Ressourcenplanung unter Nutzung der Java-Constraint-Bibliothek firstCS*

Saskia Sandow
Fraunhofer Institut für Rechnerarchitektur und Softwaretechnik, FIRST

Abstract: Einen Ansatz zur Lösung kombinatorischer Probleme bildet die Constraint-logische Programmierung. Die Ressourcenplanung gehört zu solchen Problemen und findet in vielen Bereichen Anwendung. Es wird ein Ressourcenplaner vorgestellt, der unter Nutzung der Java-Constraint-Bibliothek firstCS [Wol06] für ein Ressourcenplanungsproblem, das auf einfache Weise in XML spezifiziert werden kann, einen Plan findet. Über eine Benutzeroberfläche ist ein solcher Plan interaktiv veränderbar. Darüber hinaus ist es möglich, unter Verwendung einer zusätzlichen Konfigurationsdatei, die an das Problem angepasste Strategien enthält, das Planungswerkzeug zu steuern und dessen Performanz zu verbessern.

1 Einleitung

Das Lösen kombinatorischer Probleme ist eine häufig genutzte Anwendung der Constraint-logischen Programmierung (CLP)[1]. Hierzu gehören auch Ressourcenplanungsprobleme, die es in vielen Bereichen wie Industrie, Wirtschaft oder Medizin gibt. In CLP wird ein Problem in Form von Variablen und *Constraints*, die auf diesen Variablen definiert sind, dargestellt. Für jede Variable gibt es einen Wertebereich (Domäne), der die Werte enthält, die diese Variable annehmen kann. Ein *Constraint* ist auf einer, zwei oder mehr Variablen definiert und beschränkt die Domänen dieser Variablen auf solche Werte, die in einer Lösung vorkommen können. Eine Lösung des Problems ist eine Wertebelegung aller Variablen, so dass alle *Constraints* erfüllt sind. Für die Suche einer Lösung werden Propagation, *Labeling* und *Backtracking* kombiniert, d.h. zunächst wird durch Propagation der Suchraum eingeschränkt, indem Werte aus den Variablendomänen entfernt werden, die in jeder Lösung des Problems ausgeschlossen werden können. Ist die Wertebelegung hierdurch noch nicht vollständig, erfolgt das *Labeling*, also das Belegen einer ausgewählten Variable mit einem ausgewählten Wert ihrer Domäne, und es wird erneut propagiert. Dies wird wiederholt, bis eine Lösung gefunden ist oder ein Konflikt auftritt. Im letzteren Fall kommt es zum *Backtracking*, d.h. es wird zur letzten ausgewählten Variable zurückgekehrt und dieser ein anderer Wert ihrer Domäne zugewiesen. Dieser Vorgang kann solange wiederholt werden, bis der gesamte Suchraum durchsucht worden ist. In dem Fall wurden alle Lösungen oder keine Lösung gefunden.

*Siehe [Wol06].
[1] Als einführende Literatur zu CLP sei z.B. [HW07] genannt.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html

Da der Suchraum eines kombinatorischen Problems sehr groß werden kann, gibt es verschiedene Ansätze, um die Suche zu verbessern, z.B. durch *Conflict-directed Backjumping*[2] in [Pro93] oder durch unterschiedliche Heuristiken zur Auswahl von Variablen und ihrer Domänenwerte. Das Spektrum kombinatorischer Probleme ist allerdings ebenso sehr groß, so dass solche Ansätze zur Verbesserung der Suche im Allgemeinen nicht für alle Problemklassen gleich gut sind. Daher gibt es häufig Anwendungen, die nur bestimmte Problemklassen ansprechen, um problemspezifische Informationen und Strategien nutzen zu können.
Der hier vorgestellte Ressourcenplaner verfolgt das Ziel, einen möglichst breiten Einsatzbereich zu erreichen, ist also nicht auf einen bestimmten Anwendungsbereich spezialisiert. Daher ist es nötig, dem Planer innerhalb der Problemspezifikation alle nötigen Informationen zu dem zu lösenden Problem bereitzustellen. Weil das Problem dem Planer also im Voraus nicht bekannt ist, ist im Planer selbst nur eine einfache Suchstrategie integriert, die in vielen Fällen zu keiner guten Performanz führt. Eine dem Problem angepasste Suchstrategie kann allerdings in Form einer zusätzlichen Konfigurationsdatei bereitgestellt werden.
In Abschnitt 2 wird der Ressourcenplaner vorgestellt und an einem Beispiel veranschaulicht. Abschnitt 3 enthält experimentelle Ergebnisse und in Abschnitt 4 wird abschließend die bisherige und weitere Entwicklung des Ressourcenplaners zusammengefasst.

