Zur Verarbeitung «weicher» Constraints

Harald Meyer auf'm Hofe, Andreas Abecker

Sogenannte «weiche» Constraints ermöglichen die Repräsentation von unscharfen Relationen, Präferenzen, Kostenmaßen und Vermutungen in Constraintproblemen. Mittlerweile wurden einige Formen «weicher» Constraints inklusive unterschiedlicher Techniken zu ihrer Verarbeitung vorgeschlagen. Zu nennen sind hier z.B. *Constraintgewichte* und Constraints mit Prioritäten. Dieser Artikel stellt Gemeinsamkeiten und Unterschiede dieser Formalismen dar und bettet diese Ansätze in das Schema der *hierarchical constraint satisfation problems (HCSP)* ein, das am DFKI zur Repräsentation von *scheduling*- und Konfigurierungsproblemen entwickelt wurde.

1 Einleitung

Verfahren zur Verarbeitung von Constraints [10,15] finden zunehmende Beachtung als zentrale Inferenz in der Konfigurierung, bei Planungs- und bei Scheduling-Problemen. Viele dieser Aufgaben lassen sich als einfache Zuordnungsprobleme mit Randbedingungen — d.h. *Constraint satisfaction problems (CSP)* — beschreiben: einer endlichen Menge *V* von Variablen werden Werte aus einer Domäne *D* zugewiesen; diese Zuweisungen müssen Constraints aus einer endlichen Menge *C* genügen, die jeweils nur bestimmte Kombinationen von Belegungen für einen Teil der Variablen (ihren lokalen Variablen) erlauben. Die Aufzählung aller erlaubten Kombinationen von Belegungen der lokalen Variablen liefert die Extension eines Constraints. Das CSP besteht darin, einzelne (mehrere, alle) Belegungen der Variablen zu finden, die mit allen Constraints verträglich sind. Modelliert man reale Probleme mit Constraintformalismen, so tauchen schnell einige Unfreundlichkeiten auf, die verschiedene Erweiterungen des einfachen Basisformalismus erfordern: Problemanforderungen sind verschieden wichtig, Kosten sind zu minimieren, Begriffe unscharf definiert usw. Wir beginnen dieses Papier mit einem kurzen Überblick solcher Erweiterungen und der für sie entwickelten Techniken. Es zeigt sich, dass man alle nach dem gleichen Grundprinzip verstehen kann: Constraints werden «aufgeweicht», d.h. ihre Einhaltung wird nicht mehr *zwingend verlangt*, sondern *erwünscht*. Die Problemlösung wird damit zum Optimierungsproblem: Es muss eine Belegung gesucht werden, die in optimaler Art und Weise mit den Constraints verträglich ist. Wir werden zeigen, dass die bekannten Erweiterungen des einfachen Constraintformalismus sich nur in der Art und Weise unterscheiden, in der die Verträglichkeit bewertet wird. Solche Optimierungen sind zuerst einmal wesentlich komplexer als einfache «harte» CSPs. Zu ihrer Lösung benutzt man typischerweise Varianten des *branch&bound* Algorithmus [13] — insbesondere dann, wenn der Nachweis der Optimalität der Lösung erforderlich ist. Da jede reale Anwendung eine gemischte Verwendung der diversen Erweiterungen erfordert, zeigen wir, wie sich zwei der vorgestellten Formalismen in das umfassendere Schema der *hierarchical constraint satisfation problems (HCSP)* einbetten lassen. Diese Verbindung erlaubt es, Constrainttechniken, die für die einzelnen Formalismen entwickelt wurden, gemeinsam zur Erweiterung des *branch&bound* einzusetzen. Hierzu beschreiben wir einige Experimente mit dem ConPlan-Solver [12], welche die möglichen Auswirkungen auf die Performanz des *branch&bound* beleuchten.

2 Ansätze zur Darstellung «weicher» Constraints

Harald Meyer auf'm Hofe studierte Informatik an der Universität Kaiserslautern und arbeitet seitdem am Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI GmbH) an der Erweiterung von Verfahren der Constraintverarbeitung und deren Anwendung in Scheduling-Systemen und der wissensbasierten Konfigurierung.

Andreas Abecker studierte Informatik und Wirtschaftswissenschaften in Kaiserslautern. Als wissenschaftlicher Mitarbeiter am DFKI befaßte er sich mit Fragen der Wissensrepräsentation und Anwendungen des Maschinellen Lernens. Sein momentanes Arbeitsgebiet ist das Wissensmanagement in Unternehmen.

Die wichtigsten Erweiterungen in Richtung weicher Constraints sind:

Constraintgewichte [5]: Jedem Constraint *c* wird ein reellwertiges Gewicht $\omega(c)$ zuordnet. Ein Lösungsverfahren versucht, die Gewichtssumme verletzter Constraints so klein wie möglich zu halten. Der Ansatz wird typischerweise zur Kostenoptimierung verwendet. Ein prominenter Spezialfall ist das Problem, so viele Constraints wie möglich zu erfüllen (Constraintrelaxierung, auch *maximal constraint satisfaction* genannt).

Constraintprioritäten: Jedes Constraint *c* besitzt eine Priorität $p(c) \in]0;1]$. Eine Lösung ist dann optimal, wenn die Priorität des wichtigsten verletzten Constraints möglichst gering ist. Das Verfahren kann die unterschiedliche Bedeutung von Constraints widerspiegeln (vgl. [4,11]) und z.B. zur Einführung von Defaults, optionalen Anforderungen an eine Problemlösung oder vermuteter Eigenschaften einer Lösung dienen.

