Universität Innsbruck

Was ist das SAT-Problem?

Mathias MITTERDORFER
mathias.mitterdorfer@student.uibk.ac.at

1 Einleitung

Dieses Dokument gibt einen kurzen Überblick über das SAT-Problem, dessen Lösungsmöglichkeiten und Anwendungen in der Praxis. Es beinhaltet jedoch keine genaue Definitionen von Algorithmen. Viel mehr werden einige Lösungsmöglichkeiten aufgezählt und kurz beschrieben. Bei Interesse der LeserIn wird auf die Referenzen verwiesen. Zum Verständnis des Artikels ist ein minimales Vorwissen im Bereich Logik von Nöten.

Zu Beginn werden grundlegende Begriffe erläutert (2) und die Komplexiät (3) des Problems bestimmt. In Kapitel 4 werden konkrete Lösungsmöglichkeiten besprochen, beginnend mit einem geschichtlichen Überblick der Verfahren (4.1) bis hin zu modernen SAT-Solvern (4.2). Zum Schluss werden noch kurz praktische Anwendungen angeführt (5).

2 Begriffserklärung

Das SAT-Problem - SAT vom englischen Wort satisfiability, Erfüllbarkeit - bezeichnet ein Entscheidungsproblem für aussagenlogische Formeln. Genauer gesagt wird eine Belegung ($wahr$ oder $falsch$) für die Variablen in der Formel gesucht, welche die Formel $wahr$, also erfüllbar, macht. Das „Problem“ kommt daher, da es bis heute nur Algorithmen gibt, welche in exponentieller Zeit Formeln auf Erfüllbarkeit testen können.

3 Komplexität

1971 konnte der kanadische Wissenschaftler Stephen Arthur Cook (* 14. Dezember 1939) nachweisen, dass das SAT-Problem NP-vollständig ist.[3] Der Artikel besagt auch, dass jedes NP-vollständige Problem in polynomieller Zeit durch eine *deterministische Turingmaschine* auf das SAT-Problem reduziert werden kann (und umgekehrt). In anderen Worten: Löst man ein NP-vollständiges Problem, hat man alle gelöst.

Der Artikel warf auch eine der bedeutendsten Fragestellungen der Komplexitätstheorie auf: $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$. Dieses gehört zu den sieben *Millennium Prize Problems*[1]. Ist man also

[1] http://www.claymath.org/millennium/

im Stande das SAT-problem in polynomieller Zeit mit einer *deterministischen Turingmaschine* zu lösen, so erhält man eine Million Dollar.

4 SAT-Solver

Da der Mensch bekanntlich bei jedem Problem nach einer Lösung strebt, gibt es diese auch für das SAT-Problem. Das Erfüllbarkeitsproblem wird mit sogenannten SAT-Solvern gelöst.

4.1 Geschichtlicher Überblick

1960 Davis und Putnam entwickelten einen Algorithmus für das SAT-Problem basierend auf *Resolution*.

1962 Davis, Logemann und Loveland (*DPLL*) verbesserten den Algorithmus von 1960.[4] Den Entscheidungsweg dieses Algorithmus kann man mit dem eines Graphen (Baum) vergleichen, bei dem mit Hilfe von Tiefensuche eine erfüllbare Belegung gefunden werden soll. Dadurch wurde das Speicherplatzproblem der Vorgängerversion von 1960 gelöst.

Dieser Algorithmus ist noch immer das Grundgerüst für moderne SAT-Solver.

1986 R. Bryant entwarf einen Algorithmus, welcher aus der Formel ein Binary Decision Diagram (*BDD*) bildet.[1] Die Konvertierung vereinfacht ein Lösen von vielen ähnlichen Problemen mit den selben Variablen. Jedoch stellte sich in der Praxis heraus, dass die Umwandlung in ein BDD zu aufwendig ist (auch NP-vollständig).

1992 B. Selman, H. Levesque, and D. Mitchell designenten den *GSAT* (G steht für greedy) Algorithmus.[9] Dieser rät am Anfang eine Startbelegung für alle Variablen und versucht dann durch geschickte Invertierung von Variablen auf eine Belegung zu stoßen, welche die Formel erfüllt. Normalerweise liefert der Algorithmus nach dem ersten Durchlauf kein korrektes Ergebnis (keine erfüllbare Belegung); daraufhin wird zufällig eine neue Startbelegung ausgewählt. Es ist schon erstaunlich, dass diese Methode 1992 die schnellste Möglichkeit war um das SAT-Problem zu lösen. Dieser Algorithmus ist aber unvollständig (incomplete), d.h., der GSAT-Solver kann nur bestimmen, ob das Problem erfüllbar ist; die Unerfüllbarkeit kann jedoch nicht nachgewiesen werden.

1996 Stålmarck's Algorithmus[10] basiert auf der sogenannten *dilemma rule*: Setzen einer Variable x auf 0, durchführen des SAT-Vorgangs, bis der Solver keine weiteren Belegungen für Variablen zuweist. Danach x mit 1 belegen und wieder den Solver starten und warten bis dieser stoppt. Der Durchschnitt beider Aufrufe wird gebildet (also jene Belegung von Variablen, welche sich bei $x = 0$ und $x = 1$ nicht verändert). Z.B. wenn $x = 0 \wedge x = 1 \implies y = 0$ gilt, muss y auf jeden Fall den Wert 0 haben.

1996 *SATO* (Satisfiability Testing Optimized): Hier wurde ein schneller Algorithmus zum Testen ob eine Formel durch den momentanen Zustand ihrer Variablen erfüllt ist, eingeführt.[11]

2001 *Chaff*: Effektiver Algorithmus um die nächste Variable festzustellen von welcher eine Belegung ermittelt wird. Dies wird durch eine höhere Gewichtung von Klauseln, bei denen sich erst kürzlich eine Variable verändert hat, erreicht.[7]

4.2 Moderne SAT-Solver

Wie in Abschnitt 3 aufgezeigt wurde, ist das SAT-Problem NP-vollständig. Soll das heißen, dass es sinnlos ist, für große Eingaben nach einer Lösug zu suchen, da die Berechnungszeit exponentiell ansteigt und somit die Lösung nicht in ausreichender Zeit gefunden werden kann? Nein, Gott sei Dank nicht. Die SAT-Verfahren wurden so weiterentwickelt, dass Probleme mit tausenden von Variablen in Sekundenbruchteilen gelöst werden können. „*Es sieht fast so aus, als ob das NP-Vollständigkeitsergebnis durch die praktische Erfahrung mit SAT-Solvern überwunden wurde.*“[8]

Die meisten SAT-Solver benötigen eine Eingabe der Problemstellung in *KNF*[2].

Um sich praktisch mit dem Thema SAT-Solver zu befassen, findet man online einige Open Source Solver zum Kennenlernen - MiniSAT(C++)[3], picosat(C)[4], SAT4J (Java)[5],... .

Auf der Website http://www.satcompetition.org/ sind die schnellsten SAT-Solver der Gegenwart aufgelistet.

5 Das SAT-Problem in der Praxis

Eines der häufigsten Anwendungsgebiete für moderne SAT-Solver ist das *Bounded Model Checking*.[2] Dies wird dazu verwendet um Systeme auf gewisse Eigenschaften zu testen. Man formuliert das System und die zu erfüllenden Eigenschaften als logische Formel und lässt es den SAT-Solver überprüfen. Der Vorteil, wenn man einen SAT-Solver verwendet, liegt darin, dass dieser nicht nur „ja“ oder „nein“ als Antwort liefert, sondern auch im nicht erfolgreichen Fall ein Gegenbeispiel anbringt.

Weitere konkrete Anwendungen sind z.B.:

- *Planning as Satisfiability*[6]: beschreibt ein Problem der Künstlichen Intelligenz, wobei durch einen Graph alle möglichen Handlungen und dessen Abhängingkeiten abgebildet werden und von einem Startzustand ausgehend ein akzeptierender Zustand gesucht wird. Der kürzeste Pfad zu diesem Zustand ist der Plan.

[2] Konjunktive Normalform, siehe [5]
[3] http://www.minisat.se/
[4] http://fmv.jku.at/picosat/
[5] http://www.sat4j.org/

- *Automated Theorem Proving*: Es werden mathematische Formeln mit Hilfe von Computerprogrammen bewiesen. Hier wird typischerweise *Bounded Model Checking* angewandt.

6 Schlussfolgerung

Das SAT-Problem, als *das* NP-vollständige Problem, tritt in vielen Szenarien auf. Es ist sinnvoll nach möglichst effizienten Lösungsmöglichkeiten zu suchen, da viele Probleme in der Praxis nicht in polynomieller Zeit lösbar sind. Moderne SAT-Solver sind meistens der schnellste Weg um NP-vollständige Probleme zu lösen. Durch Forschungen werden laufend neue Heuristiken entwickelt, welche zur Beschleunigung von SAT-Solvern beitragen.

Literatur

[1] Bryant, Randal E.: Graph-based algorithms for boolean function manipulation. In: *Computers, IEEE Transactions on* 100 (1986), Nr. 8, S. 677–691

[2] Clarke, Edmund ; Biere, Armin ; Raimi, Richard ; Zhu, Yunshan: Bounded model checking using satisfiability solving. In: *Formal Methods in System Design* 19 (2001), Nr. 1, S. 7–34

[3] Cook, Stephen A.: The complexity of theorem-proving procedures. In: *Proceedings of the third annual ACM symposium on Theory of computing* ACM, 1971, S. 151–158

[4] Davis, Martin ; Logemann, George ; Loveland, Donald: A machine program for theorem-proving. In: *Communications of the ACM* 5 (1962), Nr. 7, S. 394–397

[5] Huth, Michael ; Ryan, Mark: *Logic in Computer Science: Modelling and reasoning about systems.* Cambridge University Press, 2004

[6] Kautz, Henry ; Selman, Bart u. a.: Planning as satisfiability. In: *Proceedings of the 10th European conference on Artificial intelligence*, 1992, S. 359–363

[7] Moskewicz, Matthew W. ; Madigan, Conor F. ; Zhao, Ying ; Zhang, Lintao ; Malik, Sharad: Chaff: Engineering an efficient SAT solver. In: *Proceedings of the 38th annual Design Automation Conference* ACM, 2001, S. 530–535

[8] Schöning, Uwe: Das SAT-Problem. In: *Informatik-Spektrum* 33 (2010), Nr. 5, S. 479–483

[9] Selman, Bart ; Levesque, Hector ; Mitchell, David: A new method for solving hard satisfiability problems. In: *Proceedings of the tenth national conference on Artificial intelligence*, 1992, S. 440–446

[10] Stålmarck, Gunnar: A proof theoretic concept of tautological hardness. In: *Unpublished manuscript* (1994)

[11] Zhang, Hantao: SATO: An efficient prepositional prover. In: *Automated Deduction-CADE-14.* Springer, 1997, S. 272–275