Interaktive Komponenten in constraint-basierten Planungssystemen

Hans-Joachim Goltz

Fraunhofer FIRST, Berlin
hans-joachim.goltz@first.fraunhofer.de

Abstract: Diskutiert werden Möglichkeiten und Varianten der Integration von interaktiven Komponenten in constraint-basierten Systemen, insbesondere in Planungssystemen.Drei grundsätzlichen Varianten der interaktiven Einzelplanung werden kurz beschrieben.

1 Einleitung

Die constraintlogische Programmierung über endliche Domänenbereiche konnte in vielen Anwendungen zur Lösung komplexer diskreter Probleme, insbesondere für Planungsprobleme, sehr erfolgreich eingesetzt werden (siehe z.B. [St08, Wa96]). Es existieren eine Vielzahl von Arbeiten, in denen praktische Anwendungen der constraintbasierten Programmierung beschrieben werden. In den meisten Fällen werden aber nur Algorithmen beschrieben, die eine automatische Lösungssuche realisieren. In unseren Anwendungssystem verwenden wir erfolgreich eine kombinierte automatische und interaktive Lösungssuche (siehe z.B. [GM99, Go00, Sc01, GP09]). Möglichkeiten und Varianten der Integration von interaktiven Komponenten in praktischen Anwendungen der constraintlogische Programmierung werden im Folgenden diskutiert.

Die Beschreibung der Algorithmen erfolgt teilweise in einem Pseudocode, der sich an die Syntax der constraintlogischen Programmiersprache CHIP orientiert (siehe z.B. [Di88]). Die Syntax von CHIP basiert auf PROLOG. Zusätzlich können Objekte mit Attributen, die auch Variablen sein können, definiert werden. Mit "`Obj@Attr`" kann auf den Wert eines Attributes zugegriffen und durch "`Obj@Attr <- Val`" ein Wert gesetzt werden.

2 Interaktionen bei der Lösungssuche

In erfolgreichen praktischen Anwendungen ist es oft wichtig, das der Nutzer nicht nur die Möglichkeit einer automatischen Lösungserzeugung besitzt, sondern dass er durch Nutzeraktionen einen direkten Einfluss auf die Erzeugung von Lösungen be-

sitzt. Nicht immer lassen sich alle Bedingungen quantitativ spezifizieren. Dann muss aber die Möglichkeit bestehen, dass automatisch erzeugte Lösungen durch den Nutzer modifiziert werden können. Natürlich dürfen nur solche Nutzeraktionen erlaubt werden, die die gegebenen Constraints alle erfüllen. Gerade die constraintbasierte Programmierung ist dafür besonders gut geeignet.

Durch Nutzeraktionen besteht auch die Möglichkeit, in ein Anwendungssystem die Eigenschaft zu integrieren, dass bestimmte Constraints in Ausnahmefällen verletzt werden können. So dürfen bei einer automatischen Schichtplanung die Schichtbedingungen natürlich nicht verletzt werden. Bei einer interaktiven Einzelplanung könnte man dies aber in Ausnahmefällen erlauben, da dann die Verletzungen der Schichtregeln individuell bestätigt werden müssen. Bei der Raumplanung in einer Universität dürfte es beispielsweise nicht sinnvoll sein, das ein Seminar mit wenigen Studenten in einem Vorlesungssaal stattfindet. Bei einer automatischen Planung sollte eine solche Zuordnung nicht erlaubt werden. Bei einer interaktiven Einzelplanung könnte dies aber wieder als Ausnahme ermöglicht werden. Folgende Möglichkeiten der Nutzeraktionen können unterschieden werden, wobei grundsätzlich vorausgesetzt wird, dass der Nutzer Objekte als ausgewählt markieren kann: *alle automatisch bzw. alle Restlichen automatisch einplanen, markierte einplanen, markierte ausplanen, alle ausplanen, Einzelplanung.*

