Fahiem Bacchus
Toby Walsh (Eds.)

LNCS 3569

Theory and Applications of Satisfiability Testing

8th International Conference, SAT 2005
St Andrews, UK, June 2005
Proceedings

Springer

Fahiem Bacchus Toby Walsh (Eds.)

Theory and Applications of Satisfiability Testing

8th International Conference, SAT 2005
St Andrews, UK, June 19-23, 2005
Proceedings

Volume Editors

Fahiem Bacchus
University of Toronto, Department of Computer Science
6 King's College Road, Toronto, Ontario, M5S 3H5, Canada
E-mail: fbacchus@cs.toronto.edu

Toby Walsh
National ICT Australia and University of New South Wales
School of Computer Science and Engineering
Sydney 2502, Australia
E-mail: tw@cse.unsw.edu.au

Library of Congress Control Number: 2005927321

CR Subject Classification (1998): F.4.1, I.2.3, I.2.8, I.2, F.2.2, G.1.6

ISSN 0302-9743
ISBN-10 3-540-26276-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-26276-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11499107 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3569

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Preface

The 8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005) provided an international forum for the most recent research on the satisfiablity problem (SAT).

SAT is the classic problem of determining whether or not a propositional formula has a satisfying truth assignment. It was the first problem shown by Cook to be NP-complete. Despite its seemingly specialized nature, satisfiability testing has proved to extremely useful in a wide range of different disciplines, both from a practical as well as from a theoretical point of view. For example, work on SAT continues to provide insight into various fundamental problems in computation, and SAT solving technology has advanced to the point where it has become the most effective way of solving a number of practical problems.

The SAT series of conferences are multidisciplinary conferences intended to bring together researchers from various disciplines who are interested in SAT. Topics of interest include, but are not limited to: proof systems and proof complexity; search algorithms and heuristics; analysis of algorithms; theories beyond the propositional; hard instances and random formulae; problem encodings; industrial applications; solvers and other tools.

This volume contains the papers accepted for presentation at SAT 2005. The conference attracted a record number of 73 submissions. Of these, 26 papers were accepted for presentation in the technical programme. In addition, 16 papers were accepted as shorter papers and were presented as posters during the technical programme. The accepted papers and poster papers cover the full range of topics listed in the call for papers.

We would like to thank a number of people and organizations: Ian Miguel, the Local Chair who helped us organize the conference remotely; our generous sponsors who helped us to keep costs down, especially for students; Daniel Le Berre and Laurent Simon for once again organizing the SAT Solver Competition; and Massimo Narizzano and Armando Tacchella for the QBF Solver Evaluation. We would also like to thank the members of the Programme Committee and the additional referees who contributed in the paper-reviewing process.

St Andrews
June 2005

Fahiem Bacchus, Toby Walsh

Organization

Conference Organization

Conference Chairs: Fahiem Bacchus (University of Toronto, Canada)
Toby Walsh (National ICT Australia and University of NSW, Australia)
Local Chair: Ian Miguel (University of St Andrews, UK)

Programme Committee

Dimitris Achlioptas
Fadi Aloul
Clark Barrett
Constantinos Bartzis
Paul Beame
Armin Biere
Ronen Brafman
Alessandro Cimatti
Adnan Darwiche
Alvaro del Val
Enrico Giunchiglia
Eugene Goldberg
Ziyad Hanna
Edward Hirsch
Henry Kautz
Eleftherios Kirousis
Hans Kleine Büning
Daniel Le Berre
Chu-Min Li
Fangzhen Lin
Sharad Malik
João Marques-Silva
Ilkka Niemela
Toniann Pitassi
Steve Prestwich
Jussi Rintanen
Lakhdar Sais
Karem Sakallah
Laurent Simon
Stefan Szeider
Mirek Truszczynski
Allen Van Gelder
Hans van Maaren
Lintao Zhang

Sponsors

Cadence Design Systems
Intel Corporation
Intelligence Information Systems Institute, Cornell
Microsoft Research
CoLogNet Network of Excellence