Fuzzy-Constraints [4,6]: Einzelne Variablenbelegungen werden mit einem Fuzzy-Wert versehen, der angibt, inwieweit sie das entsprechende Constraint erfüllen. Ziel ist das Erreichen eines maximalen Erfüllungsgrades bzgl. *jeden* Constraints auf dem gesamten Constraintnetz. Einsatzgebiet ist die Modellierung von Unsicherheit und Vagheit bei Begriffen und Zusammenhängen. Ein Fuzzy-Constraint kann als Repräsentation u.U. recht vieler Constraints mit Prioritäten verstanden werden.

Von den hier angeführten Formalismen können wir die ersten beiden als sog. *crisp constraints* zusammenfassen: Ein einzelnes Constraint wird immer entweder erfüllt oder verletzt; für die Bewertung einer Belegung betrachtet man die Gesamtsituation im Constraint-Netzwerk. Fuzzy-Constraints sind dagegen insofern grundsätzlich anders, als bei ihnen schon die Erfülltheit eines einzelnen Constraints quantitativ bewertet wird, d.h. es sind *zum Teil erfüllte* Constraints möglich.

Eine weitere Form weicher *crisp* Constraints sind die Constrainthierarchien des *hierarchical constraint logic programming (HCLP)* [3]. Dieser Formalismus liefert einen Rahmen für die Kombination mehrerer Gütekriterien einer Lösung. Wir werden in ähnlicher Art und Weise die Ausdrucksmöglichkeiten von gewichteten Constraints und Constraint-Prioritäten kombinieren, was dann sowohl die deklarative Beschreibung beider Phänomene als auch die integrierte Nutzung der für die beiden Erweiterungen bekannten effizienzverbessernden Verfahren ermöglicht. Zur kombinierten Beschreibung führen wir zuerst als Basisformalismus Präferenzordnungen über Constraintmengen ein und definieren mit ihrer Hilfe Prioritäten und Gewichte.

2.1 Präferenzordnungen

Alle oben beschriebenen Formen weicher Constraints verhalten sich insofern gleich, als sie eine (quantitative) Bewertung vollständiger Variablenbelegungen (d.h. potentieller CSP-Lösungen) induzieren, welche einen Vergleich der Lösungsgüte ermöglicht.

Wir können diese Bewertung als Partialordnung $>$ zwischen Belegungen verstehen, deren Semantik wir als «*ist bessere Lösung als*» interpretieren. Zur Herleitung dieser Ordnungsrelation aus den gegebenen Formalisierungen weicher Constraints betrachten wir jeweils die Menge von Constraints, die eine Belegung verletzt.

Eine Belegung ist eine bessere Lösung, wenn sie eine *«weniger wichtige»* Menge von Constraints verletzt. Die Wichtigkeit von Constraintmengen lässt sich wiederum am besten durch eine Partialordnung $\rangle$ über Mengen von Constraints – den Präferenzordnungen – ausdrükken, deren Bedeutung sich mit *«ist bevorzugterweise zu erfüllen als»* übersetzen lässt. Bezeichnet *C(L)* die Menge der Constraints, die durch die Belegung L erfüllt werden, so gilt:

$$L' > L'' \text{ gdw. } C \setminus C(L'') \rangle C \setminus C(L').$$

Wie das Beispiel der Constraintprioritäten zeigen wird, ist es (auch aus technischen Gründen) angenehmer, die Güte einer Lösung über die Wichtigkeit der verletzten Constraints zu definieren.

Durch geeignete Definition einer Präferenzordnung lassen sich mit diesem Schema alle Formalismen von *crisp constraints* ausdrücken[1]. Unter der Voraussetzung, dass Constraints eine beliebige Stelligkeit haben können, lassen sich nahezu alle Optimierungsprobleme durch Präferenzordnungen ausdrücken.

So sieht die Ordnung für **gewichtete Constraints** z.B. wie folgt aus, wobei C' und C'' für Constraintmengen stehen:

$$C' \rangle_{\omega} C'' \text{ gdw. } \sum_{c' \in C'} \omega(c') > \sum_{c'' \in C''} \omega(c'').$$

Die Gewichte werden wie Kostenfaktoren addiert.

Constraintprioritäten ergeben dagegen folgende Präferenzordnung:

---

[1] Da *non-crips constraints* teilweise erfüllt und teilweise verletzt sein können, kann hier allerdings aus einer Präferenzordnung über Constraints keine Ordnungsrelation über Lösungen hergeleitet werden.

$C' \rangle_p C''$ *gdw.* $MAX_{c' \in C'}\ p(c') > MAX_{c'' \in C''}\ p(c'')$.

Im Unterschied zu den gewichteten Constraints ist hier ein Constraint höherer Priorität wichtiger als alle Constraints niedrigerer Priorität zusammengenommen. Eine Lösung des Problems ist bis zu einer bestimmten Priorität mit allen Constraints verträglich. Entscheidend für die Bewertung einer Lösung ist also das wichtigste verletzte Constraint.

Wenn sich nun beide Formalismen in einfacher Weise durch Präferenzordnungen darstellen lassen, stellt sich die Frage, ob man mithilfe dieser Technik nicht auch Eigenschaften beider Darstellungen in homogener Weise ausdrücken kann. Diese Frage wollen wir im nächsten Abschnitt beantworten.