Für die Akzeptanz eines interaktiven Systems ist es wichtig, dass kein automatisches Backtracking von Nutzerkationen erfolgt. Außerdem sollte jeder Planungszustand gespeichert werden können (auch in einer Datei) und die Constraints müssen mit Einbeziehung des aktuellen Planungszustandes erzeugbar sein. In Anwendungssystemen kann es weiterhin sehr wichtig sein, wenn verschiedene Planungszustände der aktuellen Sitzung gespeichert werden können. So könnte beispielsweise immer der vorletzte Planungszustand gespeichert werden, so dass ein einmaliges Zurücksetzen auf einen vorangegangenen Zustand ermöglicht wird. Außerdem könnte eine vorgegebene maximale Anzahl von Planungszustände gespeichert werden können und das Vertauschen von Planungszuständen erlaubt werden. Insbesondere dadurch können dann leicht "Was-Wäre-Wenn"-Simulationen realisiert werden. Im Folgenden werden drei grundsätzlichen Varianten der interaktiven Einzelplanung diskutiert.

3 Varianten der interaktiven Einzelplanung

Für eine interaktive Einzelplanung existieren unterschiedliche Varianten. In Abhängigkeit vom Art des Problems und dem Ziel der Nutzeraktionen ist eine geeignete Variante zu wählen. Folgende Varianten der interaktiven Einzelplanung werden im Folgenden diskutiert:

1. *Umplanung nach Nutzeraktion:* Durch eine Aktion des Nutzers in der grafischen Oberfläche wird eine Umplanung erforderlich.
2. *ein einzelnes Objekt wird interaktiv eingeplant:* Für ein ausgewähltes Objekt werden die möglichen Werte grafisch dargestellt, die der Nutzer auswählen

kann. Nach einer erfolgreichen Wertezuordnung ist eine solche interaktive Einplanung beendet.

3. *interaktive Einplanung als allgemeine Aktion:* Der Nutzer wählt als allgemeine Aktion die interaktive Einplanung und das System muss dann alle Constraints erzeugen. Nachdem der Nutzer ein einzuplanendes Objekt ausgewählt hat, müssen für dieses Objekt die auswählbaren Werte grafisch dargestellt werden. Anschließend kann der Nutzer ein anderes Objekt auswählen oder die interaktive Einzelplanung beenden.

3.1 Umplanung nach Nutzeraktion

Das Erzeugen der Constraints und die Umplanung wird durch eine Aktion des Nutzers in der grafischen Oberfläche gestartet. Solche Nutzeraktionen können beispielsweise sein:

- Ein Objekt wird verschoben, was als eine Änderung der Startzeit interpretiert werden muss.
- Eine Ressource ist nicht mehr verfügbar.
- Eine Ressourcenverfügbarkeit wird zeitlich oder kapazitativ eingeschränkt.

Nach einer solchen Nutzeraktion muss das System eine Umplanung starten, die möglichst die gewünschte Nutzeraktion berücksichtigt. Für die Akzeptanz eines solchen Systems ist es wichtig, dass nicht die Umplanung aller Objekte erlaubt wird. Für eine erlaubte Umplanung können Prioritäten eingeführt werden. Im Folgenden werden drei verschiedene Prioritäten betrachtet:

- *umplanbar:* darf immer umgeplant werden;
- *eingeschränkt umplanbar:* eine Umplanung sollte möglichst vermieden werden;
- *fest geplant:* darf nur umgeplant werden, falls eine Ressourcenzuordnung nicht mehr möglich ist; ein zeitliches Verschieben ist nicht erlaubt.

Auf eine Nutzeraktion muss dass System wie folgt reagieren, wobei angenommen wird, dass das Ziel der Umplanung eine möglichst geringe Änderung des vorangegangenes Planes ist:

1. die relevanten Constraints erzeugen: für die umplanbaren und eingeschränkt umplanbaren Objekte so, als wären sie nicht eingeplant; für ein fest geplantes Objekt mit einer ungültigen Ressourcenzuordnung so, als wäre es nicht eingeplant; für alle anderen fest geplanten Objekten mit allen Zuordnungen der Einplanung.
2. eine neue Lösung suchen: zuerst für ein fest geplantes Objekt mit einer ungültigen Ressourcenzuordnung eine neue Ressourcenzuordnung finden, wobei die

Startzeiten möglichst wenig geändert werden sollten; dann werden die anderen Objekte in folgender Reihenfolge eingeplant: die beschränkt umplanbaren Objekte, ein eventuell vorhandenes verschobene Objekt und zum Schluss die umplanbaren Objekte; bei der Einplanung werden generell die alten Planungswerte bevorzugt und die Abweichung der Zeitzuordnung zum alten Wert wird möglichst minimiert.