Additional Referees

Zaher S. Andraus
Pierre Bonami
Uwe Bubeck
Yin Chen
Sylvie Coste-Marquis
Nadia Creignou
Stefan Dantchev
Sylvain Darras
Gilles Dequen
Laure Devendeville
Niklas Een
Malay K. Ganai

Aarti Gupta
Jean-Luc Guerin
Keijo Heljanko
Jinbo Huang
Dmitry Itsyson
Matti Järvisalo
Tommi Junttila
Bernard Jurkowiak
Zurab Khasidashvili
Arist Kojevnikov
Ioannis Koutis
Alexander Kulikov
Oliver Kullman
Theodor Lettmann
Lengning Liu
Ines Lynce
Yogesh Mahajan
Marco Maratea
Victor Marek
Bertrand Mazure
Maher N. Mneimneh
Alexander Nadel
Massimo Narizzano
Sergey Nikolenko
Nishant Ninha
Thomas Schiex
Armando Tacchella
Muralidhar Talupur
Daijue Tang
Yinlei Yu

Table of Contents

Preface

Solving Over-Constrained Problems with SAT Technology*

Josep Argelich[1] and Felip Manyà[2]

[1] Computer Science Department, Universitat de Lleida,
Jaume II, 69, E-25001 Lleida, Spain
josep@eup.udl.es
[2] Artificial Intelligence Research Institute (IIIA-CSIC),
Campus UAB, 08193 Bellaterra, Spain
felip@iiia.csic.es

Abstract. We present a new generic problem solving approach for over-constrained problems based on Max-SAT. We first define a clausal form formalism that deals with blocks of clauses instead of individual clauses, and that allows one to declare each block either as *hard* (i.e., must be satisfied by any solution) or *soft* (i.e., can be violated by some solution). We then present two Max-SAT solvers that find a truth assignment that satisfies all the hard blocks of clauses and the maximum number of soft blocks of clauses. Our solvers are branch and bound algorithms equipped with original lazy data structures; the first one incorporates static variable selection heuristics while the second one incorporates dynamic variable selection heuristics. Finally, we present an experimental investigation to assess the performance of our approach on a representative sample of instances (random 2-SAT, Max-CSP, and graph coloring).

1 Introduction

The SAT-based problem solving approach presents some limitations when solving many real-life problems due to the fact that it only provides a solution when the formula that models the problem we are trying to solve is shown to be satisfiable. Nevertheless, in many combinatorial problems, some potential solutions could be acceptable even when they violate some constraints. If these violated constraints are ignored, solutions of bad quality are found, and if they are treated as mandatory, problems become unsolvable. This is our motivation to extend the SAT formalism to solve over-constrained problems. In such problems, the goal is to find the *solution* that *best respects* the constraints of the problem.

In this paper we will consider that all the constraints are *crisp* (i.e., they are either completely satisfied or completely violated), but constraints can be

* Research partially supported by projects TIN2004-07933-C03-03 and TIC2003-00950 funded by the *Ministerio de Educación y Ciencia*. The second author is supported by a grant *Ramón y Cajal*.

F. Bacchus and T. Walsh (Eds.): SAT 2005, LNCS 3569, pp. 1–15, 2005.

either *hard* (i.e., must be satisfied by any solution) or *soft* (i.e., can be violated by some solution). A solution *best respects* the constraints of the problem if it satisfies all the hard constraints and the maximum number of soft constraints. In the literature of over-constrained problems, *fuzzy* constraints (i.e., intermediate degrees of satisfaction are allowed), as well as other ways of defining that a solution *best respects* the constraints of the problem, are considered. We invite the reader to consult [12] for a recent survey on different CSP approaches to solving over-constrained problems.

Given a combinatorial problem which can be naturally defined by a set of constraints over finite-domain variables, we have that each constraint is often encoded as a set (block) of Boolean clauses in such a way that a constraint is satisfiable if all those clauses are satisfied by some truth assignment and is violated if at least one of those clauses is not satisfied by any truth assignment. Thus, in contrast to the usual approach, the concept of satisfaction in SAT-encoded over-constrained problems refers to blocks of clauses instead of individual clauses. This led in turn to design Max-SAT-like solvers that deal with blocks of clauses instead of individual clauses, and exploit the new structure of the encodings.