2.2 Constrainthierarchien als integrierter Formalismus

In der praktischen Anwendung treten häufig die verschiedensten Modellierungsprobleme in Kombination und komplizierten Mischformen auf. Gerade Gewichte und Prioritäten müssen oft kombiniert werden. Es kommt vor, dass eine bestimmte Randbedingung wichtiger ist als alle weniger wichtigen Bedingungen zusammengenommen. Dies ist am einfachsten durch Constraintprioritäten auszudrücken. Andererseits ist die Erfüllung vieler Randbedingungen häufig wichtiger als die Erfüllung weniger. Dies kann *nicht* durch Constraintprioritäten ausgedrückt werden, ohne eine Vielzahl neuer Constraints einzufügen[2]. Die Lösung dieses Dilemmas besteht darin, die Menge der Constraints C in eine endliche Zahl von Hierarchieebenen $C_0,\ldots,C_m$ zu partitionieren. Jedes Constraint einer höheren Hierarchieebene (kleiner Index) soll wichtiger sein als alle niedrigeren Hierarchieebenen (höhere Indizes) zusammengenommen. Die wichtigste Ebene C_0 soll harte, unbedingt zu erfüllende Constraints enthalten. *Innerhalb* einer Hierarchieebene i werden die Constraints durch eine normale Präferenzordnung $\rangle_i$ unterschieden. Formal lässt sich die Präferenzordnung zu einer Hierachie $H=(C_0, (C_1, \rangle_1), (C_2, \rangle_2),\ldots)$ wie folgt als lexikalische Kombination der einzelnen Hierarchieebenen definieren:

$$C' \rangle_H C'' \quad \text{gdw. } C' \rangle^1 C'',$$
$$C' \rangle^i C'' \quad \text{gdw. } C' \cap C_i \rangle_i C'' \cap C_i \text{ oder } C' \rangle^{i+1} C''.$$

Die Präferenzen $\rangle_i$ auf den einzelnen Hierarchieebenen werden nacheinander ausgewertet. Können die fraglichen Constraintmengen anhand einer bestimmten Hierarchieebene geordnet werden, so ist die Präferenz bzgl. der gesamten Hierarchie bekannt. Ansonsten muss die nächst weniger wichtige Hierarchieebene herangezogen werden. Durch diese Kombination der Präferenzen $\rangle_i$ auf den einzelnen Ebenen eignen sich Constrainthierarchien damit beispielsweise zur Integration verschiedener Arten von Problemwissen wie Präferenz, Kosten oder Eigenschaften, die man bei den Lösungen des Problems *vermutet* [11].

Zwischen Constraintprioritäten und -hierarchien besteht eine Beziehung, von der bei den Suchalgorithmen noch die Rede sein wird (siehe Abschnitt 3.2). Für jedes Constraint c einer Hierarchiestufe i in der Hierarchie H ergeben sich z.B. mit $p(c)=1/_{i+1}$ Prioritäten, so dass für die entsprechenden Präferenzordnungen gilt:

$$C' \rangle_p C'' \Rightarrow C' \rangle_H C''.$$

Damit kann die Präferenz bzgl. einer Constrainthierarchie durch ein System von Constraints abgeschätzt werden, deren Präferenz durch eine Priorität gegeben ist. Zu Constraintprioritäten sind besondere Verfahren bekannt.

Die vorgestellte homogene Sichtweise bezeichnen wir als Schema der *hierarchical constraint satisfaction problems (HCSP).* Außer der Integration von *Darstellungsmöglichkeiten* für Präferenzordnungen liefert uns das HCSP-Schema einen weiteren Vorteil, der den Aufwand für die Entwicklung einer Constraint-Technologie erst rechtfertigt: Für die einzelnen Formalismen weicher Constraints wurden jeweils spezielle Verfahren vorgestellt, die besondere Eigenschaften der Problemrepräsentation ausnutzen (vgl. Abschnitt 3). Diese Verfahren ändern zwar an der Komplexitätsklasse kombinatorischer Optimierungsprobleme nichts, führen jedoch im Durchschnitt über die Probleminstanzen zu einer beträchtlichen Beschleunigung der Suche, ohne die die Anwendung dieser Formalismen undenkbar wäre. Unsere integrierte Sichtweise legt es nun nahe, diese getrennt entwickelten Verfahren zusammen zu betrachten und als Bestandteile eines modularen Systems zur Lösung kombinatorischer Suchprobleme zu implementieren. Durch die Kombination dieser Constraintverfahren lassen sich eine Vielzahl von Suchalgorithmen bilden, wobei synergetische Effekte zwischen den einzelnen Verfahren eine große Rolle spielen.

[2] Vgl. [12] für eine Beispielanwendung einer derartigen Kombination von Präferenzkriterien.