3.2 Interaktive Einplanung als Einzelaktion

Der Nutzer wählt ein Objekt für die interaktive Einzelplanung aus. Das Ziel der Aktion kann dabei sowohl eine Einplanung als auch die Modifikation einer Einplanung sein. Bei der Modifikation muss das System das Objekt zuerst intern ausplanen. Das System muss dann alle Constraints erzeugen, die für das ausgewählte Objekt relevant sind und die auswählbaren Werte für die Einplanung grafisch darstellen. Dann muss das System auf einen relevanten Mausklick des Nutzers in der grafischen Nutzeroberfläche warten. Der Nutzer kann so einen möglichen Wert auswählen. Nach einer erfolgreichen Wertzuordnung ist diese interaktive Gesamtaktion beendet. Neben der Wertauswahl müssen mindestens noch zwei weitere Fälle einer Nutzeraktion betrachtet werden: "*Abbruch der interaktiven Einzelplanung*" und "*es wird kein möglicher Wert ausgewählt*".

Im Folgenden wird die Grundstruktur eines Algorithmus für eine solche interaktive Einzelplanung angegeben. Dabei bezeichnet `search@interactive` ein Attribut der Klasse `search` zur Kontrolle der interaktiven Einzelplanung.

```
start_search_interactive(Object) :-
   generate_constraints(Object), % alle relevanten Constraints erzeugen
   additional_propagation(Object), !, % siehe Abschnitt 4
   search_interactive(Object).
start_search_interactive(_) :-
   writeln('There is not any valid value, scheduling is not possible').

search_interactive(Object) :-
   show_domain(Object), % die möglichen Werte grafisch darstellen
   search@interactive <- on,
   waiting_of_user_action,  % warten bis search@interactive \= on
   search1_interactive(search@interactive,Object).

search1_interactive(out, Object) :- !, % keinen gültigen Wert ausgewählt
   search@interactive <- on,
   waiting_of_user_action,  % das Warten wird fortgesetzt
   search1_interactive(search@interactive,Object).
search1_interactive(stop, _) :- !, % Abbruch der Einzelplanung
   redraw_gantt. % grafische Darstellung aktualisieren
search1_interactive(Value, Object) :-
   integer(Value),
```

```
   check_value_choice(Value,Object), !, % Wert zuordnen und überprüfen
   writeln('Interactive search was successful'),
   redraw_gantt. % grafische Darstellung aktualisieren
search1_interactive(Value, Object) :-
   writeln('There is a conflict if this value is selected'),
   notin_domain(Value,Object), !, % Wert aus dem Domänbereich streichen
   search_interactive(Object). % neue Werteauswahl ermöglichen
search1_interactive(Value, _) :-
   writeln('Interactive search was not successful'),
   redraw_gantt. % grafische Darstellung aktualisieren
```

3.3 Allgemeine interaktive Einplanung

Der Nutzer wählt als allgemeine Aktion die interaktive Einplanung und das System muss dann alle Constraints erzeugen. Nachdem der Nutzer ein einzuplanendes Objekt ausgewählt hat, muss das System die auswählbaren Werte darstellen. Nach der Auswahl eines Wertes und einer erfolgreichen Wertzuordnung kann der Nutzer ein weiteres Objekt zur Einplanung auswählen.
Gegenüber der Einzelaktion einer interaktiven Einplanung muss folglich noch eine "Warteschleife" für die Auswahl eines Objektes vorangestellt werden. Folgende Nutzeraktionen könnten noch integriert werden:

- Anstatt eines Wertes der grafisch dargestellten möglichen Werte wählt der Nutzer ein anderes Objekt. Dies bedeutet einen Abbruch der aktuellen Wertezuordnung und ein Start einer neuen Wertezuordnung.
- Der Nutzer wählt ein Objekt, das bereits eingeplant ist. Dies ist als Wunsch einer Modifikation der Einplanung zu interpretieren. Das ausgewählte Objekt ist zuerst auszuplanen und alle Constraints müssen neu erzeugt werden. Algorithmisch kann dies beispielsweise über ein Backtracking der bereits erfolgten Erzeugung der Constraints und einer catch-throw-Konstruktion realisiert werden. Anschließend kann das ausgewählte Objekt wie ein anderes einzuplanendes Objekt behandelt werden.