In this paper we present a new generic problem solving approach for over-constrained problems based on Max-SAT. We first define a clausal form formalism that deals with blocks of clauses instead of individual clauses, and that allows one to declare each block either as *hard* (i.e., must be satisfied by any solution) or *soft* (i.e., can be violated by some solution). We call *soft CNF formulas* to this new kind of formulas. We then present two Max-SAT solvers that find a truth assignment that satisfies all the hard blocks of clauses and the maximum number of soft blocks of clauses. Our solvers are branch and bound algorithms equipped with original lazy data structures; the first one incorporates static variable selection heuristics while the second one incorporates dynamic variable selection heuristics. Finally, we present an experimental investigation to assess the performance of our approach on a representative sample of instances (random 2-SAT, Max-CSP, and graph coloring).

Problem solving of over-constrained problems with Max-SAT local search algorithms has been investigated before in [8, 4]. In that case, the authors distinguish between hard and soft constraints at the clause level, but they do not incorporate the notion of blocks of hard and soft clauses. The notion of blocks of clauses provides a more natural way of encoding soft constraints. Besides, to the best of our knowledge, the treatment of soft constraints with exact Max-SAT solvers has not been considered before.

The paper is structured as follows. In Section 2 we introduce the formalism of soft CNF formulas. In Section 3 we describe a solver for soft CNF formulas with static variable selection heuristics. In Section 4 we describe a solver for soft CNF formulas with dynamic variable selection heuristics. In Section 5 we report the experimental investigation we performed to assess the performance of our formalism and solvers. Finally, we present some concluding remarks.

2 Soft CNF Formulas

We define the syntax and semantics of soft CNF formulas, which are an extension of Boolean clausal forms that we use to encode over-constrained problems.

Definition 1. *A soft CNF formula is formed by a set of pairs* $(clause, label)$*, where clause is a Boolean clause and label is either* h_i *or* s_i *for some* $i \in \mathbb{N}$*. A hard block of a soft CNF formula is formed by all the pairs* $(clause, label)$ *with the same label* h_i*, and a soft block is formed by all the pairs* $(clause, label)$ *with the same label* s_i*.*

All the clauses with the same label h_i (s_i) model the same hard (soft) constraint.

Definition 2. *A truth assignment satisfies a hard block of a soft CNF formula if it satisfies all the clauses of the block. A truth assignment satisfies a soft CNF formula* ϕ *if it satisfies all the hard blocks of* ϕ*. We say then that* ϕ *is satisfiable. A soft CNF formula* ϕ *is unsatisfiable if there is no truth assignment that satisfies all the the hard blocks of* ϕ*. A truth assignment satisfies a soft block if it satisfies all the clauses of the block. A truth assignment is a solution to a soft CNF formula* ϕ *if it satisfies all the hard blocks of* ϕ *and the maximum number of soft blocks.*

Definition 3. *The Soft-SAT problem is the problem of finding a solution to a Soft CNF formula.*

Example 1. We want to solve the problem of coloring a graph with two colors in such a way that the minimum number of adjacent vertices are colored with the same color. If we consider the graph with vertices $\{v_1, v_2, v_3\}$ and with edges $\{(v_1, v_2), (v_1, v_3), (v_2, v_3)\}$, that problem is encoded as a Soft-SAT instance as follows: (i) the set of propositional variables is $\{v_1^1, v_1^2, v_2^1, v_2^2, v_3^1, v_3^2\}$; the intended meaning of variable v_i^j is that vertex v_i is colored with color j; (ii) there is one hard black formed by the following at-least-one and at-most-one clauses:

$$(v_1^1 \vee v_1^2, h_1), (\neg v_1^1 \vee \neg v_1^2, h_1), (v_2^1 \vee v_2^2, h_1), (\neg v_2^1 \vee \neg v_2^2, h_1), (v_3^1 \vee v_3^2, h_1), (\neg v_3^1 \vee \neg v_3^2, h_1);$$

and (iii) there is a soft block for every edge:

$$\begin{array}{c} (\neg v_1^1 \vee \neg v_2^1, s_1), (\neg v_1^2 \vee \neg v_2^2, s_1), \\ (\neg v_1^1 \vee \neg v_3^1, s_2), (\neg v_1^2 \vee \neg v_3^2, s_2), \\ (\neg v_2^1 \vee \neg v_3^1, s_3), (\neg v_2^2 \vee \neg v_3^2, s_3). \end{array}$$