2 Ressourcenplaner

In diesem Abschnitt wird der Ressourcenplaner im Einzelnen erläutert. Dazu wird auf die Spezifikation des Problems, die eigentliche Planung und auf die vorhandene Benutzeroberfläche mit der Möglichkeit zur interaktiven Nutzung eingegangen. Gewisse Einzelheiten werden anhand eines Beispiels erläutert. Dieses Beispiel kommt aus der Medizin und hat als Ziel eine terminbasierte Planung von Dialysebehandlungen. Eine solche Behandlung besteht aus bestimmten Teilaufgaben (*Tasks*), die in vorgegebener Reihenfolge stattfinden und gewisse Zeiten und Ressourcen erfordern. Konkret bedeutet dies, ein Patient befindet sich zu seinem Termin im Dialysezentrum, bereitet sich in einer vorgegebenen Zeit auf die Dialyse vor (Umziehen und eventuelle Messungen), wird anschließend von einer verfügbaren Pflegekraft an ein Dialysegerät angeschlossen und verbleibt dort für die Dauer der Dialyse. Danach trennt eine Pflegekraft den Patienten vom Gerät und es folgt die Nachbereitung des Patienten (Umziehen, Datenerfassung oder ähnliches) sowie die Reinigung des verwendeten Gerätes durch eine Pflege- bzw. Reinigungskraft.

2.1 Problemspezifikation

Um eine Lösung für ein Ressourcenplanungsproblem zu finden, muss dieses in einer formalen Beschreibung vorliegen, die der Planer verarbeiten kann. Hierzu wird die *Extensi-*

[2] Conflict-directed Backjumping beinhaltet das Zurückspringen zur eigentlichen Ursache eines Konflikts mit zusätzlichem Lernen des Konflikts, so dass dieser bei der weiteren Suche vermieden wird.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html

ble Markup Language (XML) [BPSM+08] verwendet. Der Inhalt und Aufbau einer XML-Spezifikation ist durch ein XML Schema vorgegeben. Mit Hilfe dieses Schemas wird beim Parsen die Korrektheit der Spezifikation überprüft. Sie enthält folgende Elemente:

`makeSpan` (erforderlich)	beschreibt den Planungshorizont, d.h. die Zeitspanne, innerhalb der geplant wird
`earliestBegin`	frühester Beginn von `makeSpan`
`latestEnd`	spätestes Ende von `makeSpan`
`unit`	Einheit bzw. Taktung von `makeSpan`
`tasks` (erforderlich)	enthält alle Aktivitäten (*Tasks*), die zu planen sind
`task`	beschreibt eine einzelne *Task*
`ID`	eindeutiger Bezeichner der *Task*
`name` (opt)	Name (Beschreibung) der *Task*
`groupID` (opt)	dient der Gruppierung verschiedener *Tasks*
`begin` (opt)	feste Startzeit der *Task*
`earliestBegin` (opt)	früheste Startzeit der *Task*
`preferredBegin` (opt)	gewünschte Startzeit der *Task*
`end` (opt)	feste Endzeit der *Task*
`latestEnd` (opt)	späteste Endzeit der *Task*
`dummy` (opt)	Status als Platzhalter
`resources`	enthält die potentiellen Ressourcen, auf denen die *Task* laufen kann; zu jeder Ressource gehört ein Bezeichner (`resourceRef`) und es kann die jeweilige Dauer der *Task* (`duration`) festgelegt werden
`relations` (erforderlich)	enthält die *Constraints* des Planungsproblems
`distance`	*Constraint*, das die Reihenfolge zweier *Tasks* bestimmt (gleichzeitig, nacheinander, zeitversetzt)[3]
`allEqual`	*Constraint*, das festlegt, dass zwei oder mehrere *Tasks* zu gleichen Zeiten starten oder enden, dass sie die gleiche Dauer haben oder dass sie auf gleichen Ressourcen laufen
`allDifferent`	*Constraint*, das festlegt, dass zwei oder mehrere *Tasks* zu unterschiedlichen Zeiten starten oder enden, dass sie unterschiedliche Dauer haben oder dass sie auf unterschiedlichen Ressourcen laufen
`resourceGroups` (optional)	beschreibt gewünschte Gruppierungen von Ressourcen für den Fall, dass solche Ressourcen nicht in der gleichen Ressourcenklasse vorkommen (dient hauptsächlich einer übersichtlicheren Anordnung der Ressourcen bzw. Ressourcenklassen in der GUI)

Mit Hilfe dieser Elemente ist es möglich, auf einfache Weise (auch ohne tiefes Hintergrundwissen zu CLP) ein gegebenes Problem zu modellieren.