4 Zusätzliche Propagation

Ein Constraintlöser über endliche Domänenbereiche ist im Allgemeinen nicht vollständig. Im Domänenbereich einer Constraintvariable können folglich Werte enthalten sein, die bei einer direkten Zuordnung sofort einen Widerspruch erzeugen. Für eine interaktive Einzelplanung ist es sehr ungünstig, wenn auswählbare Werte dargestellt werden, die eigentlich nicht zur Auswahl stehen. Für die Akzeptanz einer interaktiven Einzelplanung ist es deshalb wichtig, eine zusätzlich Propagation zu aktivieren, die solche Werte aus den Domänen vorher möglichst streicht. Ein

einfacher Algorithmus für eine zusätzliche Propagation besteht darin, alle aktuell erlaubten Werte der relevanten Constraintvariable zu überprüfen, ob sie bei einer direkten Zuordnung einen Widerspruch erzeugen würden, und die Werte mit Widerspruch aus dem Domänenbereich zu streichen. Ein solcher Algorithmus kann wie folgt implementiert werden:

```
dom_check(DomVar) :-
   dvar(DomVar),
   !,
   domain(DomVar,ValueList),
   dom_check1(LValue,DomVar).
dom_check(_).

dom_check1([],_).
dom_check1([N|List],DomVar) :-
   not N = DomVar,
   !,
   DomVar #\= N,
   dom_check1(List,DomVar).
dom_check1([N|List],DomVar) :-
   dom_check1(List,DomVar).
```

Für eine Constraintvariable `DomVar`, erzeugt `domain(DomVar,ValueList)` die Liste der Werte `ValueList` , die zum Domänenberich von `DomVar` gehören. Die Relation `DomVar #\= N` bedeutet, dass `DomVar` und `N` verschieden sind, und folglich dass der Wert `N` aus dem Domänenbereich von `DomVar` gestrichen wird. Entsprechend dem Problem könnte der Test `N = DomVar` auch durch einen umfangreicheren Test ersetzt werden. Zu beachten ist aber, dass die zusätzliche Propagation nicht zuviel Zeit benötigt, da der Nutzer interaktiv mit dem System arbeiten möchte.

Literatur

[Di88] Dincbas,M.; van Hentenryck,P.; Simonis,H.; Aggoun,A.; Graf,T.; Berthier,F.: The constraint logic programming language CHIP. In *Int. Conf. Fifth Generation Computer Systems (FGCS'88)*, Tokyo 1988, S. 693–702.

[Go00] Goltz, H.-J.: Combined automatic and interactive timetabling using constraint logic programming. In (E. Burke and Erben,W. Hrsg.): *PATAT 2000, Proc. Int. Conf. Practice and Theory of Automated Timetabling 2000*, S. 78–95.

[GM99] Goltz,H.-J.; Matzke,D.: University timetabling using constraint logic programming. In (Gupta,G., Hrsg.): *Practical Aspects of Declarative Languages*, vol. 1551 of *Lecture Notes in Computer Science*, Springer-Verlag, 1999, S. 320–334.

[GP09] Goltz,H.-J.; Pieth,N.: A Tool for generating Partition Schedules of Multiprocessor Systems. In (Geske,U.; Wolf,A., Hrsg.): *Proc. 23^{rd} Workshop on (Constraint) Logic Programming 2009*, Universitätsverlag Potsdam 2010, S. 167–176.

[Sc01] Schlenker,H.; Goltz,H.-J.; Oestmann,J.W.: TAME – Time Resourcing in Academic Medical Environments. In *AIME 2001, Proceedings Int. Conf.*, LNAI 2101, Springer Verlag, 2001, S. 395-404.

[St08] Stuckey,P.J. (Hrsg.): Principles and Practice of Constraint Programming – CP 2008. Vol. 5202 of *Lecture Notes in Computer Science*, Springer-Verlag, 2008.

[Wa96] Wallace,M.: Practical Applications of Contraint Programming. *Constraints, An International Journal*, 1:139–168, 1996.