Lokale Suchverfahren für Constraint Satisfaction Probleme: Die *min conflicts* Heuristik und Tabu Search

Thomas Stützle
Technische Hochschule Darmstadt
Fachbereich Informatik, Fachgebiet Intellektik
Alexanderstr. 10, D-64283 Darmstadt, Germany
tom@intellektik.informatik.th-darmstadt.de

Zusammenfassung

Constraint Satisfaction Probleme bieten einen allgemeinen Rahmen für die Behandlung vieler wichtiger Problemstellungen. Zur Lösung für Constraint Satisfaction Probleme können auch lokale Suchverfahren mit großem Erfolg eingesetzt werden. In diesem Artikel präsentieren wir eine empirische Untersuchung eines solchen lokalen Suchverfahren, der *min conflicts* Heuristik und einiger ihrer Varianten, für Constraint Satisfaction Probleme. Insbesondere zeigen wir, wie die *min conflicts* Heuristik mit Tabu Search verbunden werden kann und damit die zugrundelegende *min conflicts* Heuristik deutlich verbessert werden kann.

1 Einführung

Probleme mit Nebenbedingungen herrschen in vielen Anwendungen vor. Ein Zweig der Intellektik [1] beschäftigt sich seit langem mit einer generellen Klasse von Problemen, den sogenannten *Constraint Satisfaction Problemen* (CSPs), mit denen allgemein Probleme mit Nebenbedingungen modelliert werden können [15]. Unter den Problemstellungen, die als CSP formuliert werden können, befinden sich auch viele wichtige Problemstellungen wie Graphfärbeprobleme, rämliches und zeitliches Schließen [11, 4] und Scheduling [23, 24]. Probleme mit Nebenbedingungen kann man als Suchprobleme auffassen, bei denen eine Konfiguration aus der Menge aller möglichen herausgesucht werden muß, so daß die Nebenbedingungen (Constraints) erfüllt sind und möglicherweise eine gegebene Zielfunktion optimiert wird. Im allgemeinen sind solche Probleme $\mathcal{NP}$-vollständig[1], sie können also nach heutigen Kenntnissen nicht mit einem deterministischen Algorithmus in polynomieller Zeit gelöst werden.

Allgemein unterscheidet man bei den Möglichkeiten CSP zu lösen [5] zwischen vollständigen (oder globalen) Suchverfahren, die garantiert eine Lösung finden sofern eine existiert, und lokalen Suchverfahren. Bei lokalen Suchverfahren wird versucht, beginnend mit einer vollständigen Variablenbelegung durch eine Folge von kleinen Veränderungen das Problem zu lösen. Lokale Suchverfahren erfreuen sich insbesondere in den letzten Jahren im Bereich der CSPs [17, 3] und auch für aussagenlogische Erfüllbarkeitsprobleme (SAT) [27, 26, 13] zunehmender Beliebtheit. Ein Grund hierfür ist sicherlich, daß mit den lokalen Suchalgorithmen sehr große Probleme gelöst werden konnten, die jenseits der Möglichkeiten von vollständigen Suchalgorithmen liegen. In diesem Artikel beschäftigen wir uns mit lokalen Suchverfahren für CSP, speziell mit

[1] So ist beispielsweise das Graphfärbeproblem ein Spezialfall der CSP.

der sogenannten *min conflicts* Heuristik [17]. Wir führen eine empirische Untersuchung der *min conflicts* Heuristik und einiger Verbesserungen gegenüber der ursprünglichen Form der Heuristik durch. Insbesondere gehen wir darauf ein, wie die *min conflicts* Heuristik durch Tabu Search, einer Metaheuristik die ursprünglich aus der KI stammt, inzwischen aber vor allem im Operations Research mit sehr großem Erfolg angewandt wird [9, 30], verbunden und signifikant verbessert werden kann. Wir werden zeigen, daß die Kombination der *min conflicts* Heuristik mit Tabu Search auch im Vergleich zu anderen Erweiterungen der *min conflicts* Heuristik bessere Ergebnisse erzielt. Hybridisierungen zwischen den Varianten ergeben nochmals weitere Möglichkeiten der Verbesserung. Die Varianten der *min conflicts* Heuristik wurden anhand von zufällig erzeugten binären CSP, Maximal Constraint Satisfaction Problemen[2] (MAX-CSP) und Graphfärbeproblemen getestet.

Der Artikel ist wiefolgt aufgebaut. In Kapitel 2 gehen wir kurz auf Constraint Satisfaction Probleme ein, stellen einige Grundlagen der lokalen Suche dar, beschreiben kurz die wesentlichen Charakteristika von Tabu Search und zeigen, wie wir Tabu Search mit der *min conflicts* Heuristik verbunden haben. Anschließend beschreiben wir in Kapitel 3 die verwendeten Testprobleme und präsentieren in Kapitel 4 die experimentellen Ergebnisse mit den verschiedenen Algorithmen. Abschließend gehen wir kurz auf andere Arbeiten ein und geben einen kurzen Ausblick über künftige Vorhaben.

2 Constraint Satisfaction und lokale Suche

2.1 Constraint Satisfaction Probleme

Ein Constraint Satisfaction Problem besteht aus einer Menge von Variablen $\mathcal{X} = \{X_1, X_2, \ldots X_n\}$, einem diskreten, endlichen Wertebereich D_i für jede Variable X_i und einer Menge von Constraints $\mathcal{C}$. In diesem Artikel beschränken wir uns auf binäre CSP, d.h. ein Constraint betrifft maximal 2 Variablen. Ein Constraint $C(X_i, X_j) \subset D_i \times D_j$, schränkt die Menge der erlaubten Wertepaare für zwei Variablen ein. Für die hier behandelten Probleme gelte, daß alle Wertebereiche der Variablen identisch sind, also $\mathcal{D} = D_1 = D_2 = \ldots = D_n = \{1, 2, \ldots, k\}$.

Eine Zuweisung eines eindeutigen Wertes $d_i \in D_i$ an eine Variable X_i nennen wir Variableninstantiierung. Gilt $((X_i, d_i), (X_j, d_j)) \notin C(X_i, X_j)$, nennen wir den Constraint $C(X_i, X_j)$ verletzt, die Werte d_i und d_j der Variablen der verletzten Constraint sind in *Konflikt*. Als Konfliktmenge $K(\mathcal{X})$ bezeichnen wir die Menge der Variablen, die in eine Constraintverletzung involviert sind, also $K(\mathcal{X}) = \{X_i \mid X_i \text{ ist in eine Constraintverletzung involviert}\}$. Die Aufgabe in einem CSP ist, eine Instantiierung für alle Variablen zu finden, so daß alle Constraints erfüllt werden. Eine Variableninstantiierung für alle Variablen bezeichnen wir kurz als *Instantiierung* $s = ((X_1, d_1), (X_2, d_2) \ldots (X_n, d_n))$. Mit $s(X_i) = d_i$ bezeichnen wir den Wert, den die Variable X_i in s annimmt.