3 Suchverfahren

Optimierungsaufgaben von der Natur weicher Constraintprobleme bearbeitet man häufig mithilfe der vollständigen Baumsuche[3]. Abbildung 1 skizziert den *branch&bound* Algorithmus [13,5], das typische Verfahren zu diesem Zweck. In dieses Schema werden die anwendbaren Constraintverfahren eingeordnet. *Bound b* und *distance* δ haben eine etwas ungewöhnliche Gestalt, weil es sich im Sinne des vorgestellten Formalismus weicher Constraints um Constraintmengen handelt, die durch die Präferenzordnung ⟩ verglichen werden. Der Algorithmus geht folgendermaßen vor: Eine unbelegte Variable wird ausgewählt (Anweisung 4), die möglichen Belegungen der Variablen werden getestet und die Suche verzweigt (Anweisung 5), wenn die aktuelle Belegung *L* nicht wichtigere oder gar harte Constraints verletzt als die im *bound b* gespeicherten (Anweisung 7). Die durch die aktuelle Belegung verletzten Constraints werden in der *distance* δ verwaltet. Wird ein Blattknoten des Suchbaums erreicht, haben also alle Variablen eine Belegung, und die aktuelle Belegung ist besser als der *bound*, so wird sie zur neuen Lösung (Anweisung 2). Andernfalls ändert sich nichts.

Die möglichen Verbesserungen dieses Grundalgorithmus setzen bei den einzelnen nicht determinierten Stellen an, die in Abbildung 1 enthalten sind: Man kann das Verfahren als ein suchendes Durchlaufen des Baumes aller möglichen Variablenbelegungen verstehen, das verschränkt ist mit dem Kappen nicht mehr weiter zu betrachtender Teilbäume. Dabei wird die Navigation im Suchraum von zwei Funktionen gesteuert: einerseits durch die Auswahl der als nächstes zu belegenden Variablen (Anweisung 4) und andererseits durch die Reihenfolge der Werte, mit denen die Variable belegt wird (Anweisung 5). Das Kappen uninteressanter Teilbäume geschieht entweder aufgrund erkannter Widersprüche, oder weil der Suchbaum eine Belegung zu schlechter Qualität repräsentiert. Die frühzeitige Erkennung solcher Situationen wird durch die Art des verwendeten Konsistenztests (Anweisung 6) unterstützt. Im folgenden diskutieren wir diese drei Ansatzpunkte.

3.1 *Forward checking*

[3] Ist der Nachweis der Optimalität einer Lösung nicht erforderlich, so stehen alternativ lokale Suchverfahren zur Verfügung. Vergleiche [12] für eine kommerzielle Anwendung eines speziellen lokalen Suchverfahrens, in dem die hier dargestellten Verfahren ebenfalls Anwendung finden.

```
branch&bound(S, b, L, δ, V, C, D, ⟩)

1.  if |L| = |V| then
2.      if  b ⟩ δ then b ← δ; S ← L; endif;
3.      return S; endif;
4.  wähle ein v∈ V, das nicht in L belegt;
5.  foreach d∈ D do
6.      δ_lokal = {c ∈ C | c ist inkonsistent mit v←d };
7.      if δ_lokal ∩ C_0 = {} und b ⟩ δ_lokal ∪ δ  then
            S ← branch&bound  (S,
                               b,
                               L ∪ { v ← d },
                               δ ∪ δ_lokal,
                               V, C, D); endif;
8.  endforeach;
9.  return S;

    Initialer Aufruf:
    branch&bound({}, C, {}, {}, V, C, D, ⟩)
```

Abbildung 1: Schema des *branch&bound*.

Die Konsistenztests können die klassischen Constrainttechniken zur Propagierung von Domänen verwenden. Dann werden in Anweisung 6 aus den Domänen der unbelegten Variablen alle Werte entfernt, die mit der Belegung *v←d* unverträglich sind. Entweder werden dabei nur die Constraints propagiert, die v als lokale Variable haben *(forward checking FC*) [7] oder es wird z.B. Kantenkonsistenz zwischen den Domänen der unbelegten Variablen, den Constraints und den aktuell gültigen Belegungen hergestellt. Letzteres Verfahren verlangt die Verwendung eines AC-Algorithmus [9] nach jeder Zuweisung, weshalb es auch *maintaining arc-consistency (MAC)* heißt. Wird dabei irgendeine Domäne leer, so besteht eine Inkonsistenz, und ein *backtracking* wird eingeleitet.

Ein *FC* in Anweisung 6 kann auch weiche Constraints [5] berücksichtigen. Die Unverträglichkeit eines Wertes in der Domäne einer unbelegten Variable mit den weichen Constraints und den aktuellen Belegungen führt nicht unweigerlich zu dessen Entfernung. Typischerweise markiert der *forward check* die Werte mit den Constraints, mit denen sie unverträglich sind. Erst wenn diese Marke zusammengenommen mit der *distance* δ bzgl. der Präferenzordnung $\rangle_H$ größer ist als der *bound b*, kann der entsprechende Wert aus der Domäne entfernt werden. *FC*-Marken ermöglichen es desweiteren, die Reihenfolge, in der die Werte einer Domäne zugewiesen werden (Anweisung 5 in Abbildung 1), zu optimieren. Es werden die Werte zuerst herangezogen, die bislang die unwichtigsten Constraints verletzen [5]. Diese Form der Gradientenmethode hat jedoch einen Nachteil: Zur Information der Suche werden nur die Constraints herangezogen, die eine belegte lokale Variable haben. Dieses

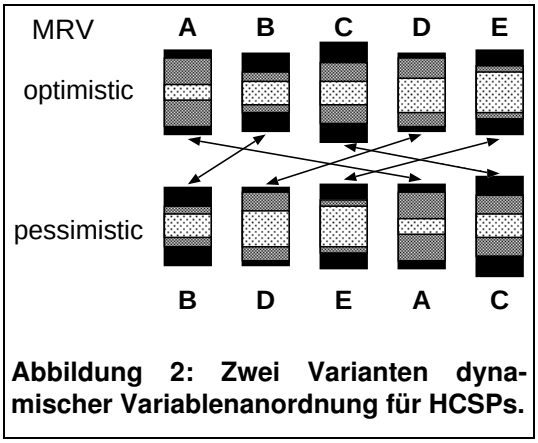

Abbildung 2: Zwei Varianten dynamischer Variablenanordnung für HCSPs.

