf eine Nachtschicht dürfen daher keine Früh- oder Spätschicht folgen, da hierbei die Ruhezeit unterschritten wird.
- Die Länge von Blöcken aufeinanderfolgender Schichten des gleichen Typs muss mindestens zwei und höchstens vier Tage bzw. Nächte dauern, um eine negative Veränderung des Tag-Nacht-Rhythmus zu verhindern.
- Eine Vorwärtsrotation beim Schichtwechsel von Früh- über Spät- nach Nachtschicht hat sich als vorteilhaft erwiesen [Be05].
- Weiterhin sollen für die Tage am Wochenende bestenfalls freie Tage, ansonsten aber möglichst gleiche Schichttypen geplant werden.
- Außerdem muss der Freizeitausgleich möglichst geblockt erfolgen.
- Nach Schichten an Sonn- und gesetzlichen Feiertagen müssen spätestens zwei bzw. acht Wochen später Ausgleichstage gegeben werden.
- Weiterhin dürfen an den Wochenenden keine Praktika stattfinden.

Lösungsansatz: Die Dienstplanerstellung erfolgt wochenweise und rollierend. Dabei wird ein Dienstplan für einen einzigen Mitarbeiter über eine Anzahl von Wochen erstellt. Für jeden weiteren Mitarbeiter wird diese Einzelplanung dann jeweils um eine Woche verschoben verwendet. Dies garantiert eine faire Verteilung der Arbeitslast auf alle Mitarbeiter. Für die Leitstelle in Cottbus ergibt sich damit ein angestrebter Planungszeitraum von ca. 40 Wochen, welcher allerdings aufgrund der Problemkomplexität nicht in vertretbarer Zeit berechnet werden kann.

Daher muss das Gesamtproblem in kleinere Teile unterteilt werden, die sich in akzeptabler Zeit lösen lassen. Für die Rahmendienstplanung eignet sich die Unterteilung in mehrere Wochenblöcke. Vor allem Constraints, die sich lediglich auf Wochentage beziehen, lassen sich dadurch lokalisieren. Beispiele hierfür sind die Relegung, dass an Wochenenden stets gleiche Schichttypen bzw. Freizeitausgleich geplant werden soll und das dort keine Praktikumsschichten stattfinden.

Für andere Constraints gilt dies leider nicht. Die Anforderungen zur Sicherstellung der Schichtstärken und die meisten gesetzlichen Rahmenbedingungen müssen nicht nur lokal für die einzelnen Problemteile, sondern global über den gesamten Planungszeitraum erfüllt sein. Zum Beispiel darf die mittlere Arbeitsbelastung über den gesamten Planungsbereich täglich acht Stunden nicht überschreiten. Ausgleichstage für Feiertagsarbeit müssen in einem Zeitraum von zwei bzw. acht Wochen, also ggf. über Teilproblemgrenzen hinweg, gewährt werden.

Im ersten Schritt des Lösungsprozesses muss zunächst lokale Konsistenz für die Teilprobleme hergestellt werden, sodass Werte-Belegungen, die zu keiner Lösung im Teilproblem

führen, eliminiert werden. Um das Gesamtproblem zu lösen muss anschließend globale Konsistenz erreicht werden. Dafür sind zwei Verfahren geeignet:

Die erste Variante ist eine Lösungssuche mittels verteiltem Backtracking über den Teilproblemen, wobei die miteinander kommunizierenden Agenten versuchen, sich auf eine Gesamtlösung zu einigen.

Eine weitere Möglichkeit besteht in der Redefinition von Constraints, die sich nicht natürlich lokalisieren lassen, in schärfere Constraints auf lokaler Ebene. Bei einer Aufteilung des Gesamtproblems von vierzig Wochen in vier Blöcke mit je zehn Wochen kann beispielsweise die lokale Schichtstärke auf den Faktor 0.25 bis 0.3 reduziert werden. Durch diese Verschärfung können Gesamtlösungen verloren gehen. Ist der Lösungsraum jedoch ausreichend groß, können genügend Lösungen übrig bleiben, sodass das Verfahren eine effiziente Lösungsstrategie bietet. Erfahrungen aus der Anwendungsdomäne helfen beim Finden geeigneter schärferer Contraints.

3 Verteilte Lösung von CSPs

Einen geeigneten Formalismus in mehrere Teilprobleme aufgeteilte Probleme zu lösen bietet die verteilte Constraint-Programmierung. Diese wird im folgenden Abschnitt näher erläutert. In Abschnitt 3.2 beschreiben wir, wie mit Hilfe der verteilten Constraint-Programmierung die oben beschriebenen Probleme schneller gelöst werden können.

3.1 Verteilte Constraint-Programmierung

Die verteilte Constraint Programmierung [YH00, Fa06] (engl. *Distributed Constraint Programming*) befasst sich mit der nebenläufigen und parallelen Lösung von CSPs (Constraint Satisfaction Problems).

Constraint Satisfaction Problems: Ein *CSP* besteht aus einer Menge $\{x_1, x_2, \ldots, x_n\}$ von n Variablen, welche jeweils Werte aus gegebenen *Domänen* $D_1, D_2, \ldots, D_n$ annehmen können, sowie einer Menge von Constraints über diesen Variablen. Dabei beschreiben die Constraints Bedingungen, welche als Relationen modelliert werden können. Ein Constraint $c(x_{c,1}, \ldots, x_{c,j})$ ist eine Teilmenge aus dem kartesischen Produkt $D_{c,1} \times \ldots \times D_{c,j}$ und beschreibt alle Kombinationen von Belegungen für die Variablen $x_{c,1}, \ldots, x_{c,j}$ unter denen das Constraint c gilt. Eine *Belegung* ist dabei die Zuordnung von Werten $v_k \in D_k$ zu den entsprechenden Variablen x_k. Eine Belegung für die Variablen des CSPs unter der das CSP (d.h. alle Constraints des CSPs) erfüllt ist, wird als *Lösung* des CSP bezeichnet. Ein CSP ist *konsistent*, wenn mindestens eine Lösung existiert, anderenfalls ist es *inkonsistent*.

Das Lösen eines CSP, also die Suche nach einer Belegung für alle Variablen, unter der alle Constraints erfüllt sind, ist im allgemeinen NP-vollständig. Eine Vielzahl von Algorithmen, Techniken und Heuristiken zielen daher auf eine Beschleunigung des Lösungsprozesses, um CSPs handhabbar zu machen ([RvBW06]).

Distributed CSPs: Bei einem *verteilten CSP* (engl. *distributed CSP*) werden die Variablen und Constraints auf mehrere Agenten verteilt. Auf der Basis eines nachrichtenorientierten Kommunikationsmodells versuchen die Agenten gemeinsam konsistente Lösungen für das CSP zu berechnen.

Während in der allgemeinen Definition verteilter CSPs zunächst jedem Agenten auch mehrere Variablen zugeordnet werden, wird dies i.d.R. bei der Betrachtung konkreter Lösungsalgorithmen, wie synchrones oder asynchrones Backtracking, eingeschränkt, sodass jedem Agenten genau eine Variable zugeordnet wird.

Dies impliziert jedoch, dass globale Constraints, welche typischerweise über dedizierte Algorithmen effizient gelöst werden, auf binäre Constraints zwischen den Agenten herrunter gebrochen werden müssen. In unserer Implementierung hält ein Agent mehrere Variablen und wir können globale Constraints auf diesen weiterhin mit ihren dedizierten Algorithmen lokal innerhalb eines Agenten bearbeiten.

Yokoo [YH00] grenzt die „verteilte Constraint-Programmierung“ von den echt parallelen Contraint-Lösungsmethoden („parallel/distributed processing“) ab, deren hauptsächliche Motivation Effizienz ist. Als Motivation für die verteilte Constraint-Programmierung führt er stattdessen an, dass hier bereits eine natürliche Verteilung des Problems oder des Wissens auf verschiedene Agenten a priori vorliegt.

Als Beispiel für Probleme mit inhärenter Verteilung auf Agenten nennt Yokoo insbesondere Multiagenten-Truth-Maintenance-Systeme, wobei die Konsistenz oder Lösungssuche auf mehrstufigen Verhandlungsprotokollen basiert. Es wird auch ein Beispiel eines Dienstplanungsproblems genannt, allerdings besteht dieses ebenfalls von vornherein aus klar abtrennbaren Teilproblemen in Form von weitgehend unabhängig planbaren Dienstplänen unterschiedlicher Abteilungen eines Betriebes.

Verallgemeinerte verteilte CSPs: Wir zielen darauf, Constraint-Probleme hinsichtlich ihrer natürlichen Verteilungsmöglichkeiten zu untersuchen und die Idee der verteilten Constraint-Programmierung gezielt zur Effizienzsteigerung einzusetzen. Sind natürliche „Sollbruchstellen“ in Constraint-Problemen nicht offensichtlich, so können Algorithmen der verteilten Constraint-Programmierung dennoch eingesetzt werden und zu Effizienzgewinnen führen.

Hierfür abstrahieren wir von den von Yokoo konzipierten Agenten und ordnen jedem Constraint-Agenten einen Constraint-Speicher zu. Jeder Agent bzw. Speicher a verwaltet eine Menge von Constraint-Variablen V_a und eine Menge von Constraints C_a. Ein solches Constraint C_a wird daher i.d.R. nicht nur eigene Variablen („interne Variablen“) sondern auch Variablen anderer Speicher („externe Variablen“) umfassen, die dem Agenten bekannt sind, jedoch nicht von ihm verwaltet werden, d.h. zwar gelesen, aber nicht geändert werden können.

Im Unterschied zu Yokoos Agenten besitzt ein Speicher also mehrere Variablen, was uns die schnelle Anwendung globaler Constraints innerhalb eines Speichers erlaubt. Yokoo

selbst diskutiert Agenten mit mehreren Variablen nur kurz und weist darauf hin, dass sein Ansatz (eine Variable pro Agent) keine Einschränkung darstellt. Agenten mit mehreren Variablen können dadurch abgebildet werden, dass mehrere Agenten auf einer Maschine laufen und schnell mit einander kommunizieren können.

Die Agenten stellen für ihre Constraint-Probleme Konsistenz her und erzeugen Teillösungen. Um die durch Abhängigkeiten zwischen den Teilproblemen auftretenden Konflikte zu lösen, folgen die Agenten einer Hierarchie.

In der Hierarchie höher liegende Speicher besitzen ein *Vorschlagsrecht* für Teillösungen, d.h. insbesondere für interne Variablen, die aus Sicht anderer – niedrigliegender Speicher – als externe Variable an deren Teilproblemen beteiligt sind. In der Hierarchie niedriger stehende Speicher haben ein *Vetorecht* und können bei auftretenden Inkonsistenzen die ihnen vorgeschlagene Teillösungen ablehnen. Die Kommunikation in unserem Modell erfolgt ebenfalls nachrichtenbasiert und unser Lösungsverfahren ist stark durch das von Yokoo entwickelte verteilte Backtracking inspiriert. Eine detaillierte Beschreibung des Modells findet sich in [Sa12].

3.2 Zerlegung und Lösung von Constraint-Problemen

Die Aufteilung eines Constraint-Problems in Teilprobleme ist stark von der Anwendungsdomäne abhängig. Mit der Zerlegung ergibt sich teilweise auch direkt eine Hierarchie der Constraint-Agenten/Speicher. Wir skizzieren im Folgenden Aufteilungen für die in Abschnitt 2 beschriebenen Constraint-Probleme. Beide Probleme sind ohne eine Aufteilung in mehrere Teilprobleme nicht in akzeptabler Zeit lösbar.

Prüfungsplanung: Bei der Prüfungsplanung ergibt sich eine erste mögliche Zerlegung direkt aus der Anwendung. Zunächst müssen die einzelnen Prüfungen einem Tag zugeordnet werden. Hierbei sind die in Abschnitt 2.1 beschriebenen Constraints zu beachten. Haben wir eine gültige Zuordnung von Prüfungen zu Tagen, können den Prüfungen Zeitblöcke zugeordnet werden. Mit dem Ergebnis dieser Blockplanung können wir anschließend jeder Prüfung einen oder mehrere geeignete Räume zuordnen.

Es bieten sich also drei Constraint-Speicher an: Jeweils einer für Tages-, Block- und Raumplanung. Die Tagesplanung schlägt ihre Teillösungen der Blockplanung vor, welche wiederum ihre Lösungen an die Raumplanung weiterreicht. Daraus ergibt sich die folgende Hierarchie: Tagesplanung $>$ Blockplanung $>$ Raumplanung.

Die Blockplanung und die Raumplanung besitzen jeweils ein Vetorecht und können bei auftretenden Inkonsistenzen erhaltene Teillösungen ablehnen. Wird eine Teillösung abgelehnt, muss eine neue Lösung für das entsprechende Problem bestimmt werden. Unser Lösungsverfahren entspricht also einem verteilten Backtracking. Hierbei werden jedoch keine einzelnen Variablenbelegungen widerrufen, sondern komplette Lösungen eines Constraint-Problems.

Ein möglicher Ablauf der Lösungsfindung könnte wie in Abbildung 1 dargestellt aussehen. Hier wird zunächst von der Tagesplanung ein Tagesplan (TP 1) errechnet und an die Blockplanung gegeben. Diese errechnet daraus eine Zuordnung von Prüfungen auf Zeitblöcke (BP 1.1) und schlägt diese der Raumplanung vor. Diese versucht zum gegebenen Tages- und Blockplan passende Räume zuzuweisen (RP 1.1.1). Dies schlägt allerdings fehl und so muss eine neue Blockplanung (BP 1.2) errechnet werden, welche erneut der Raumplanung vorgeschlagen wird. Diesmal existiert eine passende Raumbelegung und wir haben eine Lösung für das Gesamtproblem gefunden.

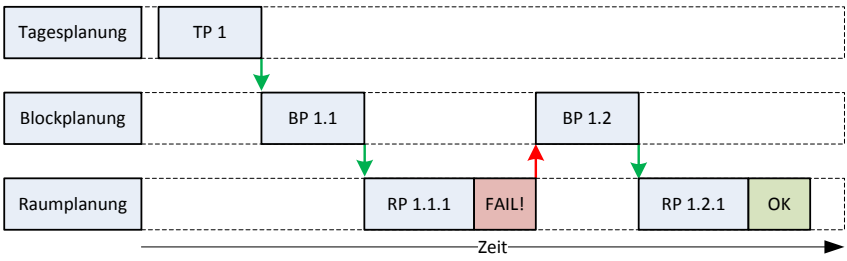

Abb. 1: Eine erfolgreiche Prüfungsplanung mit einem Backtracking-Schritt.

Werden alle Lösung für das Teilproblem des in der Hierarchie am höchsten stehenden Constraint-Speichers abgelehnt, ist das Gesamtproblem unter dieser Aufteilung nicht lösbar.

Haben wir eine Zuordnung von Prüfungen zu Tagen, können die Block- und Raumplanung für jeden Tag unabhängig voneinander erfolgen. Anstelle jeweils eines Speichers für die Block- und Raumplanung könnten wir für jeden Prüfungstag eine Block- und eine Raumplanung einsetzen. In Abbildung 2 ist dies schematisch dargestellt. Dies ermöglicht nach dem Errechnen einer Tageszuordnung die parallele Planung der Zeitblöcke und Räume für jeden Prüfungstag. Sobald jedoch eine einzige Blockplanung den aktuellen Vorschlag der Tagesplanung ablehnt, müssen alle Blockplanungen eventuell bereits errechnete Teillösungen verwerfen und mit dem neuen Tagesplan von vorne anfangen.

Dienstplanung: Bei dem Problem der Dienstplanung ergibt sich aus der Anwendung keine geeignete Aufteilung. Es lässt sich keine Aufteilung des Problems finden, bei der alle Constraints genau einem Teilproblem zugeordnet werden können. Dennoch können wir dieses Problem durch Aufteilen schneller lösen. Hierzu gibt es zwei Strategien: die Aufteilung von Constraints und das Hinzufügen neuer Constraints zur Wahrung der Konsistenz.

Wir wollen das Gesamtproblem lösen, indem wir anstelle eines großen Rahmendienstplanes über viele Wochen mehrere Teilrahmendienstpläne über wenige Wochen erstellen. So könnte ein Rahmendienstplan über 32 Wochen in zwei Zehnwochenpläne und einen Zwölfwochenplan aufgeteilt werden. Im Gegensatz zur Prüfungsplanung sind diese Teil-

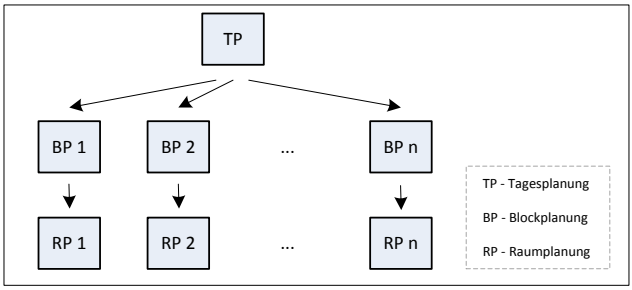

Abb. 2: Block- und Raumplanung können für jeden Prüfungstag unabhängig voneinder bestimmt werden.

probleme stark voneinander abhängig. Viele Constraints, wie zum Beispiel die Garantie der Schichtstärken, beziehen sich auf die Gesamtheit der zu planenden Wochen.

(1) Aufteilung von Constraints

Eine Möglichkeit, das Problem in Form von Teilproblemen zu lösen, ist die Aufteilung von Constraints. Dadurch gehen uns in der Regel einige Lösungen verloren.

Ein Beispiel: Nehmen wir an, wir wollen einen Rahmendienstplan für 32 Wochen erstellen und haben dieses Problem in die Planung von zwei Zehnwochenplänen und einem Zwölfwochenplan aufgeteilt. Wenn wir für Montag sieben Mitarbeiter in der Frühschicht brauchen, ist das als Constraint über den 32-Wochenplan leicht zu modellieren. In mindestens sieben der 32 Wochen muss für Montag 'Frühschicht' stehen. Hinzu kommen natürlich Constraints um sicherzustellen, dass Schichten als Block auftreten usw.

Wollen wir dieses Constraint auf die drei Teilprobleme aufteilen, muss in Summe über alle drei Teilpläne an mindestens sieben Wochen für Montag der Eintrag 'Frühschicht' stehen. Dies können wir zum Beispiel dadurch sicherstellen, dass in den beiden Zehn-Wochenplänen mindestens zwei mal für Montag 'Frühschicht' eingeplant ist und im Zwölf-Wochenplan drei mal.

Dies schließt allerdings einige gültige Verteilungen der Frühschicht am Montag aus und wir verlieren einige Lösungen. Solange wir jedoch einen gültigen Dienstplan erhalten, können wir dies verschmerzen. Diese Strategie eignet sich also besonders für Probleme mit sehr vielen Lösungen. Die Lösungen werden auch sehr schnell gefunden, da kein Backtracking über Teillösungen notwendig ist.

(2) Hinzufügen neuer Constraints

Eine andere Strategie besteht darin, lediglich die für ein Teilproblem lokalen Constraints von dem jeweiligen Constraint-Speicher verwalten zu lassen. Constraints, welche mehrere oder gar alle Teilprobleme betreffen, werden in einen separaten Constraint-Speicher

ausgelagert. Dies sind dann zum Beispiel Constraints, die Schichtstärken betreffen oder Constraints, welche einen nahtlosen Übergang zwischen den Wochenblöcken garantieren. Bei letzteren handelt es sich also um Constraints, welche ohne das Aufteilen in Teilprobleme gar nicht nötig wären.

Alle Constraint-Speicher werden dann wieder in eine Hierarchie eingeordnet, wobei der Speicher mit den Constraints über mehrere Teilprobleme ganz unten in der Hierarchie einzuordnen ist. Er hat also als letztes ein Vetorecht und kann für die Teillösungen überprüfen, ob alle Constraints, welche mehrere Teilprobleme betreffen, eingehalten sind.

Als Lösungsverfahren kommt hier wieder verteiltes Backtracking zum Einsatz. Ein möglicher Verlauf des Lösungsprozesses ist in Abbildung 3 zu sehen.

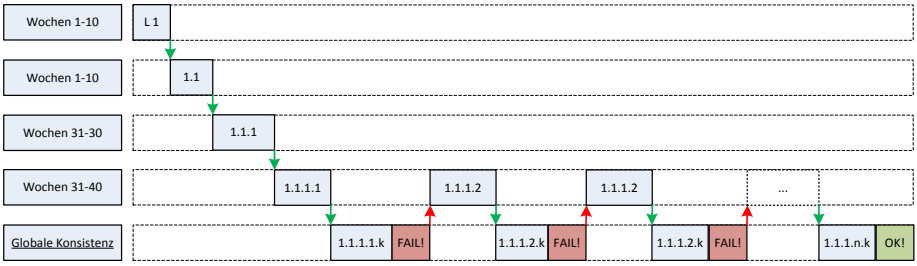

Abb. 3: Verteiltes Backtracking über drei Teilprobleme und einen zusätzlichen Constraint-Speicher, welcher mehrere Teilprobleme betreffende Constraints verwaltet.

Wie in der Abbildung zu erkennen ist, reichen die Speicher ihre Lösungen jeweils an den Speicher zur Bearbeitung des nachfolgenden Wochenblocks weiter. Als letztes erhält der Speicher, welcher mehrere Teilprobleme betreffende Constraints verwaltet, die Lösung und überprüft deren Konsistenz. Dies verursacht natürlich mehrere Backtrackingschritte. Wir verlieren jedoch keine Lösungen.

Durch das Auslagern der Constraints, welche mehrere Teilprobleme betreffen, und das Hinzufügen neuer Constraints, um die Konsistenz der Teillösungen zu garantieren, können wir verhindern, dass uns Lösungen verloren gehen. Der Lösungsprozess dauert aber erheblich länger, da durch die vielen Abhängigkeiten zwischen den Teilproblemen viele Backtrackingschritte notwendig sind. Dennoch kommen wir durch das Aufteilen häufig schneller zur Lösung als durch den Versuch, das Problem im Ganzen zu lösen, da ein Teil der Constraints innerhalb der einzelnen Teilprobleme bearbeitet wird.

4 Zusammenfassung

Anhand der Prüfungsplanung an einer Universität und der Rahmendienstplanung einer Feuerwehrleitstelle haben wir gezeigt, wie Probleme durch Aufteilen schneller gelöst werden können. Beide Probleme eignen sich unterschiedlich gut dazu, durch Aufteilen gelöst zu werden.

Die Planung der Prüfungen ist ein sehr dankbares Problem, da sich eine für den Nutzer intuitive Aufteilung aus der Anwendung ergibt. Die Menge der Constraints kann dabei so aufgeteilt werden, dass jedes Constraint genau einem Teilproblem zugeordnet werden kann. Zusätzlich können die Block- und Raumplanung für jeden Prüfungstag unabhängig voneinander geplant werden. Dies ermöglicht einen hochgradig parallelen Lösungsprozess.

Im Gegensatz dazu ergibt sich bei der Dienstplanung keine intuitive Aufteilung in Teilprobleme. Die Menge der Constraints lässt sich nicht so aufteilen, dass jedes Constraint genau einem Teilproblem zugeordnet ist. Dennoch können wir dieses Problem durch Aufteilen schneller lösen, indem wir bei ausreichend vielen Lösungen auf einige Lösungen verzichten und die Constraints für die Teilprobleme verfeinert lokalisieren. Des Weiteren können Constraints, welche mehrere Teilprobleme betreffen, auch in einen separaten Constraint-Speicher ausgelagert werden und das Problem kann mit Hilfe eines verteilten Backtrackings gelöst werden.

Literaturverzeichnis

[Be05] Beermann, Beate: Leitfaden zur Einführung und Gestaltung von Nacht- und Schichtarbeit. http://www.baua.de/de/Publikationen/Broschueren/A23.html. Bundesanstalt für Arbeitsschutz und Arbeitsmedizin (BAuA), 2005.

[Fa06] Faltings, Boi: Distributed Constraint Programming. S. 699–729. In [RvBW06], S. 699–729, 2006.

[Hä13] Häfner, Franz: Constraint-basierte Berechnung ausgeglichener und korrekter Rahmendienstpläne in akzeptabler Berechnungszeit. Masterarbeit, BTU Cottbus-Senftenberg, 2013.

[RvBW06] Rossi, Francesca; van Beek, Peter; Walsh, Toby: Handbook of Constraint Programming. Elsevier, Amsterdam, The Nethderlands, 2006.

[Sa12] Sauer, Peter: Verteilte Nebenläufige Constraint-Programmierung. Masterarbeit, BTU Cottbus, 2012.

[YH00] Yokoo, Makoto; Hirayama, Katsutoshi: Algorithms for Distributed Constraint Satisfaction. Autonomous Agents and Multi-Agent Systems, 3(2):198–212, 2000.