2.2 Lokale Suchverfahren für CSP – die *min conflicts* Heuristik

Ein lokales Suchverfahren (oft auch Nachbarschaftssuchverfahren genannt) startet typischerweise mit einer Instantiierung s und ersetzt diese schrittweise durch möglicherweise bessere Instantiierungen. Für ein CSP ist der Suchraum $\mathcal{S}$ ist durch die Menge aller möglichen Instantiierungen

[2] Einer Optimierungsvariante der CSP bei der man möglichst viele Constraints erfüllen möchte.

gegeben, in unserem Fall also $\mathcal{S} = \mathcal{D} \times \mathcal{D} \times \ldots \times \mathcal{D}$. Um eine Instantiierung s zu bewerten, benötigt man eine Zielfunktion $f(s)$. Im Falle der CSP kann man als Zielfunktion die Anzahl der verletzten Constraints wählen. Das Ziel ist dann, die Anzahl der verletzten Constraints zu minimieren. Ein CSP ist genau dann lösbar, falls $\exists s : f(s) = 0$.

Die möglichen Veränderungen, die je Schritt vorgenommen werden dürfen, werden durch eine Nachbarschaftsbeziehung $N : \mathcal{S} \mapsto 2^{\mathcal{S}}$ definiert. Die Menge der Nachbarn einer Instantiierung s bezeichnen wir mit $N(s)$. Die Entfernung $d(s, s')$ zweier Instantiierungen s und s' sei die Anzahl der unterschiedlichen Variablenwerte, also $d(s, s') = |\{X_i \mid s(X_i) \neq s'(X_i)\}|$. Sehr wichtig für lokale Suchverfahren ist, wie die Nachbarschaftsbeziehung zwischen Lösungen genau definiert ist. Die einfachste Nachbarschaftsbeziehung ist, daß sich zwei Instantiierungen s und s' in maximal einer Variableninstantiierung unterscheiden, also $d(s, s') \leq 1$. Komplexere Nachbarschaftsbeziehungen können geformt werden, wenn benachbarte Instantiierungen sich durch maximal k Variablenbelegungen unterscheiden, also $d(s, s') \leq k$. Bei der Größe der gewählten Nachbarschaft besteht dabei ein tradeoff zwischen der erreichbaren Lösungsgüte und dem Aufwand, die Nachbarschaft einer Instanziierung zu untersuchen. Ein Problem das allgemein bei lokalen Suchverfahren auftritt, ist das Vorhandensein von lokalen Minima im Suchraum $\mathcal{S}$. Ein lokales Minimum ist eine Instantiierung, so daß $\forall s' \in N(s) : f(s') \geq f(s)$.[3] Eine Möglichkeit, aus lokalen Minima zu entkommen bietet der Neustart der lokalen Suche von einer neuen, zufällig erzeugten Anfangsinstantiierung. Eine andere Möglichkeit lokale Minima zu umgehen ist, auch verschlechternde Nachbarschaftszüge zu akzeptieren. Verfahren bei denen auch schlechtere Instantiierungen angenommen werden können sind z.B. Tabu Search [9], Simulated Annealing [14] oder Genetische Algorithmen [10].

Die *min conflicts* Heuristik [17] ist ein lokales Suchverfahren für CSP, bei dem $s' \in N(s)$, falls $d(s, s') \leq 1$.[4] Die *min conflicts* Heuristik läuft im wesentlichen in zwei Schritten ab. Im ersten Schritt wird entschieden welche Variable ihren Wert verändern darf, im zweiten Schritt wird dann für diese Variable ein neuer Wert bestimmt. Die Variable wird zufällig aus der Konfliktmenge $K(\mathcal{X})$ gewählt. Dieser Variablen wird dann ein Wert aus ihrem Wertebereich zugewiesen, für den die Anzahl der Konflikte minimal ist.[5] Wird das Minimum von mehreren Werten angenommen, so wird einer von diesen zufällig gewählt. Die Wahl einer Variablen und ihres zugeörigen Wertes bezeichnen wir im folgenden als eine *Iteration* (des Algorithmus) oder auch als *Nachbarschaftszug*. Die Anfangsinstantiierung für die Heuristik kann zufällig gewählt oder aufgrund einer problemspezifischen Konstruktionsheuristik bestimmt werden. In Abbildung 1 ist die *min conflicts* Heuristik nochmals in Pseudocode dargestellt.

Ein Nachteil bei der *min conflicts* Heuristik ist, daß sie leicht in lokalen Minima stecken bleiben kann. Möglichkeiten lokale Minima zu überwinden bietet einerseits ein Neustart des Algorithmus. Dazu gibt ein Parameter MAX-CHANGES die maximal durchzuführende Anzahl an Nachbarschaftszügen an, bevor die Heuristik wieder mit einer neuen zufälligen Instantiierung gestartet wird. Eine andere Möglichkeit ist es einen *random walk* durchzuführen [32]. Bei dieser Strategie – sie wurde ursprünglich in ähnlicher Form bei GSAT verwendet [25] – wird für die gewählte Variable mit einer bestimmten Wahrscheinlichkeit p_{walk} ein Wert zufällig gewählt, andernfalls die übliche *greedy* Strategie der *min conflicts* Heuristik verfolgt. Durch den *random walk* wird die Auswahl der nächsten Instantiierung beeinflußt. Während bei der *min conflicts* Heuristik nur Nachbarschaftszüge gemacht werden können, die den Zielfunktionswert

[3] Für ein lokales Minimum im strengen Sinne gilt $\forall s' \in N(s) : f(s') > f(s)$.

[4] Bei der *min conflicts* Heuristik kann eine Variable auch ihren bisherigen Wert beibehalten.

[5] Die Anzahl der Konflikte für ein Variablen-Werte Paar (X_i, d_i) ist die Anzahl der verletzten Constraints, in die (X_i, d_i) involviert ist.

procedure *min-conflicts* ($\mathcal{X}$, $\mathcal{D}$, $\mathcal{C}$, s)
$\mathcal{X} \leftarrow$ *Variablen*, $\mathcal{D} \leftarrow$ *Wertebereich*, $\mathcal{C} \leftarrow$ *Constraints*,
$s \leftarrow$ *Instantiierung*
begin
  $s \leftarrow$ Anfangsinstantiierung der Variablen;
  repeat
    Wähle ein Variablen-Werte Paar (X_i, d_i), so daß $X_i \in K(\mathcal{X})$
    $X_i \leftarrow d_j$, so daß $d_j \in \mathcal{D}$ die Anzahl der Konflikte minimiert
    $s \leftarrow (s - (X_i, d_i)) \cup (X_i, d_j)$
  until alle Constraints erfüllt **oder** Abbruchkriterium erfüllt
end

Abbildung 1: Pseudocode der *min conflicts* Heuristik

nicht erhöhen, wird mit dem *random walk* in Kauf genommen, daß sich der Zielfunktionswert verschlechtern kann.