Kriterium ist damit gerade in größeren Suchräumen nur von lokal begrenzter Aussagekraft.

3.2 Kompatibilität

Für Constraints mit Prioritäten und Fuzzy-Constraints ist der MAX-MIN-Algorithmus vorgestellt worden, der unscharfe, kantenkonsistente Domänen berechnet [14]. Dazu wird jedem Wert in der Domäne einer Variablen eine Kompatibilität μ zugewiesen, die die größte Priorität ist, bis zu der mit allen Constraints eine Kantenkonsistenz hergestellt werden kann. Die Kompatibilität stellt aufgrund der lokalen Konsistenz, die hergestellt wird, eine Obergrenze für die erreichbare Qualität einer Lösung dar. Kompatibilitäten können daher in der *branch&bound*-Suche in Systemen von Constraints mit Prioritäten wie die Ergebnisse des *FC* weicher Constraints genutzt werden: Zur Ausfilterung von Werten, die nicht Teil einer Lösung sein können und zur Information der Gradientenmethode bei der Belegung der Variablen. Der Vorteil dieses Verfahrens ist, dass die gesamte Constraintmenge berücksichtigt wird [1]. Der Nachteil des Verfahrens ist, dass mehr Constrainttests benötigt werden als z.B. bei einem normalen *forward checking (FC).*

Ein Vorteil von HCSPs ist es, konsistente Kompatibilitäten eingeschränkt nutzen zu können. Dazu wird die Implikation aus Abschnitt 2.2 genutzt. Der MAX-MIN-Algorithmus wird verwendet, um die wichtigste Hierarchieebene abzuschätzen, die nicht mehr vollständig erfüllt werden kann. Diese Schätzung benutzen wir dann wie bei der Lösung von Constraintproblemen mit Prioritäten.

3.3 Variablenanordnung

Ein Freiheitsgrad der Baumsuche wurde noch nicht behandelt: Die Reihenfolge, in der die Variablen belegt werden. Ein Beispiel für eine statische Variablenanordnung während der Suche ist die *maximum width ordering (MWO)* [16]. Hier wird diejenige Variable ausgewählt, die durch die meisten Constraints mit bereits belegten Variablen verbunden ist. Man hofft, dass dadurch weniger kritische Variablenbelegungen zuerst durchgeführt werden, also im Suchbaum oben stehen. Zur Revision dieser ersten Variablenbelegungen muss bei der Suche das tiefste und damit aufwendigste *backtracking* durchgeführt werden. Die *MWO*-Heuristik ist statisch in dem Sinne, dass die Reihenfolge der Variablen in allen Zweigen des Suchbaums dieselbe ist. Bei dynamischen Heuristiken ist dies nicht der Fall, weil sie Zwischenergebnisse des Konsistenztests (*FC* oder *MAC*) ausnutzen, die während des Suchprozesses anfallen. Die *minimal remaining values (MRV)* Heuristik [2], auch *first fail principle* genannt, ist ein Beispiel für eine dynamische Heuristik für die Anordnung der Variablen im Suchbaum. Hier wird als nächstes die Variable belegt, deren Domäne die wenigsten mit den bisherigen Zuweisungen verträglichen Werte enthält. Dabei wird die Verträglichkeit der Werte durch die Ergebnisse des *FC* oder *MAC* harter Constraints festgestellt. Durch die Verwendung dieser Propagierungsergebnisse berücksichtigen dynamische Variablenanordnungen auch die Eigenschaften der Extension der Constraints, während statische Anordnungen keinen Unterschied zwischen schwer zu erfüllenden und leicht zu erfüllenden Constraints machen. Der Effekt der dynamischen Heuristiken hängt damit sehr stark von den verfügbaren Propagierungsergebnissen ab. Bei der Bearbeitung von HCSPs ist die Berücksichtigung konsistenter Kompatibilitäten wünschenswert.

Es ergeben sich prinzipiell zwei Möglichkeiten, die in Abbildung 2 dargestellt sind. Jedes der Kästchen symbolisiert die Domäne einer Variablen. Die Größe der Kästchen soll der Anzahl der nicht durch harte Constraints ausgefilterten Werte der dargestellten Domäne entsprechen. Die Schattierungen stellen die Konsistenz mit den Hierarchieebenen dar — je heller, desto tiefer liegt die Hierarchieebene, bis zu der Konsistenz besteht. Bei der oberen der dargestellten Varianten wird zuerst die Anzahl der Werte betrachtet, die mit den meisten Hierarchieebenen konsistent sind. Nur anhand dieses Kriteriums nicht zu unterscheidende Variablen werden anhand der nächsten Kompatibilitätsstufe geordnet usw. Diese Variante der MRV-Heuristik wird optimistisch genannt. Es wird angenommen, dass alle bislang konsistenten Hierarchieebenen auch durch eine vollständige Belegung erfüllt werden können. Trifft diese Annahme nicht in ausreichendem Maße zu, so ist die Heuristik nicht sinnvoll. Bei der pessimistischen Variante wird genau umgekehrt vorgegangen. Zunächst wird die Anzahl nicht durch Propagierung harter Constraints ausgefilterter Werte betrach-

Nr.	10 Variablen, 10 Werte pro Domäne, 45 Constraints in 6 Hierarchiestufen (p_1=1.0), Erfüllb. p_2=0.5	Zeit /sec.	Zeit/$_{sec.}$: Minimum Maximum	Knoten	Knoten: Minimum Maximum	Tests /10^5	Tests/10^5: Minimum Maximum
1.	BM + FC alle	46.3	7.8 162.1	4701	690 15312	16.39	2.79 58.46
2.	BM + FC alle, MWO	13.4	3.8 36.7	1177	385 2916	4.74	1.32 12.92
3.	BM + FC alle, MRV + MWO	7.2	3.0 13.7	545	248 981	2.53	1.03 4.78
4.	BM + FC alle, MRVpess + MWO	6.8	3.2 11.5	485	236 794	2.25	1.01 3.91
5.	BM + FC alle, MRVopt + MWO	8.6	3.4 16.7	659	260 1228	2.72	0.99 5.47
6.	BM + MAC hart, BM + FC alle, MRV + MWO	7.2	3.1 13.5	497	221 923	2.55	1.11 4.77
7.	BM + MAC hart, BM + FC alle, MRVpess + MWO	6.6	3.2 11.5	438	207 725	2.25	1.05 4.04
8.	BM + MAC hart, BM + FC alle, MRVopt + MWO	8.4	2.9 15.3	584	188 1081	2.75	0.89 5.06
9.	MAC alle, BM + FC alle, MRV + MWO	18.4	6.5 36.6	490	166 980	7.27	2.57 14.38
10.	MAC alle, BM + FC alle, MRVpess + MWO	17.7	6.7 32.6	467	171 858	6.90	2.72 12.67
11.	MAC alle, BM + FC alle MRVopt + MWO	15.2	6.9 29.8	392	168 780	5.74	2.59 11.26

Abbildung 3: Performanz unterschiedlicher Varianten des *branch&bound*.

tet, danach die Konsistenz mit der höchsten Hierarchieebene allein, danach mit den zwei höchsten Hierarchieebenen zusammengenommen usw. Abbildung 2 zeigt, dass die optimistische bzw. pessimistische Variante der MRV-Heuristik zu sehr unterschiedlichen Reihenfolgen führen können.

Nachteil dieser Verfahren ist der Aufwand, der beim Sortieren der Variablen entsteht. Dieser Aufwand kann (in seltenen Fällen) den erreichten Nutzen übersteigen.

4 Empirische Evaluation

Der vorherige Abschnitt skizzierte die Effekte verschiedener Constrainttechniken zur Verbesserung des *branch&bound* Algorithmus, wobei die verschiedenen Verfahren teilweise aufeinander aufbauen. Durch Kombination dieser Techniken lässt sich eine Vielzahl von Suchalgorithmen unterschiedlicher Performanz generieren. Ziel empirischer Evaluation ist nun, Kriterien für die Auswahl der günstigsten Kombination verfügbarer Techniken zu ermitteln. Dazu können zum einen konkrete Standardprobleme wie z.B. das Zebra- oder des *n*-Damen-Problem verwendet werden. Darüberhinaus besteht die Möglichkeit, Probleme mittels eines Zufallsfaktors zu generieren. Letzteres Verfahren ermöglicht es, beliebig viele Repräsentanten von CSPs mit bestimmten Merkmalen zu generieren. Dadurch wird es möglich, unabhängig von der konkreten Modellierung spezieller Beispielprobleme Aussagen über die Performanz von Constraintverfahren zu belegen.

Eine Standardklassifikation von Problemen mit zweistelligen Constraints berücksichtigt (vgl. z.B. [2,5,11]):

- die Anzahl der Variablen und die Größe ihrer Domänen als Maß der Komplexität des Problems,
- die Wahrscheinlichkeit p_1 der Generierung eines Constraints zwischen zwei Variablen (Dichte des Problems)
- den Quotienten p_2 der Mächtigkeit der Extension der Constraints zur Mächtigkeit des kartesischen Produkts der Domänen der beteiligten Variablen (Erfüllbarkeit der Constraints).

Diese Charakterisierung von CSPs ist zwar ausgesprochen grob, ermöglicht jedoch trotzdem einige wertvolle Beobachtungen.

Abbildung 3 zeigt die Ergebnisse einiger Beispielexperimente, in denen 10 Variablen mit jeweils 10 Werten vollständig durch Constraints vernetzt wurden, die gleichmäßig auf 6 Hierarchieebenen mit jeweils einem Gewicht von *1.0* verteilt wurden. Diese HCSPs wurden auf einer SUN SPARCstation Ultra II durch die ConPlan C++ Bibliothek bearbeitet. Diese Bibliothek stellt kombinierbare Verfahren zur Lösung von Con-

straintproblemen zur Verfügung und wurde am DFKI in Kaiserslautern sowohl für kommerzielle Anwendungen [12] als auch zur empirischen Evaluation von Constraintverfahren entwickelt. Die erste Spalte der Tabelle in Abbildung 3 bezeichnet den verwendeten Algorithmus, wobei *BM* für *backmarking* steht – die Zwischenspeicherung von Propagierungsergebnissen. *FC* wurde grundsätzlich für harte und weiche Constraints durchgeführt, *MAC* entweder nur für harte oder für harte und weiche Constraints. »MAC alle« steht für die Berechnung unscharfer kantenkonsistenter Domänen[4] (siehe Abschnitt 3.2). Die Ergebnisse der Constraintpropagierung wurden bei der Bestimmung der Reihenfolge berücksichtigt, in der die Werte ihren Variablen zugewiesen wurden. Die Reihenfolge, in der die Variablen belegt wurden, wurde durch *MRV* in der klassischen, optimistischen bzw. pessimistischen Variante bestimmt. Konnten Variablen nicht aufgrund dieses Kriteriums unterschieden werden, so wurde *MWO* angewandt. Angegeben werden für 20 gleichartige Probleme die durchschnittliche, die minimale und die maximale aufgetretene Rechenzeit, Anzahl der durchgeführten Zuweisungen (Knoten des Suchbaums) und die Anzahl der durchgeführten Constrainttests (von jeweils einem Tupel). Dabei handelt es sich um die gebräuchlichen Maße für die Performanz eines CSP-Algorithmus.

Wir fassen die wesentlichen Beobachtungen aus unseren Experimenten zusammen:

- Die Bedeutung der dynamischen Variablenanordnung im Suchbaum wird durch den Vergleich der Ergebnisse der Experimente 1 und 2 mit den restlichen Tests deutlich. Zur Berechnung einer optimalen Lösung muss auf jeden Fall der gesamte Suchbaum erkundet werden, so dass die MRV-Heuristik, die kleine Domänen oben im Suchbaum plaziert, besonders wirksam ist.
- Desweiteren zeigt sich, dass unter Verwendung von *FC* bzw. *MAC* nur harter Constraints *MRVpess* die bessere Heuristik ist. Offensichtlich führt *MRVopt* in diesen Fällen belegt durch den Vergleich mit der normalen *MRV*-Heuristik (Experimente 5 und 3 bzw. 8 und 6) zur Fehlinformation der Suche, wenn nicht genügend Propagierungsergebnisse vorliegen – oder, anders ausgedrückt, der Grad an prospektiver Constraintverarbeitung zu gering ist.
  Je größer die Konsistenz zwischen den unbelegten Variablen und den bislang während der Suche durchgeführten Zuweisungen ist, desto besser ist die optimistische Variante der *MRV*-Heuristik. Werden kantenkonsistente Kompatibilitäten berechnet, so ist zumeist *MRVopt* anzuraten. Unter Verwendung von *MRVopt* sind die zuerst aufgefundenen Lösungen im Durchschnitt besser als unter Verwendung der *MRVpess*-Heuristik (nicht in der Tabelle dargestellt). Durch den daraus resultierenden *bound* kann der Suchraum wesentlich besser eingeschränkt werden.
- Unscharfe, kantenkonsistente Domänen (»MAC alle«) verursachen eine im Vergleich mit dem im dargestellten Experiment effektivsten Mix von Verfahren eine wesentlich höhere Anzahl von Constrainttests (man vergleiche z.B. die Experimente 7 und 11). Jedoch ist bei Verwendung der *MRVopt*-Heuristik die Anzahl der durchgeführten Zuweisungen (Knoten im Suchbaum) am besten. Dies ist ein Zeichen dafür, dass kantenkonsistente Kompatibilitäten zwar einen vorteilhaften Effekt haben, jedoch im allgemeinen zuviel Aufwand bei der Propagierung verursachen. Für spezielle Extensionen von Constraints sind allerdings effizientere Propagierungsverfahren bekannt.[5] Dominieren in einer Anwendung Constraints mit solchen Extensionen, so verändert sich das durchschnittliche Verhältnis zwischen den während der Suche durchgeführten Tests und den durchgeführten Zuweisungen. Ergebnis: Für spezielle Probleme kann die Berechnung kantenkonsistenter Kompatibilitäten den Suchaufwand verkleinern.

5 Fazit

Constraints erfreuen sich seit vielen Jahren wachsenden Interesses. Die Problembeschreibung durch Relationen zwischen Variablen mit explizit repräsentierten Domänen ist nicht nur adäquat zur Beschreibung vielfältiger kombinatorischer Probleme; sie bietet auch Vorteile für ihre effiziente Bearbeitung. Die These dieses Beitrags ist, dass für den Bereich der weichen Constraints genau dasselbe gilt: Zum einen ergeben sich Erleichterungen bei der kombinierten Repräsentation von auf den ersten Blick unterschiedlichen Aspekten einer Anwendung wie Bedingungsgewichtung, Kostenoptimierung und vermutete Lösungseigenschaften. Zum anderen ermöglicht die Repräsentation auch dieser Problemaspekte durch Constraints die Anwendung effektiver Verfahren.

[4] Allerdings wurde mit den Ergebnissen des *MAC* mit weichen Constraints kein *backmarking* durchgeführt.

[5] Einfachstes Beispiel sind funktionale Abhängigkeiten, zu deren Propagierung nur die Urbilder der repräsentierten Funktion und nicht die Elemente der gesamten Extension aufgezählt werden müssen [8].

Üblicherweise wird, um Optimierungsaspekte in Constraintproblemen zu berücksichtigen, das klassische Constraintproblem um eine Zielfunktion erweitert, die Belegungen *aller* Variablen eine Bewertung zuordnet [15]. Diese Form der Repäsentation verhindert die Anwendung von Constraintverfahren auf das Optimierungskriterium, die zu einer wesentlich effektiveren Bearbeitung des Optimierungsproblems führen.

Der wesentliche Beitrag dieses Artikels besteht in einer praxisorientierten Möglichkeit, verschiedene Varianten weicher *crisp* Constraints durch ein einheitliches Schema zu beschreiben: *Hierarchical constraint satisfaction problems.* Den ersten Teil der am Anfang dieses Abschnitts formulierten These belegt unsere Anwendungserfahrung [12]: Constrainthierarchien sind nützlich bei der Strukturierung weicher Kriterien für die Beurteilung eine Lösung. Der zweite Teil der These wird durch die in diesem Beitrag dargestellten Constraintverfahren belegt.

Die vorgestellte integrierte Sichtweise eröffnet insbesondere die Möglichkeit, ebenfalls die auf einzelne Constraintformalismen anwendbaren Verfahren zu integrieren. Durch Kombination einzelner Techniken lässt sich eine Vielzahl an Suchalgorithmen zusammenstellen. Dabei sind, wie am Beispiel prospektiver Constraintverfahren und dynamischer Variablenanordnungen gezeigt, vielfältige Effekte zu berücksichtigen, die sich aus der Kombination unterschiedlicher Verfahren ergeben. Da somit eine Vielzahl von Algorithmen mit unterschiedlichen Eigenschaften möglich ist, kommt der Auswahl des besten Verfahrens für eine konkret gegebene Problemklasse besondere Bedeutung zu. Leider ist diese Auswahl immer noch eher eine Frage der Erfahrung als einer gesicherten Theorie.

Die notwendige Erfahrung wird zum einen durch praktische Anwendung und zum anderen durch empirische Untersuchungen gewonnen. Um eine Vergleichbarkeit der Ergebnisse empirischer Untersuchungen zu erzielen, wurde die gängige Systematik für die Generierung von Zufallsproblemen dargestellt. Die exemplarische Untersuchung der vorgestellten Verfahren verdeutlicht die Schwierigkeiten, die allgemein bei der Interpretation von Daten empirischer Tests bestehen.

Literatur

1. S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie. *Semiring-based CSPs and valued CSPs: Basic properties and comparisons*. M. Jampel (Hrsg.) Over-Constrained Systems. LNCS 1106. Springer-Verlag, 1996.
2. F. Bacchus und P. van Run. *Dynamic variable ordering in CSPs*. Aus *CP-95: Proceedings of the 1st International Conference on Principles and Practice of Constraint Programming*. Seiten 258-275. LNCS 976, Springer-Verlag, 1995.
3. A. Borning, B. Freeman-Benson, M. Wilson. *Constraint hierarchies*. Lisp and Symbolic Computation, 5:233-270, 1992.
4. D. Dubois, H. Fargier und H. Prade. *Propagation and satisfaction of flexible constraints.* Aus: R. Yager und L.A. Zadeh (Hrsg.): *Fuzzy Sets, Neural Networks and Soft Computing,* Seiten 166-187, New York, 1993. Van Nostrand Reinhold.
5. E. Freuder und R. Wallace. *Partial constraint satisfaction*. Artificial Intelligence, 58:21-70, 1992.
6. H. W. Guesgen. *A formal framework for weak constraint satisfaction based on fuzzy sets*. Technical Report TR-94-026, Berkeley International Computer Science Institute (ICSI), Juni 1994.
7. R. Haralick und G. Elliot. *Increasing tree search efficiency for constraint satisfaction problems*. Artificial Intelligence, 14:263-313, 1980.
8. P. Van Hentenryck, Y. Deville und Choh-Man Teng. A generic arc-consistency algorithm and its specializations. Artificial Intelligence, 57: 291-321, 1992.
9. A. Mackworth. *Consistency in networks of relations*. Artificial Intelligence, 8(1):99-118, 1977.
10. M. Meyer: Finite Domain Constraints: Declarativity meets Efficiency, Theory meets Application. DISKI 79. infix Verlag, Sankt-Augustin. 1995.
11. H. Meyer auf'm Hofe. *Partial satisfaction of constraint hierarchies in reactive and interactive configuration*. Aus: Walter Hower und Zsófia Ruttkay (Hrsg.) Non-Standard Constraint Processing, Working Notes of the ECAI'96 Workshop W27, Budapest, Ungarn, 1996.
12. H. Meyer auf'm Hofe, E. Tolzmann. *ConPlan/SIEDAplan: Personaleinsatzplanung als Constraintproblem,* KI - Künstliche Intelligenz, 1/1997.
13. E. M. Reingold, J. Nievergelt und N. Deo. *Combinatorical algorithms: theory and practice*. Prentice Hall, Englewood Cliffs, New Jersey, 1977.
14. P. Snow und E. Freuder. *Improved relaxation and search methods for approximate constraint satisfaction with a maximin criterion.* Aus: Proc. of the 8th biennal conf. of the canadian society for comput. studies of intelligence, Seiten 227-230, Mai 1990.
15. E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press Limited, London. 1993.
16. R. Wallace und E. Freuder. *Conjunctive width heuristics for maximal constraint satisfaction*. Aus: AAAI-93: Proceedings of the 11th National Conference on Artificial Intelligence, Washington DC, Seiten 762-768. AAAI Press / MIT Press, 1993.