[3] Ein `distance`-*Constraint* hat die Form $A + x\ comp\ B$, wobei A und B Start- oder Endzeiten verschiedener *Tasks* sind, x der Abstand und $comp \in \{<, \leq, =, >, \geq\}$

ISBN 978-3-88579-286-4

2.2 Planung

Nach dem Einlesen und Parsen der Problemspezifikation in XML Format erfolgt die Erstellung des *Constraint*-Systems, indem die nötigen Variablen und folgende *Constraints* mit Hilfe der Java-Constraint-Bibliothek firstCS ([Wol06]) erzeugt werden:

- Es werden die Ressourcenklassen ermittelt und ein *AlternativeResource-Constraint* für jede Ressourcenklasse erstellt, das die zu dieser Ressourcenklasse gehörigen *Tasks* auf die zur Verfügung stehenden Ressourcen verteilt.
- Ein *Element-Constraint* wird für die *Tasks* hinzugefügt, deren Dauer von der Ressource abhängt.
- Die in `relations` angegebenen *Constraints* werden entsprechend übersetzt. Hierzu werden *AllEqual-* und *AllDifferent-Constraints* sowie ein *Before-Constraint* für jede `distance relation` genutzt.
- Zur Einhaltung der Wunschzeiten (für den Fall, dass welche angegeben wurden) dienen ein *IsInDomain-Constraint* für jede Wunschzeit und ein *WeightedSum-Constraint*. Mit dem *WeightedSum-Constraint* wird die Anzahl der erlaubten Verletzungen gesteuert, d.h. die Zahl der eingehaltenen Wunschzeiten muss der Anzahl der *Tasks* mit Wunschzeiten abzüglich der erlaubten Verletzungen entsprechen. Für die Einhaltung der Wunschzeiten ist das *IsInDomain-Constraint* zuständig, indem es dafür sorgt, dass der Wert der Startvariablen einer *Task* mit angegebener Wunschzeit innerhalb des Wertebereichs [Wunschzeit, Wunschzeit + Toleranz] liegt.

Anschließend wird propagiert und ein *Choice Point* (*Level* 0) gesetzt. Ist die Variablenbelegung an dieser Stelle noch nicht vollständig, so wird eine Variable ausgewählt, mit einem Wert ihres Wertebereichs belegt und wieder propagiert. Führt dies zu einem Konflikt, findet *Backtracking* statt und es wird ein anderer Wert aus dem Wertebereich gewählt. Führt dies zu keinem Konflikt, wird die nächste Variable gewählt. Dieser Vorgang wiederholt sich, bis die Variablenbelegung vollständig und somit ein Plan vorhanden ist oder bis der gesamte Suchraum durchlaufen wurde, d.h. es existiert keine Lösung.
Die Suche nach einem Plan erfolgt im Standardfall durch einfaches *Backracking* und die Auswahl einer Variablen erfolgt in einer mehr oder weniger beliebigen Reihenfolge (abhängig von der Speicherart im System). Da verschiedene Problemklassen verschiedene Suchstrategien erfordern, um eine gute Performanz zu erreichen, und das Planungswerkzeug nicht auf bestimmte Probleme beschränkt sein soll, wurde dieser Standardfall gewählt. Dem erfahrenen und mit firstCS vertrautem Benutzer ist es allerdings möglich, in einer Konfigurationsdatei (ebenso als XML) eine angepasste Suchstrategie vorzugeben. Hierzu stellt firstCS bereits verschiedene *Labeler* bereit – vom einfachen *Backtracking* bis hin zu *Labeler*, die speziell für das *Scheduling* verwendet werden. Darüber hinaus bietet firstCS die Möglichkeit und den Rahmen, um zusätzliche *Labeler* erweitert zu werden. Mit Hilfe des in firstCS vorhandenen `MetaLabel` können verschiedene *Labeling*-Strategien kombiniert werden.
In Abbildung 1 ist eine solche Konfigurationsdatei für das Beispiel `dialysis7on4` dargestellt. Hier wird eine Kombination von einfachem *Backtracking* (`BtLabel`) und einem

ISBN 978-3-88579-286-4

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <resourceSchedulingRequestSearchStrategy>
3      <search ref="MetaLabel">
4          <search ref="MetaLabel">
5              <forEachGroup groupOrder="id">
6                  <search ref="SingleTaskOnAlternativesScheduler">
7                      <task taskID="2.*"/>
8                      <int val="4"/>
9                  </search>
10                 <search ref="BtLabel">
11                     <task taskID="3.*" var="resource"/>
12                     <task taskID="4.*" var="resource"/>
13                     <task taskID="5.*_b" var="resource"/>
14                 </search>
15             </forEachGroup>
16         </search>
17         <search ref="BtLabel">
18             <forEachGroup>
19                 <task taskID="1.*" var="start"/>
20             </forEachGroup>
21         </search>
22         <search ref="BtLabel">
23             <forEachGroup>
24                 <task taskID="5.*_a" var="start"/>
25             </forEachGroup>
26         </search>
27     </search>
28 </resourceSchedulingRequestSearchStrategy>
```

Abbildung 1: Beispiel einer XML-Suchstrategie für `dialysis7on4`

auf *Scheduling* spezialisierten *Labeler* (`SingleTaskOnAlternativesScheduler`) angewandt. Für die Kombination verschiedener *Labeler* dient das sogenannte `MetaLabel` (siehe Zeile 3), das als eine Liste von *Labeler* zu betrachten ist.
Im `dialysis7on4`-Beispiel haben wir 7 Patienten, d.h. 7 *Task Groups*. Der Bezeichner einer *Task*, z.B. `2.1`, gibt Rückschlüsse auf die Art der *Task* (erste Zahl) und auf die Gruppe der *Task* (zweite Zahl). In diesem Beispiel bildet das äußerste `MetaLabel` (Zeile 3-27) eine Liste in Form von `[MetaLabel,BtLabel,BtLabel]`. Diese Liste wird von vorne nach hinten abgearbeitet. Das innere `MetaLabel` (Zeile 4-16) ist eine Liste der Form `[SingleTaskOnAlternativesScheduler,BtLabel]`, die für jede Gruppe (Zeile 5 : `forEachGroup`) durchgegangen wird. Dies bedeutet, wir haben die Gruppen in lexikographischer Ordnung (`groupOrder`) vorliegen und für jede Gruppe wird zunächst der `SingleTaskOnAlternativesScheduler` (Zeile 6) und dann das `BtLabel` (Zeile 10) angewandt.
Der `SingleTaskOnAlternativesScheduler` benötigt als Parameter eine *Task*, die Wunschzeit und einen Wert, der die Einheit für die Relaxation bestimmt. Die Startzeit der *Task* ist somit die ausgewählte Variable und die Wunschzeit der erste Wert aus deren Wertebereich, mit dem die Variable belegt wird. Wird mit diesem Wert keine Lösung gefunden, wird mit Hilfe des Relaxationswertes ein nächster Wert für die Variable gesucht, d.h. diese Suche erfolgt zunächst in dem Bereich [Wunschzeit, Wunschzeit + Relaxati-

on] und dann [Wunschzeit - Relaxation, Wunschzeit]. Gibt es wieder keine Lösung wird der Relaxationswert schrittweise vervielfacht, bis der gesamte Wertebereich der Variablen betrachtet worden ist. In unserem Beispiel wird die *Task* mit dem Bezeichner `2.*`, die die eigentliche Dialyse beinhaltet, ausgewählt und der Relaxationswert auf 4 gesetzt, was dem Vierfachen der *Makespan Unit* entspricht. Dieser *Labeler* wird für die entsprechende *Task* aus jeder Gruppe durchgeführt, wobei nach jeder Durchführung zunächst noch das `BtLabel` in der aktuellen Gruppe angewandt wird. Das `BtLabel` (Zeile 10) in unserem Beispiel besagt, dass aus der aktuellen Gruppe die drei *Tasks* mit den Bezeichnern `3.*`, `4.*` und `5.*_b` (entspricht Anschließen, Trennen und Reinigen durch eine Pflegekraft) gewählt und deren Ressourcenvariablen in der angebebenen Reihenfolge belegt werden.
Wurde dieses `MetaLabel` (Zeile 4-16) für jede Gruppe ausgeführt, erfolgt ein `BtLabel` (Zeile 17) für die Startvariablen aller *Tasks* mit dem Bezeichner `1.*` (entspricht der gesamten Behandlung) und ein `BtLabel` (Zeile 22) für die Startvariablen aller *Tasks* mit dem Bezeichner `5.*_a` (entspricht der Reinigung auf dem Gerät).
Wichtig ist bei der Formulierung der Suchstrategie, dass das *Labeling* aller Variablen sichergestellt ist. Dazu ist es nicht zwingend erforderlich, ein *Labeling* auf jede Variable direkt anzuwenden. Sind die Werte von Variablen durch *Constraints* eindeutig an die Werte solcher Variablen gebunden, auf die ein *Labeler* angewendet wird, so ist das *Labeling* dieser Variablen indirekt gegeben. Sind noch Variablen vorhanden, denen kein eindeutiger Wert zugeordnet wurde, gibt es eine Warnung, dass das *Labeling* unvollständig ist, zusammen mit der Information, welche Variablen nicht belegt worden sind.
Mit Hilfe der *Constraints* zur Einhaltung der Wunschzeiten (falls vorhanden) und einer Variablen, die die Anzahl der Verletzungen anzeigt, wird ein optimaler Plan in folgendem Sinne gefunden. Es wird zunächst ein Plan gesucht, der alle Wunschzeiten berücksichtigt, also KEINE Verletzungen enthält (optimaler Plan). Wird ein solcher Plan nicht in einer festen Zeitspanne (*Timeout*) gefunden, so erfolgt, ebenfalls für die Dauer des *Timeouts*, die Suche nach einem Plan, der EINE Verletzung der Wunschzeiten enthält. Wird wieder kein Plan gefunden, so wird die Anzahl der erlaubten Verletzungen auf zwei erhöht usw.. Wird eine Lösung gefunden, wird diese ausgegeben oder, falls die Benutzeroberfläche genutzt wird, grafisch angezeigt. Wird keine Lösung gefunden, erhält der Benutzer ebenfalls eine Meldung.

2.3 Visualisierung

Die grafische Benutzeroberfläche (*Graphical User Interface* GUI) dient der Veranschaulichung des Planes und der leichteren Bedienbarkeit. Außerdem ermöglicht sie dem Benutzer zusätzlich, interaktiv Änderungen am Plan vorzunehmen (siehe Abschnitt 2.4). Für den Plan gibt es verschiedene Ansichten:

1. Die Ressourcenansicht (siehe Abbildung 2), in der die Zeilen den Ressourcen entsprechen und die Balken im Graphen den *Tasks*. Beim Passieren oder Anklicken eines solchen Balken erscheint eine Kurzinformation, die die ID, den Namen und die Gruppe der *Task* enthält sowie deren Start- und Endzeit. Die Anordnung der Ressourcen ist durch die vorhandenen Ressourcenklassen (sind optisch voneinander

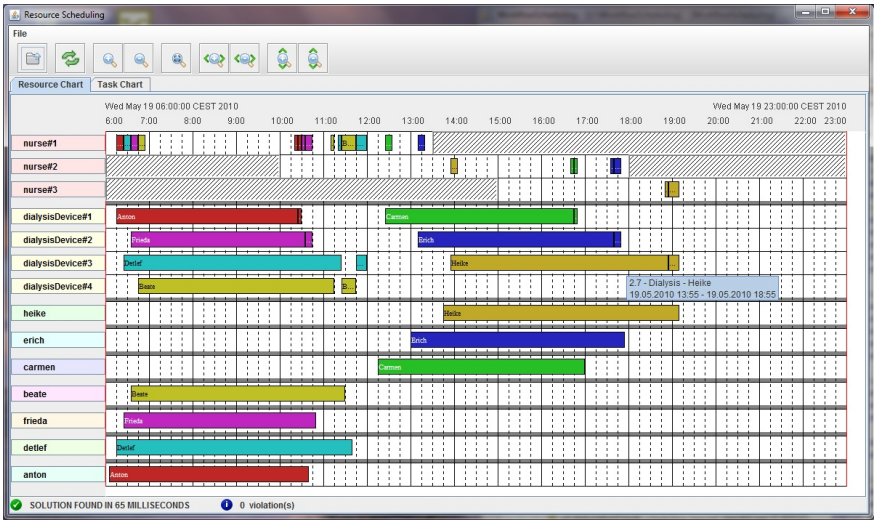

Abbildung 2: Ressourcenansicht des Beispiels `dialysis7on4`

abgesetzt) und die optional vom Benutzer angebenen Ressourcengruppen bestimmt. Hat der Benutzer für einen Teil oder alle *Tasks* auch eine Gruppenzugehörigkeit angegeben, so erhalten die *Tasks* einer Gruppe die gleiche Farbe (dient der leichteren Orientierung innerhalb des Graphen). Im Beispiel der Dialyseplanung können die *Tasks* einer Behandlung, die zu einem Patienten gehören, zu einer Gruppe zugewiesen werden, die dann in der Ansicht die gleiche Farbe erhalten und somit sofort einem bestimmten Patienten zugeordnet werden können.

2. Die Aktivitätenansicht, in der die Zeilen den *Tasks* entsprechen und die Balken im Graphen den Zeitabschnitt markieren, in dem die entsprechende Ressource belegt ist. Es gibt zwei verschiedene Aktivitätenansichten, die sich durch die Anordnung der *Tasks* unterscheiden:

   (a) Die *Tasks* sind geordnet nach der *Task*-ID.

   (b) Die *Tasks* sind geordnet nach der Gruppen-ID der *Tasks*.

2.4 Interaktion

Über die Benutzeroberfläche wird dem Benutzer ermöglicht, interaktiv Umplanungen vorzunehmen. Diese beinhalten zum einen das Verschieben einer *Task* per Maus auf eine gewünschte Startzeit und zum anderen das Deaktivieren einer Ressource durch Klicken auf den entsprechenden *Button*. Im letzteren Fall wird die deaktivierte Ressource aus den Domänen der Ressourcenvariablen der *Tasks* entfernt und somit nicht mehr verplant. Die Umplanung erfolgt, indem die Startzeiten aller anderen *Tasks* möglichst beibehalten werden, so dass so wenig Änderungen am Plan vorgenommen werden wie möglich. Dabei

bleibt die Suchstrategie erhalten (entweder die Standardsuche oder die in der Konfigurationsdatei vorgegebene Suche). Der vorherige Plan wird gespeichert und für den Fall, dass keine neue Lösung für die interaktive Änderung gefunden wird, wiederhergestellt zusammen mit der Meldung, dass keine Lösung gefunden wurde.

3 Benchmarks

In diesem Abschnitt wird der Ressourcenplaner an folgenden Beispielen getestet. Zu jedem Beispiel gibt es auch eine zusätzliche XML-Datei, die die Suchstrategie vorgibt.

- `dialysis7on4.xml`
  In diesem Beispiel gibt es 7 Patienten, 4 Dialysegeräte und 3 Krankenschwestern, wobei die Krankenschwestern in Schichten eingeteilt sind.
- `dialysis7on4-clean.xml`
  Dieses Beispiel ist wie das erste mit der Ausnahme, dass es hier eine zusätzliche Reinigungskraft gibt, die ausschließlich für das Reinigen der Dialysegeräte zuständig ist.
- `dialysis20on10.xml`
  In diesem Beispiel gibt es 20 Patienten, 10 Dialysegeräte und 4 Krankenschwestern.
- `dialysis20on10-clean.xml`
  Dieses Beispiel entspricht dem `dialysis20on10` mit zusätzlicher Reinigungskraft.
- `surgery.xml`
  In diesem Beispiel geht es um OP-Planung, d.h. es gibt 3 Krankenschwestern, 3 OP-Säle, 3 Chirurgen, 2 Vorbereitungs- und 2 Aufwachräume. Der Ablauf für einen Patienten sieht folgendermaßen aus : Zunächst erfolgt die Vorbereitung des Patienten, wofür ein Raum und eine Krankenschwester benötigt werden, dann kommt es zur OP, wofür ein OP-Saal und ein Chirurg erforderlich sind. Anschließend kommt der Patient in einen Aufwachraum, während sich eine Krankenschwester um die Reinigung des benutzten OP-Saals kümmert.

Getestet wurde auf einem Intel(R) Xeon(R) E5540 (2,53 GHz) Rechner mit 6 GB Arbeitsspeicher. Die Tests wurden jeweils mehrfach durchlaufen und der Durchschnitt ermittelt, wobei zu beobachten ist, dass die Zeiten nur um wenige Millisekunden variieren, da die Suche für ein Problem immer den gleichen festen Ablauf hat. Für jedes der fünf Beispiele enthält Tabelle 1 Laufzeiten unter verschiedenen Kriterien. Die Laufzeiten sind in Millisekunden zu verstehen und die Zahl in Klammern gibt die Anzahl der Verletzungen der Wunschzeiten bzw. bei Interaktion der vorherigen Zeiten an.
Die Suche erfolgt entweder durch einfaches *Backtracking* (Standard) oder durch die in der Konfigurationsdatei angegebene Strategie. Hierbei ist anzumerken, dass die angegebene

	Standardsuche			benutzergesteuerte Suche		
	Timeout			Timeout		
	1000	2000	5000	1000	2000	5000
dialysis7on4	9.3(0)	7.8(0)	5.0(0)	9.2(0)	11.1(0)	11.1(0)
1. 2.7 auf 16:00	12.6(0)	4.6(0)	6.3(0)	7.9(0)	8.0(0)	7.9(0)
2. 2.6 auf 8:00	-(-)	-(-)	-(-)	288.4(1)	287.0(1)	283.9(1)
3. nurse1 gesperrt	-(-)	-(-)	-(-)	2647.5(4)	4632.8(4)	10620.6(4)
dialysis7on4-clean	9.3(0)	6.4(0)	6.3(0)	6.3(0)	7.9(0)	7.8(0)
1. 2.7 auf 16:00	9.5(0)	7.8(0)	9.1(0)	6.2(0)	1.6(0)	4.7(0)
2. 2.6 auf 8:00	-(-)	-(-)	-(-)	81.0(1)	73.6(1)	78.0(1)
3. nurse1 gesperrt	-(-)	-(-)	-(-)	2357.1(4)	2616.6(4)	10344.5(4)
dialysis20on10	2118.2(2)	4111.8(2)	10114.6(2)	3112.7(3)	6119.2(3)	15095.1(3)
1. 2.7 auf 16:00	76.4(0)	74.9(0)	75.0(0)	56.3(0)	62.5(0)	59.4(0)
2. 2.7 auf 8:00	-(-)	-(-)	-(-)	1157.5(1)	2154.3(1)	5154.4(1)
3. nurse4 gesperrt	-(-)	-(-)	-(-)	-(-)	-(-)	-(-)
dialysis20on10-clean	1399.9(2)	2454.4(2)	5363.4(2)	2186.6(3)	4166.1(3)	10158.2(3)
1. 2.7 auf 16:00	38.8(0)	35.9(0)	38.9(0)	39.0(0)	37.5(0)	36.0(0)
2. 2.7 auf 8:00	-(-)	-(-)	-(-)	115.7(1)	115.7(1)	112.3(1)
3. nurse4 gesperrt	-(-)	-(-)	-(-)	1157.4(2)	2162.2(2)	5165.1(2)
surgery	-(-)	-(-)	-(-)	6303.2(7)	12351.4(7)	30299.6(7)
1. 2.4_a auf 16:00				15.6(0)	12.2(0)	15.8(0)
2. 2.6_a auf 8:00				830.4(1)	802.2(1)	812.6(1)
3. nurse1 gesperrt				7559.8(9)	14489.6(9)	35537.2(9)

Tabelle 1: Laufzeiten unter verschiedenen Kriterien

Strategie genauso gut auch fest in der Implementierung untergebracht werden könnte. Eine Auslagerung der Suchstrategie in eine leichter anzupassende Konfigurationsdatei führt dabei nur zu einem unbedeutenden Laufzeitverlust. Betrachtet werden verschiedene *Timeouts*, also die Zeiten, die dem Solver zur Verfügung stehen, um eine Lösung mit keiner Verletzung, einer Verletzung, zwei Verletzungen usw. zu finden. Die erste Zeile jeden Beispiels enthält die Laufzeiten für die initiale Lösung, d.h. unter Berücksichtigung der in der XML-Spezifikation angegebenen Wunschzeiten. Die drei Zeilen darunter zeigen die Laufzeiten von Beispielen interaktiver Nutzung, von der initialen Lösung ausgehend. Unter 1. wird die *Task* mit dem angegebenen Bezeichner auf die angegebene Zeit verschoben, was offensichtlich (also bereits in der grafischen Ansicht klar erkennbar) KEINE Verletzung hervorruft. Unter 2. wird ebenfalls eine *Task* verschoben, wobei hier offensichtlich EINE Verletzung entsteht. Und unter 3. wird die angegebene Ressource deaktiviert, was offensichtlich MEHR ALS EINE Verletzung nach sich zieht.
Das Beispiel `dialysis7on4` gibt es jeweils ohne und mit Reinigungskraft. Hier wird unter 1. die *Task* `2.7-Dialysis-Heike` auf 16 Uhr verschoben, unter 2. die *Task* `2.6-Dialysis-Erich` auf 8 Uhr und unter 3. wird die Krankenschwester `nurse1` gesperrt.
Das Beispiel `dialysis20on10` gibt es ebenfalls ohne und mit Reinigungskraft. Hier wird unter 1. die *Task* `2.7-Dialysis-Carmen` auf 16 Uhr verschoben, unter 2. ebenfalls die *Task* `2.7-Dialysis-Carmen` auf 8 Uhr und unter 3. wird die Krankenschwester `nurse4` gesperrt.
Im Beispiel `surgery` wird unter 1. die *Task* `2.4_a-SurgeryOR-Patient#4` auf 16 Uhr verschoben, unter 2. die *Task* `2.6_a-SurgeryOR-Patient#6` auf 8 Uhr und unter 3. wird die Krankenschwester `nurse4` gesperrt.
Beim Blick auf die Tabelle ist sofort erkennbar, dass die Standardsuche, also einfaches *Backtracking*, wie zu erwarten, schnell an ihre Grenzen stößt, d.h. bei interaktiver Nut-

ISBN 978-3-88579-286-4

zung, bei der Verletzungen entstehen (auch wenn es nur eine ist), wird keine Lösung in der verfügbaren Zeit gefunden. Bei der initialen Lösung kann es allerdings vorkommen (z.B. `dialysis20on10`), dass durch die Standardsuche eine Lösung mit weniger Konflikten gefunden wird als durch die benutzergesteuerte Suche. Dadurch verkürzt sich auch die Laufzeit. Gibt es Lösungen ganz ohne Verletzungen, sind die Laufzeiten bei der Standardsuche vergleichbar mit denen der benutzergesteuerten Suche. Diese Tatsache und die schlechten Ergebnisse bei interaktiver Nutzung sind darauf zurückzuführen, dass die Variablenauswahl davon abhängt, in welcher Reihenfolge die *Tasks* im System gespeichert sind, und somit entscheidend ist, wo im Suchraum eine Lösung vorhanden ist und ob diese zufällig in der begrenzten Zeit gefunden wird.
Bei der benutzergesteuerten Suche ist zu sehen, dass es hier nur an einer Stelle Probleme gibt: Im Beispiel `dialysis20on10` führt das Ausschalten der Krankenschwester `nurse4` zu keiner Lösung, was möglicherweise daran liegt, dass es keine Lösung gibt. Die Verlängerung des *Timeouts* lässt keinen Effekt bezüglich der Qualität der Lösungen erkennen. Es werden Lösungen mit der gleichen Anzahl von Verletzungen gefunden (unabhängig von der Länge des *Timeouts*), wodurch sich die Laufzeiten nur verlängern. Die Frage ist also, ist die Länge des *Timeouts* nicht lang genug, um eine Lösung mit weniger Verletzungen zu finden, oder gibt es keine Lösung mit weniger Verletzungen. Im Beispiel `dialysis20on10` wissen wir allerdings, dass es auch eine Lösung mit 2 Verletzungen gibt, die die benutzergesteuerte Suche nicht gefunden hat.
Verbesserungen an der Standardsuche könnte ein zufälliges Auswählen einer zu belegenden Variablen sein, um bei der Suche in verschiedene Bereiche des Suchraums zu gelangen. Auch bei der benutzergesteuerten Suche könnte man Veränderungen vornehmen, um eventuell eine bessere Performanz zu erreichen. Dies könnte beispielsweise erreicht werden, indem man dem Benutzer noch mehr Ansätze zum Konfigurieren gibt. Solche Angaben könnten ein vom Benutzer festgelegtes *Timeout* sein oder auch, in welcher Reihenfolge die Werte aus dem Wertebereich einer Variablen bei einfachen Labelern wie `BtLabel` als Belegung gewählt werden. Vorteilhaft für den Nutzer wäre es auch die Toleranzspanne einer Verletzung festzulegen oder auch die eigentliche Zielfunktion vorzugeben, die bisher auf Einhaltung von Wunschzeiten beschränkt ist. Möglichkeiten hierfür wären z.B. das effektive Ausnutzen von Ressourcen, d.h. so wenig Leerlaufzeiten wie möglich, oder auch eine Kostenfunktion, die berücksichtigt, dass eine Ressource zu bestimmten Zeiten günstiger ist.

4 Ausblick

Es wurde ein Planungswerkzeug vorgestellt, das für Ressourcenplanungsprobleme, die in Form von XML-Spezifikationen vorliegen, einen Plan erzeugt und das über eine Benutzeroberfläche Interaktion durch den Benutzer ermöglicht. Des Weiteren lässt es sich über eine Konfigurationsdatei (in XML) steuern. Die Sprachelemente für die Spezifikation eines Ressourcenproblems sind gezielt einfach gehalten, um die Verwendung des Planers auch Nutzern zu ermöglichen, die kein oder kaum theoretisches Hintergrundwissen zu

CLP besitzen. Aktuell[4] werden Ressourcenprobleme auf alternativen Ressourcen gelöst, künftig sollen aber auch z.B. Ressourcenprobleme auf kumulativen Ressourcen hinzukommen. Hierzu muss die Spezifikationssprache zur Problemmodellierung entsprechend erweitert werden. Die Interaktionsmöglichkeiten über die Benutzeroberfläche beschränken sich derzeit auf das Verschieben von *Tasks* und das Deaktivieren von einzelnen Ressourcen. Hier gibt es ebenfalls Erweiterungsmöglichkeiten wie das Hinzufügen und Entfernen von *Tasks* oder *Task*-Gruppen oder das Verschieben einer *Task* auf eine bestimmte Ressource (innerhalb einer Ressourcenklasse). Außerdem ist es wünschenswert, wenn nicht nur die letzte Änderung des Benutzers berücksichtigt wird, sondern auch vorhergehende Interaktionen. Dies könnte z.B. durch Festlegung von Prioritäten auf den Interaktionen realisiert werden. Die ausgelagerte Konfiguration über eine Datei erlaubt dem erfahrenen Nutzer eine einfache Steuerung des Systems, ohne die eigentliche Implementierung anfassen zu müssen. Durch das Einlesen und Verarbeiten einer solchen Datei entsteht kein nennenswerter Laufzeitverlust. Der aktuelle Stand erlaubt dem Benutzer lediglich die Konfiguration der Suchstrategie, d.h. die Art des *Labeling*, wobei auch Kombinationen mehrerer Labeling-Strategien möglich sind, und die Reihenfolge der *Tasks* für das *Labeling* können bestimmt werden. Dies soll künftig erweitert werden durch die Vorgabe der Reihenfolge der Variablen und der Domänenwerte, die bei der Variablenbelegung während der Suche eine Rolle spielen, sowie die Festlegung der Zielfunktion und des Timeouts, die das Optimierungskriterium bestimmen.

Literatur

[BPSM+08] Tim Bray, Jean Paoli Paoli, C.M. Sperberg-McQueen, Eve Maler und François Yergeau. Extensible Markup Language (XML) 1.0. Bericht, W3C, Nov 2008.

[HW07] P. Hofstedt und A. Wolf. *Einführung in die Constraint-Programmierung: Grundlagen, Methoden, Sprachen, Anwendungen*. Springer Verlag, 2007.

[Pro93] Patrick Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, 9(3):268–299, August 1993. (also available as technical report AISL-46-91, Stratchclyde, 1991).

[Wol06] Armin Wolf. Object-Oriented Constraint Programming in Java Using the Library `firstCS`. In Michael Fink, Hans Tompits und Stefan Woltran, Hrsg., *20th Workshop on Logic Programming, Vienna, Austria, February 22–24, 2006*, Jgg. 1843-06-02 of *INFSYS Research Report*, Seiten 21–32. Technische Universität Wien, 2006.

[4] Aktuell entspricht hier dem Zeitpunkt der Entstehung dieses Artikels.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html