2.3 Tabu Search

Tabu Search[6] [8, 12] ist eine *Metaheuristik* [19] zur "Leitung" eines lokalen Suchalgorithmus.[7] Tabu Search hat die Eigenschaft, daß lokale Minima wieder verlassen werden können. Es handelt sich aber im Gegensatz zu *random walk* oder Simulated Annealing um ein Verfahren, bei dem weniger die probabilistischen Elemente betont werden, als vielmehr die Suche stärker determinisisch geleitet wird. Bei Tabu Search wird in jeder Iteration der bestmögliche Nachbar gewählt, auch falls dieser den Zielfunktionswert verschlechtert. Dies kann zu Zyklen führen, da man wieder zu einer vorher besuchten Lösung zurückkehren kann. Um diese Zyklen zu vermeiden, werden Attribute früherer Instantiierungen in einer *Tabuliste* für eine bestimmte Anzahl t an Iterationen gespeichert. Eine Nachbarlösung wird als *verboten* angesehen, falls diese ein Attribut, das momentan in der Tabuliste steht, enthält. Dabei kann es aber durchaus geschehen, daß sehr gute Nachbarlösungen, die noch nicht besucht wurden, durch die Attribute in der Tabuliste als verboten deklariert werden. Um den Tabustatus solcher Lösungen zu überschreiben verwendt man die sogenannten *aspiration level* Bedingungen. Diese wählt man meist so, daß man eine Nachbarlösung, die den bisher besten Zielfunktionswert verbessert, trotzdem akzeptiert.

Tabu Search wird meist als zusätzliche Metaheuristik verwendet, die eine zugrundeliegende lokale Suchheuristik leiten soll. Für CSP verwenden wir deshalb die *min conflicts* Heuristik als lokale Suchmethode. Eine wesentliche Einflußmöglichkeit bei der Verbindung von Tabu Search mit der *min conflicts* Heuristik bietet die Wahl der als Tabu zu erklärenden Attribute. In einer Variante haben wir Variablen-Werte Paare als Tabu erklärt. Somit wird direkt die Auswahl der Variablenbelegungen beeinflußen. Wird z.B. durch die *min conflicts* Heuristik das Variable-Werte Paar (X_i, d_i) verändert zu (X_i, d_j), so wird (X_i, d_i) für die nächsten t Iterationen Tabu erklärt. Für diesen Fall haben wir auch *aspiration level* Bedingungen verwendet. In einer anderen Variante haben wir eine ganze Variablen als Tabuattribut gewählt. Wählt man eine Variable als Tabuattribut, so bedeutet das, daß diese für eine feste Anzahl von t Schritten

[6] Eine andere Bezeichnung für Tabu Search ist *steepest descent – mildest ascent*.

[7] Eine *Metaheuristik* ist allgemein eine iterative Methode, die eine untergeordnete Heuristik *führt*, indem verschiedene Konzepte der Exploration verschiedener Lösungen und der Intensivierung der Suche intelligent kombiniert werden.

ihren Wert nicht ändern darf. Damit wird insbesondere die Auswahl der Variablen, die als nächste ihren Wert verändern darf, beeinflußt. Es kann auch vorkommen, daß eine Variable X_i ausgewählt werden muß, für die $X_i \notin K(\mathcal{X})$.

3 Verwendete Probleme

3.1 Zufällig erzeugte CSP

Eine durchaus übliche Methode ist, verschiedene Algorithmen an zufällig erzeugten Problemen zu testen [27, 28], mit denen insbesondere sehr schwere Testprobleme erzeugt werden können. Wir haben hier ebenfalls zufällig erzeugte binäre CSP, die gemäß dem üblichen $< n, k, p, q >$-Modell erzeugt wurden [22, 28], als Testprobleme verwendet. Bei diesem Modell für die Erzeugung zufälliger CSP, ist n die Anzahl der Variablen, k die Größe des für alle Variablen identischen Wertebereichs, p die Wahrscheinlichkeit, daß zwischen einem Paar von Variablen eine Constraint besteht und q die bedingte Wahrscheinlichkeit, daß ein Paar von Werten zulässig ist, falls zwischen diesen Variablen eine Constraint besteht.[8] Diese zufällig erzeugten CSP unterliegen bei einer kritischen Grenze der Parameter für vollständige Suchalgorithmen einer Phasentransition [22, 29, 28]. Hält man die Werte für n, k, p fest, so steigt der mittlere Suchaufwand stark bis zu einem Spitzenwert bei einem bestimmten Wert q_{krit} und sinkt dann wieder schnell ab. Allgemein wird angenommen, daß bei zufällig erzeugten CSP die kritischen Parameterwerte für die Phasentransition über die erwartete Anzahl der Lösungen mit verbunden sind [28]. Es zeigt sich, daß auch für lokale Suchalgorithmen im Mittel Probleme aus dem *harten* Bereich deutlich schwerer sind. Für lokale Suchverfahren sind sogar ähnliche Effekte eines Phasenübergangs nachzuweisen [2], falls man sich auf lösbare Instanzen beschränkt.

Da lokale Suchmethoden nicht in der Lage sind die Unerfüllbarkeit eines Problems zu zeigen, haben wir für unsere Untersuchung nur lösbare CSP verwendet. Bei den Problemen handelt es sich durchweg um Probleme mit $n = 20$ Variablen und jeweils $k = 10$ möglichen Werten. Wir haben jeweils 100 Probleme mit Werten für $p \in \{0,1 \quad 0,3 \quad 0,5 \quad 0,7 \quad 0,9\}$ und verschiedenen Werten für q erzeugt, so daß q jeweils einige Werte um q_{krit} annimmt. q wurde in Schritten von $0,01$ variiert. Die für die lokale Suche verwendeten CSP wurden mittels eines vollständigen Algorithmus aus den nicht lösbaren Problemen herausgesucht, da als lösbar erzeugte Probleme viel leichter zu lösen sind. Dies beschränkt auch den erfüllbaren Bereich, aus dem die Probleme genommen werden können, da für zu niedrige Werte für q nur ein verschwindend geringer Teil der Probleme erfüllbar ist und der entstehende Aufwand diese herauszufiltern enorm wird.

3.2 Maximal-CSP

Für einige CSP, die viele Constraints besitzen und diese Constraints einen großen Teil der mögliche Wertepaare ausschließen, kann es passieren, daß keine Instantiierung gefunden werden kann, die alle Constraints erfüllt. In einem solchen Fall mag es immer noch sinnvoll sein, die wichtigsten Constraints zu erüllen. Ein Beispiel eines Problems, bei dem jeder Constraint als gleich wichtig angesehen wird, ist das Maximal Constraint Satisfaction Problem (MAX-CSP)[9], bei dem eine möglichst große Anzahl an Constraints erfüllt werden soll [6]. Prinzipiell handelt

[8] In der Literatur ist es auch üblich $p_2 = 1 - q$ zu verwenden, wobei p_2 die Wahrscheinlichkeit angibt, mit der ein Paar von Werten inkonsistent ist.

[9] Das Analogon bei SAT Problemen ist das MAX-SAT Problem.

es sich beim MAX-CSP um ein Optimierungsproblem. Lokale Suchverfahren können verwendet werden, um in kurzer Zeit optimale oder fast-optimale Lösungen zu finden. Außerdem sind auch globale Suchverfahren wie Branch and Bound Algorithmen darauf angewiesen, anfangs möglichst niedrige obere Schranke für den Zielfunktionswert zu haben [6, 31]. Diese oberen Schranken können wiederum durch lokale Suchverfahren bestimmt werden. Als Testprobleme für die Varianten der *min conflicts* Heuristik haben wir binäre CSP unterschiedlicher Größe erzeugt, mit Werten für q aus dem Bereich, in dem praktisch alle Probleme unerfüllbar sind.

3.3 Graphfärbeprobleme

Das Graphfärbeproblem ist ein binäres CSP, bei dem die Knoten des Graphen den Variablen des CSP entsprechen, der Wertebereich die zur Verfügung stehenden k Farben umfaßt, und die Constraints besagen, daß die Farbe benachbarter Knoten verschieden sein muß. Mit der *min conflicts* Heuristik kann insbesondere die Entscheidungsvariante des Graphfärbeproblems gelöst werden. Die Optimierungsvariante – gesucht ist die minimal benötigte Anzahl an Farben, um einen Graphen zu färben – kann durch eine Folge von CSP mit fallendem k gelöst werden. Überschreitet man eine maximal zulässige Anzahl an Iterationen, so gilt des Problem als nicht erfüllbar und das kleinste k, für das der Graph gefärbt werden konnte, wird als Lösung ausgegeben. Die hier verwendeten Graphfärbeprobleme sind aus dem DIMACS Wettbewerb[10] und entsprechen dieser Optimierungsvariante. Wir haben einige Instanzen für Werte von k nahe der optimalen Lösung getestet.

4 Experimentelle Ergebnisse

Wir haben an fast allen Problemen auch die *min conflicts* Heuristik, die wir im folgenden kurz als `mch` bezeichnen, getestet. Die hier untersuchten Varianten der `mch` erzielten durchweg signifikant bessere Ergebnisse, so daß wir uns meist auf einen Vergleich der verschiedenen Varianten beschränken. Es wurden die folgenden Varianten untersucht. Einmal die Erweiterung der *min conflicts* Heuristik mit *random walk*, beschrieben in Kapitel 2.2, die wir mit `wmch` bezeichnen. Eine andere Möglichkeit ist, zufällig ein verletztes Constraint auszuwählen und für eine in diesen Constraint involvierte Variable einen Wert zu wählen, so daß dieser Constraint möglichst nicht mehr verletzt ist. Diese Variante wird als `wmchf` bezeichnet. Die Verbindung der *min conflicts* Heuristik mit Tabu Search, bei der Variablen-Werte Paare als Tabu erklärt werden bezeichnen wir mit `tmch`. Wird als Tabu-Attribut eine ganze Variable gewählt, zeigte sich keine eindeutigen Verbesserung gegenüber der *min conflicts* Heuristik. Jedoch brachte eine Kombination mit der *random walk* Strategie deutliche Verbesserungen gegenüber `wmch`. Bei diesem Algorithmus wird die Auswahl der Variable durch die Tabuliste beeinflußt, die Auswahl des Variablenwertes läuft gemäß der Version `wmch`. Dieser hybride Algorithmus wird als `tvwmch` bezeichnet. Eine Kombination von `tmch` und `wmch`, hier als `tw-mch` bezeichnet, zeigte dagegen keine Verbesserung gegenüber dem reinen `tmch`, wohl aber gegenüber `wmch`. Als Maß für den Lösungsaufwand haben wir die Anzahl der Iterationen gewählt, die durchgeführt werden müssen, um eine Probleminstanz zu lösen. Da für die einzelnen Probleminstanzen der Lösungsaufwand je Anwendung des Algorithmus stark variieren kann, haben wir jede Probleminstanz mehrmals gelöst und verwenden die durchschnittliche Anzahl der benötigten Iterationen als erwartungstreuen Schätzer für den eigentlichen Lösungsaufwand.

[10]Die Probleme sind verfügbar unter ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color

Für die lokalen Suchalgorithmen müssen oftmals verschiedene Parameterwerte bestimmt werden. Im wesentlichen sind das für die verschiedenen Algorithmen die Parameter MAX-CHANGES, die Walkwahrscheinlichkeit p_{walk} und die Tabulistenlänge t. Für die hybriden Algorithmen müssen alle drei Parameterwerte festgelegt werden. Hierbei zeigte sich für die *min conflicts* Heuristik eine sehr starke Abhängigkeit von MAX-CHANGES, siehe Abbildung 2 für zufällig erzeugte CSP mit $p = 0,5$. Die besten Ergebnisse werden mit MAX-CHANGES zwischen 100 und 250 erzielt. Wird MAX-CHANGES zu hoch gewählt, ist der mittlere Aufwand sehr hoch, wird MAX-CHANGES zu niedrig gewählt, steigt er ebenfalls wieder an. In den folgenden Abbildungen geben wir immer nur die Resultate für die verschiedenen Varianten an, da die `mch` einen deutlich höheren Lösungsaufwand benötigt, vergleiche Skala in Abbildung 2 und 3.

Für die Varianten der *min conflicts* Heuristik zeigte sich, daß der Einfluß von MAX-CHANGES nicht sehr stark war, also wurde für die Bestimmung der Parameterwerte hauptsächlich die Walkwahrscheinlichkeit p_{walk} und die Tabulistenlänge t untersucht.[11] Diese Beobachtung deckt sich durchaus mit denen bei ähnlichen Versuchen, siehe [7, 20]. Bei den Versuchen ergab sich überraschenderweise, daß für `tmch` die Länge der Tabuliste sehr niedrig gewählt werden muß, eine Tabulistenlänge von $t = 2$ oder $t = 3$ brachte für alle behandelten Probleme sehr gute Ergebnisse. Für die anderen Algorithmen zeigte sich, daß die besten Parameterwerte durchaus von den einzelnen behandelten Problemen abhängen können. Dies gilt insbesondere für die optimale Walkwahrscheinlichkeit. Die nachfolgend präsentierten Ergebnisse beruhen auf den jeweils besten Parametereinstellungen, die in den aufwendigen Vorläufen gefundenen wurden.

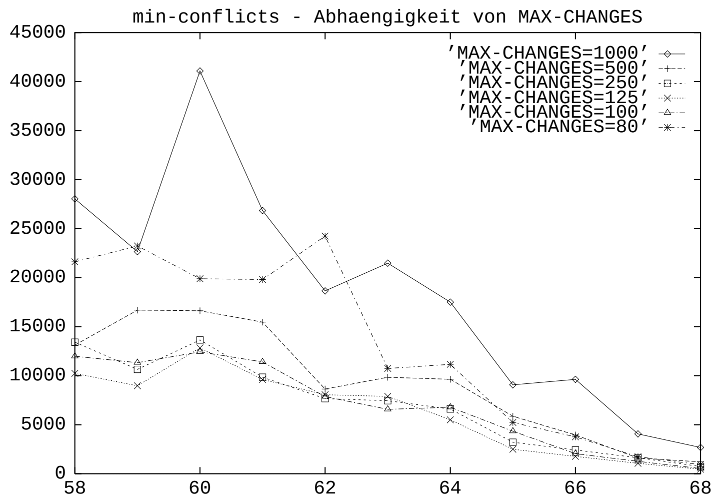

Abbildung 2: Abhängigkeit von MAX-CHANGES für die *min conflicts* Heuristik für zufällig erzeugte CSP mit $p = 0,5$. Die y-Achse gibt den Median des Lösungsaufwandes an, die x-Achse gibt $100 \cdot q$ an.

4.1 Zufällig erzeugte CSP

Bei den zufällig erzeugten CSP wurde jede Probleminstanz 25 mal gelöst und der Mittelwert der Anzahl der benötigten Iterationen als Schätzer für den Lösungsaufwand verwendet. Die Abbildungen 2 bis 5 beruhen jeweils auf den Median des so geschätzten Lösungsaufwand bei

[11] MAX-CHANGES muß aber deutlich höher als bei `mch` gewählt werden.

100 Problemen. Die Kurven für den mittleren Lösungsaufwand sehen in ihrem Verlauf und der relativen Lage den hier gezeigten Kurven für den Median sehr ähnlich, mit dem einzigen Unterschied, daß der mittlere Lösungsaufwand auf einem höheren Niveau liegt. Dies beruht darauf, daß der Mittelwert empfindlich auf Ausreißer reagiert und für einige wenige Probleme der Lösungsaufwand deutlich über dem Durchschnitt liegt. Mit dem Parameter p kann die erwartete Anzahl an Constraints gesteuert werden. Es zeigte sich, daß der Parameter p hauptsächlich die Höhe des Lösungsaufwand für Probleme aus dem Phasenübergangsbereich bestimmt. Dieser ist umso höher je höher auch p ist. Auf die relative Lage der Algorithmen, hatte er allerdings keinen wesentlichen Einfluß.[12]

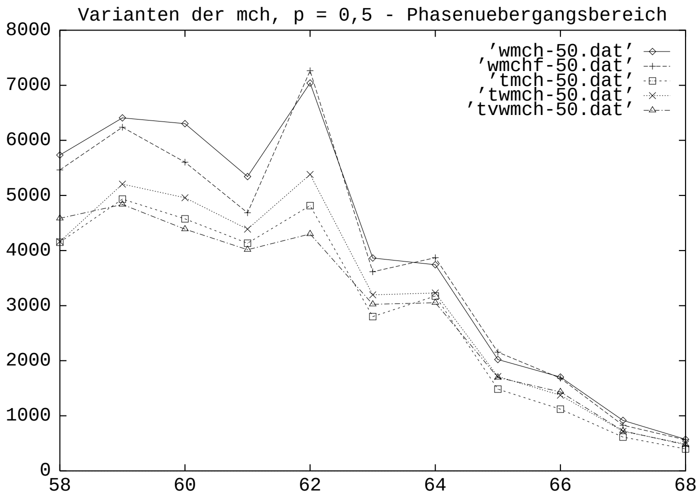

Abbildung 3: binäre CSP, $p = 0,5$. Die y-Achse gibt den Median des Lösungsaufwandes an, die x-Achse gibt $100 \cdot q$ an.

Alle hier dargestellten Varianten schneiden signifikant besser ab als die grundlegende **mch**, vergleiche Abbildung 2 mit Abbildung 3. Die Varianten mit Tabu Search bieten zusätzlich noch eine deutliche Verbesserung gegenüber den beiden Varianten mit *random walk*. Die Variante **tmch** schneidet durchweg sehr gut ab. Die zusätzliche Kombination mit *random walk* in der Version **twmch** ergibt allerdings für **tmch** keine Verbesserung mehr[13], wohl aber konnte **wmch** durch die zusätzliche Verwendung von Elementen aus Tabu Search verbessert werden. Bei den beiden Varianten der *min conflicts* Heuristik mit *random walk* ergibt sich ein sehr ähnlicher Aufwand, wobei die Variante **wmchf** für $p = 0,5$ und $p = 0,9$ leicht besser scheint.[14] Eine andere Möglichkeit der Verbesserung für die Varianten mit *random walk* bietet **tvwmch**, bei der die Tabuliste die Auswahl der Variablen beeinflußt.

Der mittlere Lösungsaufwand ist für die meisten Probleminstanzen ähnlich hoch. Jedoch für einige wenige ist der benötigte Lösungsaufwand meist deutlich höher, weshalb wir die verschiedenen Algorithmen anhand einiger besonders schwerer Instanzen getestet haben. Hierfür haben wir die 10 schwersten Probleme für $p = 0,5$ und $q = 0,62$ ($q_{krit} = 0,616$) herausgesucht und

[12] Das heißt, daß die erwartete Anzahl an Constraints keinen Einfluß auf die relative Bewertung der Algorithmen hatte.

[13] Dies zeigte sich auch bei der Bestimmung der Parameterwerte für **twmch**, man hätte eigentlich $p_{walk} = 0.0$ wählen können. Da dies aber dann der Variante **tmch** entsprechen würde haben wir hier für die Walkwahrscheinlichkeit den nächstkleinsten untersuchten Wert 0,025 gewählt.

[14] Für beide Varianten erwies sich ein eine Walkwahrscheinlichkeit von $p_{walk} = 0.1$ am besten.

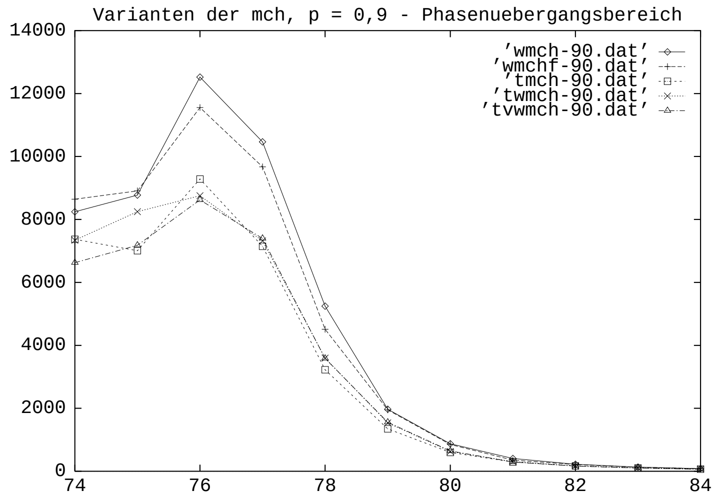

Abbildung 4: binäre CSP, $p = 0,9$.

Tabelle 1: Lösungsaufwand für die 10 schwersten Probleme für $p = 0,5$ und $q = 0,62$

	tmch	**twmch**	**wmch**	**wmchf**	**tvwmch**
Median	28355	31052	34885	32104	25466
Mittelwert	41384	41705	44339	44855	32641

jede Probleminstanz 100 mal gelöst, siehe Tabelle 1. Auch bei den härtesten Problemen zeigt sich wiederum die Überlegenheit der Varianten mit Elementen aus Tabu Search.

4.2 MAX-CSP

Für MAX-CSP haben wir je 25 Probleme mit $n \in \{20,\ 50,\ 100\}$ mit $p \in \{0,1\ 0,5\ 0,9\}$ erzeugt. Jede Probleminstanz wurde 10 mal gelöst und die mittlere Anzahl der verletzten Constraints berechnet. Die in Tabelle 2 angegebenen Ergebnisse entsprechen der durchschnittlichen Anzahl der verletzten Constraints bei den 25 zufällig erzeugten Problemen nach 10000 Iterationen für $n = 20$ und 25000 Iterationen für $n = 50$ und $n = 100$. Alle Varianten schneiden signifikant besser als die *min conflicts* Heuristik ab. Nur für die größten Probleme mit 100 Variablen scheinen die Varianten mit Tabu Search leicht besser, für kleinere Probleme sind die sich ergebenden Constraintverletzungen nahezu identisch. Für die relative Performanz der Algorithmen hat der Wert für p wiederum keinen merklichen Einfluß.

Der typische Verlauf der Anzahl der verletzten Constraints ist in Abbildung 6 wiedergegeben. Anfangs sinkt die Anzahl der verletzten Constraints bei allen Algorithmen sehr rasch ab. Die *min conflicts* Heuristik bleibt dann aber in lokalen Minimas hängen und die Anzahl der verletzten Constraints nimmt nach 10000 Iterationen nur noch unmerklich weiter ab. Anders allerdings bei den hier verwendeten Varianten der *min conflicts* Heuristik. Hier nimmt bei einer höheren Anzahl an Iterationen die Anzahl der verletzten Constraints immer noch weiter ab. Die Varianten **tmch** und **twmch** zeigen eine leicht schnellere Konvergenz zu besseren Lösungen, bei 100000 Iterationen sind die Unterschiede aber nur noch sehr gering.

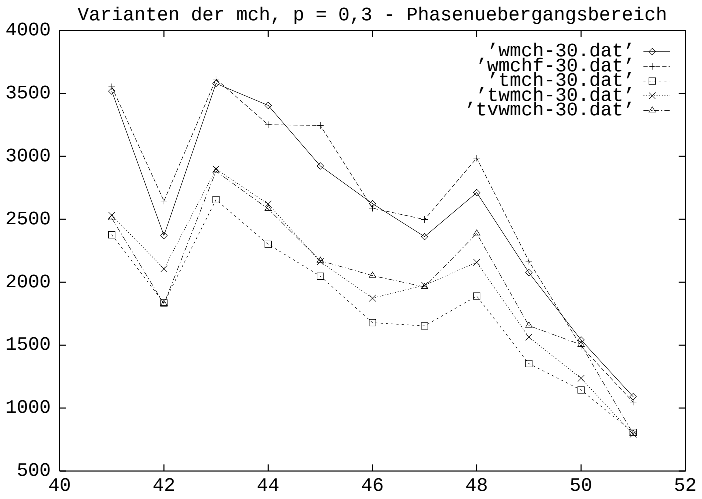

Abbildung 5: binäre CSP, $p = 0,3$.

Tabelle 2: Ergebnisse für MAX-CSP, Mittelwerte über jeweils 25 Probleme. Jedes Problem wurde 10 mal gelöst. $k = 10$.

n	p	q	`mch`	`tmch`	`twmch`	`wmch`	`tvwwmch`
20	0,1	6	5,9	5,6	5,6	5,6	5,6
20	0,5	51	10,2	7,5	7,4	7,4	7,4
20	0,9	66	13,4	10,0	10,0	10,2	10,0
50	0,1	29	19,8	15,2	15,1	15,1	15,0
50	0,5	73	37,7	31,0	30,8	30,9	30,7
50	0,9	80	68,1	58,8	58,7	58,7	58,5
100	0,1	52	48,4	40,3	40,4	40,8	40,6
100	0,5	83	118,4	107,7	107,2	109,6	109,1
100	0,9	85	272,0	255,1	254,7	256,7	256,0

4.3 Graphfärbeprobleme

Wir haben verschiedene Erweiterungen der *min conflicts* Heuristik auch an harten Graphenfärbeproblemen aus dem DIMACS Wettbewerb getestet, siehe Tabelle 3. Die Tests wurden für einige Instanzen mit unterschiedlicher Anzahl an zur Verfügung stehenden Farben durchgeführt. Für die Probleme DSJC* wurden maximal 5 Millionen Iterationen je Versuch erlaubt, für die Instanzen *flat300_28_0.col* und *le450_15c.col* maximal eine Million Iterationen. Für `tmch` wurde wieder eine Tabulistenlänge von 2 verwendet. Für `wmch` zeigte sich, daß die beste Walkwahrscheinlichkeit von den einzelnen Instanzen, speziell von der Dichte des Graphen, abhängig ist.

Auch für die Graphfärbeprobleme zeigte sich wieder, daß Varianten mit Tabu Search deutlich besser abgeschnitten haben. Dies zeigt sich einerseits an dem niedrigeren Lösungsaufwand bei den Instanzen und auch daran, daß bei den schwierigsten Problemen bei einem höheren Anteil der Versuche eine Färbung des Graphen gefunden werden konnte. Für die Instanzen *DSJC125.1.col, DSJC125.5.col* und *DSJC250.5.col* konnten jeweils die besten bekannten Lösungen gefunden werden, bei *DSJC125.9.col* fand keiner der Algorithmen eine Lösung für 43 Farben. Die `mch` wurde nur an dem Problem *DSJC125.5.col* getestet. Mit ihr konnten in zwei von 25

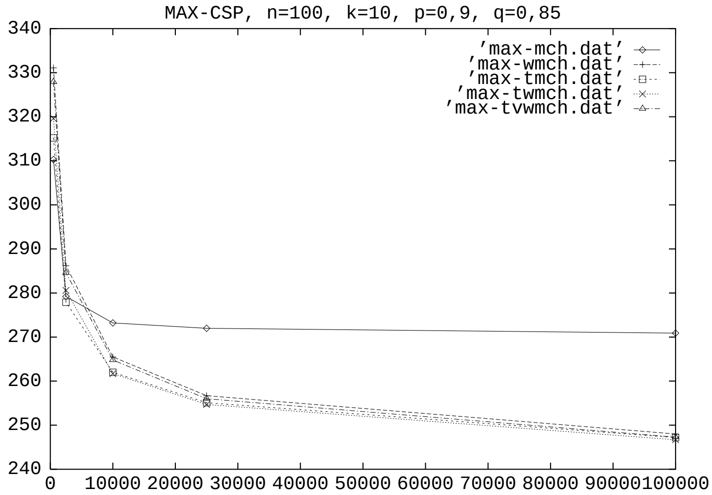

Abbildung 6: Verlauf der verletzten Constraints für MAX-CSP, Die y-Achse gibt die Anzahl der verletzten Constraints an, die x-Achse die Anzahl der Iterationen.

Tabelle 3: Ergebnisse für Graphenfärbeprobleme, k ist die Anzahl der verwendeten Farben. Wurde eine Probleminstanz in jedem Versuch gelöst, geben wir den Median des Lösungsaufwand an, andernfalls gibt (x/y) den Anteil der erfolgreichen Versuche an. k.Erg. besagt, daß keine Versuche für dieses Problem durchgeführt wurden.

Problem	k	`tmch`	`twmch`	`wmch`	`tvwmch`
DSJC125.1.col	5	33452,1	26243,5	67879,5	43846,5
DSJC125.5.col	18	12980,0	15815,0	25290.0	19773,5
DSJC125.5.col	17	(36/50)	(15/25)	(17/50)	(12/50)
DSJC125.9.col	44	17026,0	15161,0	18704,0	13770,0
DSJC250.5.col	30	70570,0	144878,0	462421,0	460533,0
DSJC250.5.col	29	(13/25)	k.Erg.	(1/25)	(1/25)
DSJC500.5.col	48	61479,5	k.Erg.	(8/10)	k.Erg.
flat300_28_0.col	34	59208,0	172948,5	(5/10)	k.Erg.
le450_15c.col	22	21736,5	(8/10)	90296,5	k.Erg.

Versuchen mit maximal 2 Millionen[15] Iterationen eine Färbung mit 18 Farben gefunden werden. Daran zeigt sich, daß die Erweiterungen auch bei Graphfärbeproblemen deutliche Verbesserungen bieten. Für die stärker strukturierten Probleme *flat300_28_0.col* und *le450_15c.col* konnte kein Algorithmus die optimale Lösung von 28 bzw. 15 finden. `tmch` war in der Lage eine Lösung mit 33 Farben für *flat300_28_0.col* und eine mit 20 Farben für *le450_15c.col* zu finden, was mit `wmch` nicht erreicht werden konnte.

Man beachte, daß für GSAT berichtet wurde [25], daß die Probleminstanz *DSJC500.5.col* aufgrund ihrer Größe wegen Speicherproblemen durch die Kodierung als SAT-Problem nicht gelöst werden konnte. Für die *min conflicts* Heuristik spielen dagegen Speichergrenzen auch bei deutlich grösseren Instanzen noch keine Rolle.

[15] MAX-CHANGES wurde auf 1250 gesetzt, ein Versuch bestand also aus 1600 Anwendungen der *min conflicts* Heuristik mit zufällig erzeugten Ausgangslösungen.

5 Zusammenfassung

In diesem Artikel haben wir Varianten der *min conflicts* Heuristik mit Tabu Search und *random walk* empirisch untersucht. Es zeigte sich, daß in Verbindung mit der Metaheuristik Tabu Search die Leistungsfähigkeit der *min conflicts* Heuristik signifikant gesteigert werden kann. In der emprischen Untersuchung konnte anhand von zufällig erzeugten CSPs, MAX-CSP und einigen harten Graphfärbeproblemen gezeigt werden, daß Tabu Search auch bessere Ergebnisse als die sonst meist verwendete Variante der *min conflicts* Heuristik mit *random walk* liefert. Dadurch zeigt sich, daß auch für schwere CSP eine stärker deterministische Leitung der Suche, wie sie durch Tabu Search gegeben wird, durchaus sinnvoll ist. Es ergeben sich auch Implikationen für lokale Suchverfahren für SAT. So wurde in [16] gezeigt, daß Tabu Search auch für SAT sehr effektiv ist und den mittleren Lösungsaufwand signifikant reduzieren kann.

Die *min conflicts* Heuristik wurde ursprünglich in [17] beschrieben. Die Heuristik war inspiriert von einem neuronalen Netz, das zum Scheduling des Hubble Space Teleskops verwendet wurde. Zu den verschiedenen Verbesserungen, die an der grundlegenden Form der Heuristik vorgenommen wurden, gehört die *Breakout*-Methode [18] sowie das auf dieser Strategie zur Vermeidung lokaler Minima beruhende GENET [3]. Die in Kapitel 2.2 vorgestellte Variante mit *random walk* ist in [32] beschrieben. In diesem Artikel befindet sich auch ein erster Ansatz mit Tabu Search. Dieser unterscheidet sich von unserer Anwendung von Tabu Search insbesondere in der Auswahl der Nachbarschaftszüge und der Verbindung zur *min conflicts* Heuristik. Eine andere Variante von *random walk* für die *min conflicts* Heuristik wurde in [21] vorgeschlagen, inspiriert von einem Vergleich zur *weak-commitment* Methode [33].

Anscheinend ist die Rolle von Tabu Search für CSP und auch für SAT bisher deutlich unterschätzt worden. Diese Arbeit aber zeigt, daß es sich durchaus lohnt, dieser Metaheuristik auch für Constraint Satisfaction Probleme stärkere Beachtung zu schenken.

Literatur

[1] W. Bibel. "Intellektik" statt "KI" — Ein ernstgemeinter Vorschlag. *Rundbrief der Fachgruppe Künstliche Intelligenz in der Gesellschaft für Informatik*, 22:15–16, Dezember 1980.

[2] D. Clark, J. Frank, I. Gent, E. MacIntyre, N. Tomov, und T. Walsh. Local Search and the Number of Solutions. In E. Freuder (Hg.), *Principles and Practice of Constraint Programming - CP'96*, Band 1118 in *Lecture Notes in Computer Sciene*, Seite 119–133. Springer Verlag, 1996.

[3] A. Davenport, E. Tsang, C.J. Wang, und K. Zhu. GENET: A Connectionist Architecture for Solving Constraint Satisfaction Problems by Iterative Improvement. In *Proceedings of the 12th National Conference on Artificial Intelligence*, 1994.

[4] R. Dechter, I. Meiri, und J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61–95, 1991.

[5] E.C. Freuder. The Many Paths to Satisfaction. In *Constraint Processing*, Band 923 in *Lecture Notes in Computer Science*, Seite 103–119. Springer Verlag, 1994.

[6] E.C. Freuder und R.J. Wallace. Partial Constraint Satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[7] I. Gent und T. Walsh. Unsatisfied Variables in Local Search. In J. Hallam (Hg.), *Hybrid Problems, Hybrid Solutions*, Seite 73–85, Amsterdam, 1995. IOS Press.

[8] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. 13(5):533–549, 1986.

[9] F. Glover, E. Taillard, und D. de Werra. A User's Guide to Tabu Search. *Annals of Operations Research*, 41:3–28, 1993.

[10] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company Inc., 1989.

[11] H.W. Guesgen und J. Hertzberg. *A Perspective of Constraint-Based Reasoning*, Band 597 in *Lecture Notes in Artificial Intelligence.* Springer Verlag, 1992.

[12] P. Hansen und B. Jaumard. Algorithms for the Maximum Satisfiability Problem. *Computing*, 44:279–303, 1990.

[13] H. Kautz und B. Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proceedings of the 13th National Conference on Artificial Intelligence*, Band 2, Seite 1194–1201. MIT Press, 1996.

[14] S. Kirkpatrick, C.D. Gelatt Jr. und M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[15] A.K. Mackworth. *Encyclopedia of Artificial Intelligence*, Band 1, Kapitel Constraint Satisfaction, Seite 205–211. John Wiley & Sons, 1987.

[16] B. Mazure, L. Sais, und E. Gregoire. TWSAT: A New Local Search Algorithm for SAT – Performance and Analysis. In *CP'95 Workshop on Solving Hard Problems*, 1995.

[17] S. Minton, M.D. Johnston, A.B. Philips, und P. Laird. Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method. In *Proceedings of the 9th National Conference on Artificial Intelligence*, Seite 17–24, 1990.

[18] P. Morris. The Breakout Method for Escaping from Local Minima. In *Proceedings of the 11th National Conference on Artificial Intelligence*, Seite 40–45, 1993.

[19] I.H. Osman und G. Laporte. Metaheuristics: A Bibliography. *Annals of Operations Research*, 63:513–628, 1996.

[20] A.J. Parkes und J.P. Walser. Tuning Local Search for Satisfiability Testing. In *Proceedings of the 13th National Conference on Artificial Intelligence*, Band 1, Seite 356–362, 1996.

[21] D. Pothos und B. Richards. An Empirical Study of Min-Conflict Hill Climbing and Weak Commitment Search. In *CP'95 Workshop on Solving Hard Problems*, 1995.

[22] P. Prosser. An Empirical Study of Phase Transitions in Binary Constraint Satisfaction Problems. Forschungsbericht AISL-49-94, Department of Computer Science, University of Strathclyde, 1994.

[23] N. Sadeh, K. Sycara, und Y. Xiong. Backtracking Techniques for the Job Shop Scheduling Constraint Satisfaction Problem. *Arificial Intelligence*, 76:455–480, 1995.

[24] N.M. Sadeh und M.S. Fox. Variable and Value Ordering Heuristics for the Job Shop Scheduling Constraint Satisfaction Problem. *Artificial Inteligence*, 86:1–41, 1996.

[25] B. Selman und H.A. Kautz. Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Seite 290–295, 1993.

[26] B. Selman, H.A. Kautz, und B. Cohen. Noise Strategies for Improving Local Search. In *Proceedings of the 12th National Conference on Artificial Intelligence*, Seite 337–343. MIT press, 1994.

[27] B. Selman, H. Levesque, und D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the 10th National Conference on Artificial Intelligence*, Seite 440–446. MIT press, 1992.

[28] B.M. Smith. Phase Transitions and the Mushy Region in Constraint Satisfaction Problems. In *Proceedings of the 11th European Conference on Artificial Intelligence*, Seite 100–104, 1994.

[29] T.Hogg, B.A. Huberman, und C.P. Williams. Phase Transitions and the Search Problem (Editorial). *Artificial Intelligence*, 81(1–2):1–16, 1996.

[30] R.J.M. Vaessens, E.H.L. Aarts, und J.K. Lenstra. Job Shop Scheduling by Local Search. Forschungsbericht COSOR Memorandum 94-05, revised version, Eindhoven University of Technology, 1994.

[31] R.J. Wallace. Enhancements of Branch and Bound Methods for the Maximal Constraint Satisfaction Problem. In *Proceedings of the 13th National Conference on Artificial Intelligence*, Band 1, Seite 188–195, 1996.

[32] R.J. Wallace und E. Freuder. Heuristic Methods for Over-Constrained Constraint Satisfaction Problems. In M. Jampel, E. Freuder, und M. Maher (Hg.), *Over-Constrained Systems*, Band 1106 in *Lecture Notes in Computer Sciene*. Springer Verlag, 1996.

[33] M. Yokoo. Weak-commitment Search for Solving Constraint Satisfaction Problems. In *Proceedings of the 12th National Conference on Artificial Intelligence*, Seite 313–318, 1994.