The use of blocks is relevant for two reasons. On the one hand, it provides to the user information in a more natural way about constraint violations. On the other hand, it allows us to get more propagation at certain nodes (this point is discusses in the next section). Besides, the structure of blocks will be important when we extend our formalism to deal with fuzzy constraints.

3 Soft-SAT-S: A Solver with Static Variable Selection Heuristic

The space of all possible assignments for a soft CNF formula ϕ can be represented as a search tree, where internal nodes represent partial assignments and leaf nodes represent complete assignments. The branch and bound algorithm for solving the Max-SAT problem of soft CNF formulas with static variable selection heuristics that we have designed and implemented, called Soft-SAT-S, explores that search tree in a depth-first manner. At each node, the algorithm backtracks if the current partial assignment violates some clause of the hard blocks, and applies the one-literal rule[1] to the literals that occur in unit clauses of hard blocks.[2] If the current partial assignment does not violate any clause of the hard blocks, the algorithm compares the number of soft blocks unsatisfied by the best complete assignment found so far, called upper bound (ub), with the number of soft blocks unsatisfied by the current partial assignment, called lower bound (lb). Obviously, if $ub \leq lb$, a better assignment cannot be found from this point in search. In that case, the algorithm prunes the subtree below the current node and backtracks to a higher level in the search tree. If $ub > lb$, it extends the current partial assignment by instantiating one more variable, say p, which leads to create two branches from the current branch: the left branch corresponds to instantiate p to false, and the right branch corresponds to instantiate p to true. In that case, the formula associated with the left (right) branch is obtained from the formula of the current node by applying the one-literal rule [11] using the literal $\neg p$ (p). The value that ub takes after exploring the entire search tree is the minimum number of soft blocks that cannot be satisfied by a complete assignment.

In branch and bound Max-SAT algorithms like [2, 17], the lower bound is the sum of the number of unsatisfied clauses by the current partial assignment plus an underestimation of the number of clauses that will become unsatisfied if we extend the current partial assignment into a complete assignment, which is calculated taking into account the inconsistency counts of the variables not yet instantiated. The concept of inconsistency counts cannot be easily extended to soft blocks[3] and our lower bound is not so powerful as the lower bounds of [2, 15, 17, 18]. In Soft-SAT-S, like in [2, 17], the initial lower bound is obtained with a GSAT-like [14] local search algorithm.

In Section 5 we define the notion of inconsistency counts for SAT encoded Max-CSP and graph coloring instances by exploiting the structure hidden in the encoding, and we are able to define a lower bound that incorporates an

[1] Given a literal $\neg p$ (p), the one-literal rule [11] deletes all the clauses containing the literal $\neg p$ (p) and removes all the occurrences of the literal p ($\neg p$).

[2] Observe that this pruning technique cannot be applied to exact Max-SAT solvers that deal with individual clauses; in Max-SAT solvers each clause can be viewed as a soft block.

[3] This is due to the fact that a block is unsatisfied by an interpretation I when I does not satisfy one clause of the block.

underestimation of the number of soft blocks that will become unsatisfied if we extend the current partial assignment into a complete assignment.

When branching is done, algorithms for Max-SAT like [2, 17, 3] apply the one-literal rule (simplifying with the branching literal) instead of applying unit propagation (i.e., the repeated application of the one-literal rule until a saturation state is reached) as in the Davis-Putnam-style [6] solvers for SAT. If unit propagation is applied at each node, the algorithm can return a non-optimal solution. For example, if we apply unit propagation to $\{p, \neg q, \neg p \vee q, \neg p\}$ using the unit clause $\neg p$, we derive one empty clause while if we use the unit clause p, we derive two empty clauses. However, when the difference between the lower bound and the upper bound is one, unit propagation can be safely applied, because otherwise by fixing to false any literal of any unit clause we reach the upper bound. Soft-SAT-S performs unit propagation in that case too. Moreover, as pointed out before, Soft-SAT-S applies the one-literal rule when a clause of a hard block becomes unit. This propagation, which leads to substantial performance improvements, cannot be safely applied in Max-SAT solvers like [2, 17, 3], and is a key feature of our approach.

Our current version of Soft-SAT-S incorporates two static variable selection heuristics:

- **MO:** We instantiate first the variables that appears Most Often (MO). Ties are broken using the lexicographical order.
- **csp:** In SAT encodings that model CSP variables, each CSP variable with a domain of size k is represented by a set of k Boolean variables $x_1, \ldots, x_k$. We associate a weight to each one of these sets: the sum of the total number of occurrences of each variable of the set. We order the sets according to such weight. Heuristic **csp** instantiates, first and in lexicographical order, the Boolean variables of the set with the highest weight. Then, it instantiates, in lexicographical order, the Boolean variables of the set with the second highest weight, and so on. This heuristic is used, in the experimental investigation, to solve problems with finite-domain variables (Max-CSP and graph coloring). The idea behind this heuristic is to instantiate first the CSP variables that occur most often. This way, we emulate an n-ary CSP branching by means of a binary branching (i.e., we consider all the possible values of the CSP variable under consideration before instantiating another CSP variable). As we will see in the experiments, we get some performance improvements for the fact of dealing with n-ary branchings.

The fact of using static variable selection heuristics allows us to implement extremely efficient data structures for representing and manipulating soft CNF formulas. Our data structures take into account the following fact: we are only interested in knowing when a clause has become unit or empty. Thus, if we have a clause with four variables, we do not perform any operation in that clause until three of the variables appearing in the clause have been instantiated; i.e., we delay the evaluation of a clause with k variables until $k - 1$ variables have been instantiated. In our case, as we instantiate the variables using a static order, we

do not have to evaluate a clause until the penultimate variable of the clause in the static order has been instantiated.

The data structures are defined as follows: For each clause we have a pointer to the penultimate variable of the clause in the static order, and the clauses of a soft CNF formula are ordered by that pointer. We also have a pointer to the last variable of the clause. When a variable p is fixed to true (false), only the clauses whose penultimate variable in the static order is $\neg p$ (p) are evaluated. This approach has two advantages: the cost of backtracking is constant (we do not have to undo pointers like in adjacency lists) and, at each step, we evaluate a minimum number of clauses.

4 Soft-SAT-D: A Solver with Dynamic Variable Selection Heuristic

The second solver we have designed and implemented is Soft-SAT-D, which is like Soft-SAT-S except for the fact that its variable selection heuristics are dynamic. This fact, in turn, did not allow us to implement the data structures we have described in the previous section. The data structures implemented in Soft-SAT-D are the two-watched literal data structures of Chaff [13]. They are also lazy data structures, but are not so efficient because here we need to maintain the watched literals.

Our current version of Soft-SAT-D incorporates two dynamic variable selection heuristics:

- **MO:** We instantiate first the variables that appears Most Often (MO). Ties are broken using the lexicographical order. Observe that we do not use the variable that appears most often in minimum size clauses (heuristic MOMS) because this is difficult to know with the lazy data structures of Chaff. However, most of the instances we used in the experimental investigation contain a big amount of binary clauses.
- **MO-csp:** This is the dynamic version of heuristic *csp* of Soft-SAT-S. We associate a weight to each set of free Boolean variables that encode a same CSP variable: the sum of the total number of occurrences of each variable of the set that has not been yet instantiated. We select the set with the highest weight and instantiate its variables in lexicographical order. Like in heuristic *csp*, we emulate an n-ary branching.

5 Experimental Investigation

We next report the experimental investigation we conducted to evaluate the performance of our problem solving approach. All the experiments were performed on a 2GHz Pentium IV with 512 Mb of RAM under Linux.

We performed experiments with ssoft-SAT solvers as well as with weighted Max-SAT solvers and a Max-CSP solver [10]. The solvers used are the following ones: