MONOGRAFIES DE L'INSTITUT D´INVESTIGACIÓ EN INTEL·LIGÈNCIA ARTIFICIAL

Max-SAT Formalisms with Hard and Soft Constraints

Josep Argelich

Consell Superior d´Investigacions Científiques

MONOGRAFIES DE L'INSTITUT D'INVESTIGACIÓ
EN INTEL·LIGÈNCIA ARTIFICIAL
Number 41

Institut d'Investigació
en Intel·ligència Artificial

Consell Superior
d'Investigacions Científiques

Max-SAT Formalisms with Hard and Soft Constraints

Josep Argelich

Foreword by Felip Manyà

2009 Consell Superior d'Investigacions Científiques
Institut d'Investigació en Intel·ligència Artificial
Bellaterra, Catalonia, Spain.

Series Editor
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Foreword by
Felip Manyà
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Volume Author
Josep Argelich
Departament d'Informàtica i Enginyeria Industrial
Universitat de Lleida

Institut d'Investigació
en Intel·ligència Artificial

Consell Superior
d'Investigacions Científiques

© 2009 by Josep Argelich
NIPO: 472-09-117-X
ISBN: 978-84-00-08849-1
Dip. Legal: B.33300-2009

All rights reserved. No part of this book may be reproduced in any form or by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.
Ordering Information: Text orders should be addressed to the Library of the IIIA, Institut d'Investigació en Intel·ligència Artificial, Campus de la Universitat Autònoma de Barcelona, 08193 Bellaterra, Barcelona, Spain.

A la família.

Contents

Copia gratuita. Personal free copy http://libros.csic.es

© CSIC © del autor o autores / Todos los derechos reservados

Copia gratuita. Personal free copy http://libros.csic.es

List of Figures

Copia gratuita. Personal free copy http://libros.csic.es

© CSIC © del autor o autores / Todos los derechos reservados

List of Tables

Copia gratuita. Personal free copy http://libros.csic.es

© CSIC © del autor o autores / Todos los derechos reservados

List of Algorithms

Copia gratuita. Personal free copy http://libros.csic.es

Foreword

The study of Max-SAT formalisms, and the design and implementation of fast Max-SAT solvers have become very active research topics in the last few years. The scientific community working on Satisfiability has devised novel solving techniques that allow to solve many instances that were beyond the reach of the Max-SAT solvers developed just a few years ago, and has promoted the study of formalism like Partial Max-SAT and Weighted Max-SAT that allow to define Max-SAT encodings of practical combinatorial optimization problems in a natural and compact way. Besides, the Max-SAT Evaluation, which is a co-located event of the International Conference on Theory and Applications of Satisfiability Testing since 2006, has provided new challenging benchmarks and has actively encouraged the development of new solvers.

This monograph, which is based on the Ph.D. dissertation of Dr. Josep Argelich, is concerned with the study of Max-SAT formalisms for solving optimization problems with hard and soft constraints. Among all the contributions, I would like to highlight the introduction of of a new MaxSAT formalism, called Soft-SAT, and the novel solving techniques for Weighted Partial Max-SAT that have been devised.

Soft-SAT deals with blocks of clauses, and because of that it produces compact encodings that avoid the use of auxiliary variables, naturally represents soft constraints, and applies inference techniques which are local to each block. The conducted experimental investigation, with the Soft-SAT solvers that have been developed during the Ph.D. Thesis, provides empirical evidence of the good performance of Soft-SAT for solving combinatorial optimization problems.

On the other hand, the novel solving techniques for Weighted Partial Max-SAT that have been devised during the Ph.D. Thesis include sound inference rules, lower bound computation methods, clause learning and non-chronological backtracking derived from the conflict analysis of both hard and soft constraints, and preprocessing techniques based on variable saturation, randomization and restarts. All these techniques led to the development of two new solvers for Weighted Partial Max-SAT: PMS and W-MaxSatz. Currently, W-MaxSatz is one of the best performing state-of-the-art Weighted Partial Max-SAT solvers, and is widely used by the scientific community.

I hope that you enjoy reading this monograph, which is the fruit of the enthusiasm and effort that the author put into this scientific adventure.

Copia gratuita. Personal free copy http://libros.csic.es

It was an honor and pleasure to serve as thesis advisor of Josep, and I am willing to continue working with him in the future.

Bellaterra, October 2009

Felip Manyà
IIIA-CSIC

© CSIC © del autor o autores / Todos los derechos reservados

Abstract

In this thesis we investigate Max-SAT formalisms for solving combinatorial optimization problems with hard and soft constraints. Such formalisms incorporate the notion of *partiality*; i.e., they contain clauses which are mandatory and clauses (or sets of clauses) which are relaxable. On the one hand, this notion captures the constraints of real problems in a more natural way, and produces more compact encodings. On the other hand, the distinction between mandatory and relaxable clauses has a significant impact on the solving techniques that can be applied in branch and bound solvers.

Firstly, we define a new Max-SAT formalism, called Soft-SAT, that deals with a block of hard clauses and several blocks of soft clauses. In this formalism, solving an instance consists in finding a truth assignment that satisfies the hard block of clauses and maximizes the number of satisfied soft blocks. Dealing with blocks of clauses allows to define soft constraints without introducing auxiliary variables, and has positive consequences in the solvers because we can define solving techniques which are local to each block, and make an earlier application of inference rules in which the premises are required to be short.

Secondly, we describe the Soft-SAT solvers we have designed and implemented: Soft-SAT-S and Soft-SAT-D. They are original Soft-SAT branch and bound solvers equipped with original lazy data structures, powerful inference techniques, good quality lower bounds, and original variable selection heuristics. The heuristics of Soft-SAT-S are static while the heuristics of Soft-SAT-D are dynamic. The experimental investigation performed on a representative sample of instances provides empirical evidence that Soft-SAT is a very competitive generic problem approach, compared with the state-of-the-art approaches developed in the CSP and SAT communities.

Thirdly, we present novel solving techniques for Partial Max-SAT, which is a suitable formalism for encoding and solving combinatorial optimization problems, and that has become a standard in the community. The solving techniques we have developed include new sound inference rules, lower bound computation methods based on unit propagation, clause learning and non-chronological backtracking derived from the analysis of both hard and soft conflicts, and preprocessing techniques based on variable saturation, randomization and restarts.

Finally, we describe the Partial Max-SAT solvers we have designed and implemented: PMS and W-MaxSatz. PMS was implemented from scratch, and

Copia gratuita. Personal free copy http://libros.csic.es

its most important feature is the learning module for hard and soft conflicts. W-MaxSatz was implemented on top of the state-of-the-art Max-SAT solver MaxSatz, and its most important feature is the method for computing lower bounds, which includes the computation of underestimations using unit propagation enhanced with failed literal detection, and the application of sound inference rules. W-MaxSatz also incorporates a hard learning module. The experimental comparison between PMS, W-MaxSatz, and the best performing state-of-the-art Partial Max-SAT solvers, as well as the results of the 2007 Max-SAT Evaluation, provide empirical evidence that both solvers are robust and very competitive.

© CSIC © del autor o autores / Todos los derechos reservados

Acknowledgments

First of all, I would like to thank my supervisor Felip Manyà for his experience, generosity and his infinite patience. He gave me the opportunity to grow as researcher and I always will be grateful with him. Without him, this work would not have been possible.

Many thanks to my office mates: Jordi, Carlos and Paula; the remainder members of the Artificial Intelligence Research Group: Alba, Carles, Carlos, César, Magda, Ramón, Tere,. . . ; the secretaries of the department: Àngels and Dèlia; and the people that helped me in my stays in Amiens and Lisbon: Chu Min, Gilles, Inês, Laure, Sidney, Sylvain, Vasco,. . . . Many thanks to all of them with whom I shared very good moments.

Thanks to the *Ministerio de Educación y Ciencia* and the *Universitat de Lleida* for the funding provided and their technical support. This research has been partially supported by MEC research projects TIC2003-00950, TIN2004-07933-C03-03, TIN2006-15662-C02-02, TIN2007-68005-C04-04, CONSOLIDER CSD2007-0032, INGENIO 2010, and Acción Integrada HP2005-0147.

Finally, I would like to thank my family, friends and Àngels for their support, frienship and understanding.

Copia gratuita. Personal free copy http://libros.csic.es

Chapter 1

Introduction

1.1 Introduction

Since the introduction of Computer Science in our society, computer scientists have tried to model real-life problems and solve them using the power of the computer systems. Nevertheless, many problems are known to be *computationally intractable*, in the sense that, unless P=NP, the algorithms for solving them require an exponential number of steps in the length of the input for some of their instances. In this case, the challenge for computer scientists is to devise algorithms that solve as many instances of intractable problems as possible in a reasonable amount of time.

Among the intractable problems, the Boolean satisfiability problem (SAT) has been considered a central problem in Artificial Intelligence, Electronic Design Automation and Theoretical Computer Science, and nowadays it is commonly acknowledged that solving combinatorial *decision* problems via their reduction to SAT is one of the best performing problem solving approaches. SAT has shown to be competitive in a variety of domains, including hardware verification [eSSMS99, MMZ$^+$01, VB01, BK02, KSHK07], bioinformatics [LMS06b, LMS06a], planning [KS96, Kau06], and scheduling [BM00, ZLS04].

In this thesis we investigate different Max-SAT problems, which are optimization versions of SAT that, despite not being so well-studied as SAT, have seen an increasing activity in the community working on satisfiability problems [LM09]. Even an evaluation of Max-SAT solvers is organized since 2006 as a co-located event of the International Conference on Theory and Applications of Satisfiability Testing.

Our research program aims at converting Max-SAT formalisms into a competitive generic problem solving approach for solving combinatorial *optimization* problems, and in particular, converting them into competitive approaches for solving over-constrained problems with soft and hard constraints. To this end, in this thesis, we study Max-SAT formalisms that incorporate the notion of partiality, and design and implement solving techniques for such formalisms. Our

Copia gratuita. Personal free copy http://libros.csic.es

empirical investigation provides evidence that the solvers that we have developed exhibit a good performance profile on a wide collection of benchmarks.

The structure of this chapter is as follows. In Section 1.2, we present basic definitions of SAT and Max-SAT. In Section 1.3, we explain the motivation of our work. In Section 1.4, we describe the objectives of our research. In Section 1.5, we describe the main contributions of the thesis. In Section 1.6, we enumerate the publications we have made during the course of the thesis. In Section 1.7, we present an overview of the remaining chapters.

1.2 SAT and Max-SAT problems

In propositional logic, a variable x_i may take values 0 (for false) or 1 (for true). A *literal* ℓ_i is a variable x_i or its negation $\neg x_i$. A *clause* is a disjunction of literals, and a CNF formula is a conjunction of clauses. A *weighted clause* is a pair (C_i, w_i), where C_i is a disjunction of literals and w_i, its weight, is a positive number, and a *weighted CNF formula* is a conjunction of weighted clauses. A (weighted) *CNF formula* is often represented as a set of clauses.

An *assignment* of truth values to the propositional variables satisfies a literal x_i if x_i takes the value 1 and satisfies a literal $\neg x_i$ if x_i takes the value 0, satisfies a clause if it satisfies at least one literal of the clause, and satisfies a CNF formula if it satisfies all the clauses of the formula. A CNF formula is *satisfiable* if there exists an assignment that satisfies the formula; otherwise, it is *unsatisfiable*.

The *SAT problem* for a CNF formula ϕ is the problem of deciding whether there exists a satisfying assignment for ϕ. The *Max-SAT problem* for a CNF formula ϕ is the problem of finding an assignment of values to propositional variables that maximizes the number of satisfied clauses. In this sequel we often use the term Max-SAT meaning Min-UNSAT. This is because, with respect to exact computations, finding an assignment that minimizes the number of unsatisfied clauses is equivalent to finding an assignment that maximizes the number of satisfied clauses.

Two SAT instances are *equivalent* if they are satisfied by the same set of assignments. In Max-SAT, two instances ϕ_1 and ϕ_2 are *equivalent* if ϕ_1 and ϕ_2 have the same number of unsatisfied clauses for every complete assignment of ϕ_1 and ϕ_2.

We will also consider three extensions of Max-SAT which are more well-suited for representing and solving over-constrained problems: weighted Max-SAT, Partial Max-SAT and weighted Partial Max-SAT.

The *weighted Max-SAT* problem for a weighted CNF formula ϕ is the problem of finding an assignment of values to propositional variables that maximizes the sum of weights of satisfied clauses (or equivalently, that minimizes the sum of weights of unsatisfied clauses).

A *Partial Max-SAT* instance is a CNF formula in which some clauses are *relaxable* or *soft* and the rest are *non-relaxable* or *hard*. The *Partial Max-SAT problem* for a Partial Max-SAT instance ϕ is the problem of finding an assignment that satisfies all the hard clauses and the maximum number of soft clauses.

© CSIC © del autor o autores / Todos los derechos reservados

The *weighted Partial Max-SAT* problem is the combination of weighted Max-SAT and Partial Max-SAT.

1.3 Motivation

We started our research on Max-SAT formalisms with hard and soft constraints in 2003. At that time, SAT was —as it is nowadays— a central topic in Artificial Intelligence, Electronic Design Automation and Theoretical Computer Science. There were publicly available complete solvers such as Chaff [MMZ$^+$01], GRASP [MSS99], MiniSat [ES03], Posit [Fre95], Relsat [BS97], and Satz [LA97a, LA97b], as well as local search solvers such as GSAT and WalkSAT [SK93, SKC94, SLM92]. There was also enough empirical evidence about the merits of the generic problem solving approach which consists in modeling NP-complete decision problems as SAT instances, solving the resulting encodings with a state-of-the-art SAT solver, and mapping the solution back into the original problem.

Despite the remarkable activity on SAT, there was a reduced number of papers dealing with the design and implementation of exact Max-SAT solvers; solving NP-hard problems by reducing them to Max-SAT was not considered as a suitable alternative for solving optimization problems; and the activity on Max-SAT was basically concentrated on theoretical results. This is in contrast with what happened in the Constraint Programming community, where the Weighted Constraint Satisfaction Problem (Weighted CSP) was a problem attracting the interest of that community, which published a considerable amount of results about weighted CSP and consolidated a research line on soft constraints [MRS06].

The most remarkable implementations of exact Max-SAT solvers were the branch and bound solvers developed by Wallace and Freuder [WF96], and Borchers and Furman [BF99]. These solvers can be seen as an adaptation to Max-SAT of the Davis-Logemann-Loveland (DLL) procedure [DLL62], and were the starting point for developing some of the most successful modern Max-SAT solvers.

An approach for producing good performing Max-SAT solvers was based on adapting to Max-SAT technology that was proven to be successful in DLL-style SAT solvers such as optimized data structures, clever variable selection heuristics, clause learning, and non-chronological backtracking. Another approach was to improve the quality of the lower bounds in branch and bound Max-SAT solvers by incorporating powerful inference rules that preserve the number of unsatisfied clauses and that, in the best case, make explicit some contradictions; and by incorporating new ways of computing underestimations of the number of unsatisfied clauses that become unsatisfied if the partial assignment associated to a node of the search space is extended to a complete assignment.

The experience on SAT-based problem solving has shown that both the solver and the encoding are important for solving efficiently combinatorial problems. In contrast with other parallel investigations whose main focus were the solvers, *our initial motivation was to investigate Max-SAT formalisms that produce natural and compact encodings of combinatorial optimization problems, and equip them*

Copia gratuita. Personal free copy http://libros.csic.es

with robust solvers that exploit structural properties of the encodings.

All our work is around the notion of *partiality* in Max-SAT. Partiality amounts to have clauses which are mandatory and clauses (or sets of clauses) which are relaxable. On the one hand, this notion captures the constraints of real problems in a more natural way, and produces more compact encodings. On the other hand, the distinction between mandatory and relaxable clauses has a significant impact on the solving techniques that can be applied in branch and bound solvers. In a sense, we could say that the notion of partiality allows to define formalisms between SAT and Max-SAT for effectively solving combinatorial optimization problems.

Finally, we would like to point out that our research has benefited a lot from the 2006 and 2007 editions of the Max-SAT Evaluation. They allowed to compare our solvers with the most representative state-of-the-art solvers, and make publicly available a good collection of benchmarks for testing our solvers.

1.4 Objectives

The general objective of our research is to study Max-SAT formalisms that incorporate the notion of partiality, and design and implement solving techniques for such formalisms that exhibit a good performance profile on a wide collection of benchmarks. The final goal is to show that Max-SAT formalisms can become a competitive generic problem solving approach for solving over-constrained problems.

The concrete objectives to achieve in the thesis can be summarized as follows:

- Define a new formalism for solving over-constrained problems, with hard and soft constraints, that deals with blocks of clauses rather than individual clauses in order to produce more compact and natural encodings. Equip such a formalism with exact solvers that incorporate good performing solving techniques, optimized data structures, and heuristics that exploit structural properties of the encodings.

- Define new inference rules and learning schemes for the Partial Max-SAT formalism, and design and implement exact Partial Max-SAT solvers that incorporate them. We plan to learn clauses from the conflicts produced when a hard clause is violated during the exploration of the search space (hard conflict), as well as when a soft clause is violated (soft conflict).

- Design and implement a preprocessor for Partial Max-SAT that applies solving techniques which, despite being too costly for being applied to each node of the search space, can produce gains if they are applied before starting the search. To assess the impact of the preprocessor, we plan to conduct an empirical evaluation using the most representative state-of-the-art Partial Max-SAT solvers and the instances of the last Max-SAT evaluation.

© CSIC © del autor o autores / Todos los derechos reservados

- Conduct an empirical evaluation of the techniques incorporated into the solvers we plan to develop in this thesis, and in particular of the techniques that take into account the distinction between hard and soft clauses. Identifying their strengths and weaknesses should allow us to gain new insights for developing more powerful solving techniques.

- Conduct an empirical comparison between the solvers of the thesis and the best performing state-of-the-art Partial Max-SAT solvers. Knowing the performance profile of other solvers can help improve the performance of our solvers. Moreover, we plan to make the solvers publicly available and actively participate in the Max-SAT evaluations.

1.5 Contributions

The main contributions of the thesis can be summarized as follows:

- We defined the Soft-SAT formalism, which allows to encode over-constrained problems in a natural and compact way. Soft-SAT encodes constraints as blocks of clauses without needing to introduce auxiliary variables for dealing with soft constraints. This has positive consequences in the solvers because we can define solving techniques which are local to each block, and apply inference rules, in which the premises are short clauses, earlier than in other formalisms that need to use auxiliary variables in order to ensure that there is exactly one clause for each violated constraint (cf. Example 4.5). Moreover, we developed Soft-SAT solvers with branching heuristics and underestimation techniques that take into account the structure of the domains in the original problem by exploiting information which is hidden in Boolean encodings.

- We extended, to Partial Max-SAT, existing solving techniques for SAT and Max-SAT. Such techniques include variable selection heuristics that take into account the size of the clause in which the variable appears, lower bound computation methods based on unit propagation, failed literal detection to improve the lower bound, and local search solvers to obtain a good initial upper bound.

- We defined new inference rules for Partial Max-SAT. These rules are proven to be sound, can be applied efficiently, and can be seen as unit resolution refinements.

- We incorporated the 1-UIP learning schema [MMZ$^+$01] to our solvers in order to analyze the conflicts detected in hard clauses. We learn a clause from each conflict and backtrack non-chronologically. To the best of our knowledge, it was the first time that learning was incorporated into a branch and bound Partial Max-SAT solver.

- We defined new soft learning techniques that are applied every time we reach a soft conflict.

Copia gratuita. Personal free copy http://libros.csic.es

- We designed and implemented two Soft-SAT solvers:
  - **Soft-SAT-S:** It was the first solver developed for the Soft-SAT formalism. It uses a static variable selection heuristic, extremely efficient lazy data structures, and an underestimation based on inconsistency counts.
  - **Soft-SAT-D:** It was the second solver developed for the Soft-SAT formalism. It uses a dynamic variable selection heuristic with n-ary branching for Soft-SAT encoded CSP instances, lazy data structures based on two-watched literals, and an underestimation based on inconsistency counts.
- We designed and implemented two Partial Max-SAT solvers:
  - **PMS:** It was the first branch and bound solver that we developed for the Partial Max-SAT formalism. It is an implementation from scratch, and the most important feature of this solver is the learning module for hard and soft constraints. PMS also incorporates advanced techniques for bounds computation and simple inference rules.
  - **W-MaxSatz:** It was the second solver that we developed for the Partial Max-SAT formalism. It is build on top of the Max-SAT solver MaxSatz, and the most important feature of this solver is the advanced techniques used for the lower bound computation, which include the computation of underestimation with unit propagation enhanced with failed literal detection and the application of sound inference rules. W-MaxSatz also incorporates a hard learning module.
- We designed and implemented several preprocessors for Partial Max-SAT instances. The most important are:
  - **Variable saturation:** This preprocessing saturates the formula w.r.t. a limited number of variables. It helps reduce the search space by removing variables from the initial formula.
  - **Learning and restarts:** This preprocess adds to the initial formula a set of redundant clauses from several search spaces.
- We conducted an empirical evaluation of the learning techniques and new inference rules developed in this thesis. We observed that hard learning is an important feature for Partial Max-SAT solvers, soft learning can improve the results in some sets of instances, and in combination with our inference rules, gives rise to the best performance profile.
- We conducted an empirical evaluation between our solvers and the best performing state-of-the-art Partial Max-SAT solvers. We observed that our Soft-SAT solvers are competitive against weighted CSP solvers and state-of-the-art Partial Max-SAT solvers. PMS has a good general performance,

© CSIC © del autor o autores / Todos los derechos reservados

and W-MaxSatz is competitive on several types of instances, specially on random instances.

- We conducted an empirical evaluation of the preprocessors developed in this thesis using both our solvers and the most representative state-of-the-art Partial Max-SAT solvers. We observed that our preprocessors can reduce the CPU time needed to solve several types of instances.

1.6 Publications

Some of the results presented in this thesis have already been published in journals and conference proceedings. The articles are chronologically listed and classified according to the main topics of the thesis, Soft-SAT and Partial Max-SAT:

Soft-SAT

– Josep Argelich and Felip Manyà. Solving Over-Constrained Problems with Max-SAT Algorithms. In *Workshop on Modelling and Solving Problems with Constraints, 16th European Conference on Artificial Intelligence, ECAI-2004, Valencia, Spain*, pages 116–124, Workshop Proceedings, 2004.

– Josep Argelich and Felip Manyà. An Exact Max-SAT Solver for Over-Constrained Problems. In *6th International Workshop on Preferences and Soft Constraints, 10th International Conference on Principles and Practice of Constraint Programming, CP-2004, Toronto, Canada*, pages 1–11, Workshop Proceedings, 2004.

– Josep Argelich. Solving Over-Constrained Problems with SAT. In *11th International Conference on Principles and Practice of Constraint Programming, CP-2005, Sitges, Spain*, page 838, Springer LNCS 3709, 2005.

– Josep Argelich and Felip Manyà. Solving Over-Constrained Problems with SAT Technology. In *8th International Conference on Theory and Applications of Satisfiability Testing, SAT-2005, St. Andrews, Scotland*, pages 1–15, Springer LNCS 3569, 2005.

– Josep Argelich and Felip Manyà. Exact Max-SAT Solvers for Over-Constrained Problems. *Journal of Heuristics*, 12(4-5):375-392, 2006.

Partial Max-SAT

– Josep Argelich and Felip Manyà. Learning Hard Constraints in Max-SAT. In *11th Annual ERCIM Workshop on Constraint Solving and Constraint Programming, CSCLP-2006, Caparica, Portugal*, pages 5–12, Workshop Proceedings, 2006.

Copia gratuita. Personal free copy http://libros.csic.es

– Josep Argelich and Felip Manyà. Partial Max-SAT Solvers with Clause Learning. In *10th International Conference on Theory and Applications of Satisfiability Testing, SAT-2007, Lisbon, Portugal*, pages 28–40, Springer LNCS 4501, 2007.

– Josep Argelich, Chu Min Li and Felip Manyà. An Improved Exact Solver for Partial Max-SAT. In *The International Conference on Nonconvex Programming: Local and Global Approaches, NCP-2007, Rouen, France*, pages 230–231, Conference Proceedings, 2007

Preprocessing techniques

– Josep Argelich, Chu Min Li and Felip Manyà. A Preprocessor for Max-SAT Solvers. In *11th International Conference on Theory and Applications of Satisfiability Testing, SAT-2008, Guangzhou, P. R. China*, pages 15–20, Springer LNCS 4996, 2008.

Encodings

– Josep Argelich, Alba Cabiscol, Inês Lynce and Felip Manyà. Encoding Max-CSP into Partial Max-SAT. In *38th International Symposium on Multiple-Valued Logic, ISMVL-2008, Dallas, Texas*, pages 106–111, IEEE CS Press, 2008.

– Josep Argelich, Alba Cabiscol, Inês Lynce and Felip Manyà. Modelling Max-CSP as Partial Max-SAT. In *11th International Conference on Theory and Applications of Satisfiability Testing, SAT-2008, Guangzhou, P. R. China*, pages 1–14, Springer LNCS 4996, 2008.

Miscellaneous

– Josep Argelich, Xavier Domingo, Felip Manyà and Jordi Planes. Towards Solving Many-Valued Max-SAT. In *36th International Symposium on Multiple-Valued Logic, ISMVL-2006, Singapore*, paper 26, IEEE CS Press, 2006.

– Josep Argelich, Chu Min Li, Felip Manyà and Jordi Planes. The First and Second Max-SAT Evaluations. *Journal on Satisfiability*, submitted for second review, 2008.

1.7 Overview

This section provides an overview of the thesis. We briefly describe the contents of each of the remaining chapters:

Chapter 2: SAT algorithms. We present an overview of the most relevant methods for solving SAT. First, we define some basic concepts in SAT. Second, we present the resolution method, which applies an inference rule that provides a refutation complete inference system. Third, we describe

© CSIC © del autor o autores / Todos los derechos reservados

DP, the first effective method for producing resolution refutations. Fourth, we present the DLL procedure, implemented in the majority of state-of-the-art complete SAT algorithms, and review the main solving techniques that have been incorporated into DLL in order to devise fast SAT solvers. Finally, we give the basis of the current state-of-the-art local search algorithms for SAT.

Chapter 3: Max-SAT algorithms. We introduce some background knowledge about Max-SAT, and review the solving techniques that have proved to be useful in terms of performance. First, we define some basic concepts in Max-SAT and Max-CSP. Second, we present the branch and bound schema, which is the most commonly used approach to exact Max-SAT solving. Third, we define a complete resolution rule for Max-SAT. Fourth, we review the main Max-SAT approximation algorithms. Fifth, we describe the solving techniques that have been defined for dealing with hard and soft constraints under the formalism of Partial Max-SAT. Finally, we present the 2006 and 2007 Max-SAT Evaluations.

Chapter 4: The Soft-SAT formalism. We present a new generic problem solving approach for over-constrained problems based on Max-SAT. We first define a Boolean clausal form formalism that deals with blocks of clauses instead of individual clauses, and that allows one to declare a block of hard clauses and several blocks of soft clauses. 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 the hard block of clauses and maximizes the number of satisfied soft blocks. Our solvers are branch and bound algorithms equipped with original lazy data structures, powerful inference techniques, good quality lower bounds, and original variable selection heuristics. Finally, we report an experimental investigation on a representative sample of instances which provides experimental evidence that our approach is competitive with the state-of-the-art approaches developed in the CSP and SAT communities.

Chapter 5: The Partial Max-SAT formalism. We focus on Partial Max-SAT, which is a problem between SAT and Max-SAT which is well-suited for representing and solving over-constrained problems, and has become a standard in the recent years. First, we present an overview of the Partial Max-SAT problem. Second, we define novel techniques for Partial Max-SAT solving, and introduce the solving techniques that incorporate the modern Partial Max-SAT solvers. Third, we present some efficient and original preprocessing techniques for Partial Max-SAT. Next, we describe the two Partial Max-SAT solvers we have designed and implemented. Finally, we report on an experimental investigation that we conducted in order to assess the performance of our solvers and preprocessing techniques. The experimental results indicate that our solvers are among the best state-of-the-art Partial Max-SAT solvers.

Copia gratuita. Personal free copy http://libros.csic.es

Chapter 6: Conclusions. We briefly summarize the main contributions of the thesis, and point out some open problems and future research directions that we plan to tackle in the near future.

© CSIC © del autor o autores / Todos los derechos reservados

Chapter 2

SAT algorithms

In this chapter we present an overview of the most relevant methods for solving SAT. Section 2.1 defines some basic concepts in SAT. Section 2.2 presents the resolution method, which applies an inference rule that provides a refutation complete inference system. Section 2.3 describes DP, the first effective method for producing resolution refutations. Section 2.4 presents the DLL procedure, implemented in the majority of state-of-the-art complete SAT algorithms, and reviews the main solving techniques that have been incorporated into DLL in order to devise fast SAT solvers. Finally, Section 2.5 gives the basis of the current state-of-the-art local search algorithms for SAT.

2.1 Basic concepts in SAT

In propositional logic, a variable x_i may take values 0 (for false) or 1 (for true). A *literal* ℓ_i is a variable x_i or its negation $\neg x_i$. The *complementary* of a literal ℓ, denoted by $\bar{\ell}$, is x if $\ell = \neg x$ and is $\neg x$ if $\ell = x$. A *clause* is a disjunction of literals, and a CNF formula is a conjunction of clauses. A *CNF formula* is often represented as a set of clauses. The *length* of a clause is the total number of literal occurrences in the clause. A clause with one literal is called *unit*, with two literals is called *binary*, and with three literals is called *ternary*. The *size* of CNF formula ϕ, denoted by $|\phi|$, is the sum of the lengths of all its clauses.

An *assignment* of truth values to the propositional variables satisfies a literal x_i if x_i takes the value 1 and satisfies a literal $\neg x_i$ if x_i takes the value 0, satisfies a clause if it satisfies at least one literal of the clause, and satisfies a CNF formula if it satisfies all the clauses of the formula. A CNF formula is *satisfiable* if there exists an assignment that satisfies the formula; otherwise, it is *unsatisfiable*. An *empty clause*, denoted by $\Box$, contains no literals and cannot be satisfied. A *tautology* is a CNF formula that is satisfied by any truth assignment. Two SAT instances are *equivalent* if they are satisfied by the same set of assignments.

An assignment for a CNF formula ϕ is *complete* if all the variables occurring in ϕ have been assigned; otherwise, it is *partial*. A partial truth assignment also

Copia gratuita. Personal free copy http://libros.csic.es

partitions the clauses of a CNF formula into three sets: *satisfied* clauses, the clauses that contain at least one satisfied literal; *unsatisfied* clauses, the clauses in which all its literals are unsatisfied, and *unresolved* clauses, the clauses that the partial assignment makes them not to be decided. The unassigned literals of a clause are referred to as its *free literals.* In a search context, an unresolved clause is said to be *unit* if the number of its free literals is one. Similarly, an unresolved clause with two free literals is said to be *binary*, and an unresolved clause with three free literals is said to be *ternary.*

The *SAT problem* for a CNF formula ϕ is the problem of deciding whether there exists a satisfying assignment for ϕ.

Example 2.1 *Let us consider a CNF formula ϕ having three clauses c_1, c_2 and c_3:*

$$\begin{array}{lcl} c_1 & : & (x_1 \vee x_2) \\ c_2 & : & (x_2 \vee \neg x_3) \\ c_3 & : & (x_1 \vee x_2 \vee x_3) \end{array}$$

Suppose that the current truth assignment is $\mathcal{A} : \{x_1 = false, x_3 = false\}$. This implies having clauses c_1 and c_3 unresolved and clause c_2 satisfied. Observe that clauses c_1 and c_3 are also unit because x_2 is the only free literal. Hence, the CNF formula is unresolved.

Suppose that this assignment is extended with $x_2 = false$; i.e., $\mathcal{A}' : \{x_1 = false, x_2 = false, x_3 = false\}$. Then, clause c_3 becomes unsatisfied. This means that the search reached an empty clause; i.e., the CNF formula is unsatisfied by $\mathcal{A}'$, $\mathcal{A}'(\phi) = false$. Also, suppose that in the subsequent search we have the assignment $\mathcal{A}'' : \{x_1 = true, x_3 = false\}$. Clearly, all the clauses get satisfied and, therefore, the CNF formula is satisfiable, $\mathcal{A}''(\phi) = true$.

2.2 Resolution

One of the techniques that allows to solve SAT automatically is resolution [Rob65]. Resolution is an inference rule that provides a complete inference system by refutation. Given two clauses c_1, c_2, called *parent clauses*, then r is a *resolvent* of c_1 and c_2 if there is one literal $\ell \in c_1$ such that $\bar{\ell} \in c_2$, and r has the form

$$r = (c_1 \setminus \{\ell\}) \cup (c_2 \setminus \{\bar{\ell}\}).$$

The resolution step for a CNF formula ϕ, denoted by $Res(\phi)$, is defined as follows:

$$Res(\phi) = \phi \cup \{r \mid r \text{ is a resolvent of two clauses in } \phi\}$$

A resolution procedure consists in computing resolution steps to a formula ϕ until $Res(\phi) = \phi$; i.e., no more resolvents can be derived. Then, the formula is unsatisfiable if $\Box \in \phi$; otherwise, ϕ is satisfiable. Algorithm 2.1 describes this procedure [Sch89].

© CSIC © del autor o autores / Todos los derechos reservados

Algorithm 2.1: `Resolution`(ϕ) : Resolution based SAT algorithm

```
Output: Satisfiability of $\phi$
Function Resolution( $\phi$ : CNF formula ) : Boolean
    repeat
        $\phi' \leftarrow \phi$
        $\phi \leftarrow Res(\phi)$
    until $\square \in \phi \vee \phi = \phi'$
    if $\square \in \phi$ then return false
    else return true
```

2.3 The Davis-Putnam procedure

The first effective method for producing resolution refutations was the Davis-Putnam procedure (DP) [DP60]. DP is based on iteratively simplifying the formula until the empty clause is generated or until the formula is empty. It consists of three rules:

1. `Unit Propagation` (UP), also referred to as *Boolean constraint propagation* [ZM88], is the iterated application of the *Unit Clause* (UC) *rule* (also referred to as the *one-literal rule*) until an empty clause is derived or there are no unit clauses left. If a clause is unit, then its literal must be assigned value *true*. If $\{\ell\}$ is a unit clause of a CNF formula ϕ, UC consists in deleting all the clauses of ϕ with literal ℓ and removing all the occurrences of literal $\bar{\ell}$.

2. `Pure literal rule` (also referred to as *monotone literal rule*). A literal is pure if its complementary literal does not occur in the CNF formula. The satisfiability of a CNF formula is unaffected by satisfying its pure literals. Therefore, all clauses containing a pure literal can be removed.

3. `Resolution` is applied in order to iteratively eliminate each variable from the CNF formula. In order to do so, DP applies a refinement (a restriction) of the resolution method, known as *variable elimination*: Let $\mathcal{C}_\ell$ be the set of clauses containing ℓ and $\mathcal{C}_{\bar{\ell}}$ the set of clauses containing $\bar{\ell}$, the method consists in generating all the non-tautological resolvents using all clauses in $\mathcal{C}_\ell$ and all clauses in $\mathcal{C}_{\bar{\ell}}$, and then removing all clauses in $\mathcal{C}_\ell \cup \mathcal{C}_{\bar{\ell}}$. After this step, the CNF formula contains neither ℓ nor $\bar{\ell}$.

The pseudo-code of DP is given in Algorithm 2.2. The algorithm selects a variable to be eliminated among the shortest clauses. Each time a variable is eliminated, the number of clauses in the CNF formula may grow quadratically in the worst case. Therefore, the worst case memory requirement for DP is exponential. In practice, DP can only handle SAT instances with tens of variables because of this memory explosion problem [Urq87, CS00]. The procedure stops applying resolution when the CNF formula is found to be either satisfiable or

Copia gratuita. Personal free copy http://libros.csic.es

Algorithm 2.2: DavisPutnam(ϕ) : Davis-Putnam procedure for SAT

```
Output: Satisfiability of $\phi$
Function DavisPutnam($\phi$ : CNF formula) : Boolean
    UnitPropagation($\phi$)
    PureLiteralRule($\phi$)
    if $\phi = \varnothing$ then return true
    if $\Box \in \phi$ then return false
    $\ell \leftarrow$ literal in $c \in \phi$ having $c$ the minimum length
    $\mathcal{R}_\ell \leftarrow$ all possible non-tautological resolvent clauses between all
    clauses in $\mathcal{C}_\ell$ and all clauses in $\mathcal{C}_{\bar{\ell}}$
    return DavisPutnam($\phi \cup \mathcal{R}_\ell \setminus (\mathcal{C}_\ell \cup \mathcal{C}_{\bar{\ell}})$)
```

unsatisfiable. It is declared to be unsatisfiable whenever a conflict is reached, detected while applying rule UC. If no conflict is reached, the CNF formula becomes empty and is declared to be satisfiable.

Example 2.2 *Given the following CNF formula, we demonstrate its satisfiability using algorithm DP:*

$$(x_1), (x_1 \vee x_2), (x_2 \vee x_4), (\neg x_1 \vee x_3 \vee \neg x_4), (x_3 \vee x_5), (\neg x_1 \vee \neg x_3 \vee \neg x_5)$$

We show the steps applied by algorithm DP using a table. In the first column the input formula is displayed, where each line represents a different clause. The rest of the columns represent the result of applying UC. The table below shows the application of the rule to literal x_1. Removed clauses are marked with a '$\times$' and modified clauses are displayed in bold face.

ϕ	x_1
(x_1)	$\times$
$(x_1 \vee x_2)$	$\times$
$(x_2 \vee x_4)$	$(x_2 \vee x_4)$
$(\neg x_1 \vee x_3 \vee \neg x_4)$	$(\mathbf{x_3} \vee \neg \mathbf{x_4})$
$(x_3 \vee x_5)$	$(x_3 \vee x_5)$
$(\neg x_1 \vee \neg x_3 \vee \neg x_5)$	$(\neg \mathbf{x_3} \vee \neg \mathbf{x_5})$

In a second step, DP applies the pure literal rule. The table below shows the application of the rule to literal x_2 and then to literal $\neg x_4$.

ϕ'	x_2	$\neg x_4$
$(x_2 \vee x_4)$	$\times$	
$(x_3 \vee \neg x_4)$	$(x_3 \vee \neg x_4)$	$\times$
$(x_3 \vee x_5)$	$(x_3 \vee x_5)$	$(x_3 \vee x_5)$
$(\neg x_3 \vee \neg x_5)$	$(\neg x_3 \vee \neg x_5)$	$(\neg x_3 \vee \neg x_5)$

Finally, DP applies resolution. The table below shows the elimination of variable x_3 by resolution. Observe that a tautological clause appears, and is removed

© CSIC © del autor o autores / Todos los derechos reservados

by the method.

ϕ''	x_3	
$(x_3 \vee x_5)$	$\times$	
$(\neg x_3 \vee \neg x_5)$	$(x_5 \vee \neg x_5)$	$\times$

At the end, the CNF formula becomes empty. Thus, the original CNF formula is satisfiable.

2.4 The Davis-Logemann-Loveland procedure

The vast majority of state-of-the-art complete SAT algorithms are built upon the backtrack search algorithm of Davis, Logemann and Loveland (DLL) [DLL62]. DLL replaces the application of resolution in DP by the splitting of the CNF formula into two subproblems. Given a literal ℓ occurring in ϕ, the first subproblem ($\phi_{\bar{\ell}}$) is the application of UC over ϕ with $\bar{\ell}$, and the second subproblem (ϕ_ℓ) is the application of UC over ϕ with ℓ. Then, ϕ is unsatisfiable if and only if ϕ_ℓ and $\phi_{\bar{\ell}}$ are unsatisfiable. This method is shown in Algorithm 2.3.

Procedure DLL essentially constructs a binary search tree in a depth-first manner (e.g., Figure 2.1). The leaf nodes of the search tree represent complete assignments (i.e., all variables are assigned) while internal nodes represent partial assignments (i.e., some variables are assigned, the rest are free). The DLL procedure explores the search tree and determines whether there exists an assignment that satisfies the input formula.

DLL incorporates `Unit Propagation` and `Pure Literal Rule` in order to avoid the explicit exponential enumeration of all the leaves of the search tree. Using a variable selection heuristic, the branching variables are selected to reach a dead-end as early as possible.

Example 2.3 *The search tree for the CNF formula below is displayed in Figure 2.1.*

$$(x_1 \vee x_5) \wedge (x_1 \vee \neg x_6) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee \neg x_4) \wedge$$
$$(\neg x_2 \vee \neg x_4 \vee \neg x_5) \wedge (x_2 \vee x_4 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_2 \vee x_3 \vee x_6)$$

Solid lines are for splitting assignments, and dashed lines for unit propagation and monotone literal assignments. Black nodes mark reached conflicts.

The authors in [DLL62] identified three advantages of DLL over DP:

1. DP increases the number and length of the clauses rather quickly. DLL never increases the length of clauses.

2. Many redundant clauses may appear after resolution in DP, and seldom after splitting in DLL.

3. DLL often can yield new unit clauses, while DP not often will.

Copia gratuita. Personal free copy http://libros.csic.es

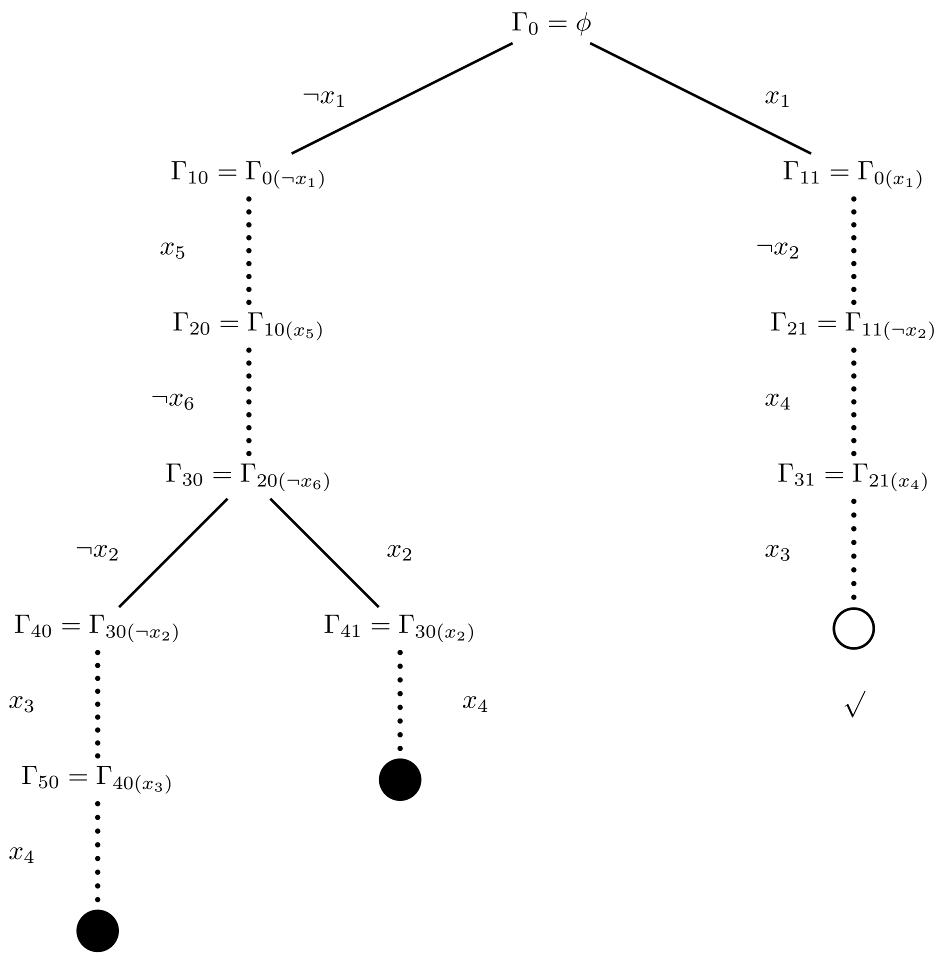

$\Gamma_0 = \phi$
$\neg x_1$
x_1
$\Gamma_{10} = \Gamma_{0(\neg x_1)}$
$\Gamma_{11} = \Gamma_{0(x_1)}$
x_5
$\neg x_2$
$\Gamma_{20} = \Gamma_{10(x_5)}$
$\Gamma_{21} = \Gamma_{11(\neg x_2)}$
$\neg x_6$
x_4
$\Gamma_{30} = \Gamma_{20(\neg x_6)}$
$\Gamma_{31} = \Gamma_{21(x_4)}$
$\neg x_2$
x_2
x_3
$\Gamma_{40} = \Gamma_{30(\neg x_2)}$
$\Gamma_{41} = \Gamma_{30(x_2)}$
x_3
x_4
$\surd$
$\Gamma_{50} = \Gamma_{40(x_3)}$
x_4

Figure 2.1: Search tree for DLL applied to Example 2.3.

2.4.1 Solving techniques for improving DLL

In this section we focus on important solving techniques that should be taken into account when developing SAT solvers that implement the DLL procedure: the variable selection heuristic, the data structures, clause learning, application of restarts, and reasoning on special structures. Our description of clause learning follows closely the presentation of [Zha03].

Among the most relevant complete algorithms developed in the last years based on the DLL procedure, we highlight the following ones:

BerkMin [GN01] by Evgueni Goldberg and Yakov Novikov.

eqsatz [Li03] by Chu Min Li.

© CSIC © del autor o autores / Todos los derechos reservados

Algorithm 2.3: `DavisLogemannLoveland`(ϕ) : DLL procedure for SAT

```
Output: Satisfiability of $\phi$
Function DavisLogemannLoveland($\phi$ : CNF formula) : Boolean
    UnitPropagation($\phi$)
    PureLiteralRule($\phi$)
    if $\phi = \varnothing$ then return true
    if $\square \in \phi$ then return false
    $\ell \leftarrow$ literal in $c \in \phi$ having $c$ the minimum length
    return (DavisLogemannLoveland($\phi_\ell$) $\vee$
    DavisLogemannLoveland($\phi_{\bar{\ell}}$))
```

GRASP [MSS99] by João Marques-Silva and Karem Sakallah.

MiniSat [ES03] by Niklas Eén and Niklas Sörensson.

POSIT [Fre95] by Jon William Freeman.

Relsat [BS97] by Roberto Bayardo and Robert Schrag.

SATO [ZS96] by Hantao Zhang and Mark Stickel.

Satz [LA97b] by Chu Min Li and Anbulagan.

siege [Rya04] by Lawrence Ryan.

zChaff [MMZ+01] by Matthew Moskewicz, Conor Madigan, Ying Zhao, Lintao Zhang and Sharad Malik.

Variable selection heuristics

The variable selection heuristic is decisive for finding as quick as possible a solution with the DLL procedure [MS99]. A bad heuristic can lead to explore the whole search tree, whereas a good heuristic allows to cut several branches, and even not to traverse more than a single branch in the best case.

The original variable selection heuristic in DLL selects a variable occurring in clauses of minimum size. The variable is selected after applying unit propagation and the pure literal rule, and is used to split the CNF formula into two subproblems.

Example 2.4 *Let ϕ be the following CNF formula:*

$$(\neg x_1 \vee x_2), (\neg x_2 \vee x_4 \vee \neg x_3), (x_1 \vee \neg x_5), (x_2 \vee x_4 \vee x_6), (\neg x_2 \vee x_4 \vee x_6), (x_2 \vee \neg x_3 \vee x_4)$$

The shortest clauses in ϕ are $(\neg x_1 \vee x_2)$ and $(x_1 \vee \neg x_5)$, hence the heuristic of DLL chooses any of the following variables: x_1, x_2 or x_5.

Copia gratuita. Personal free copy http://libros.csic.es

The MOMS (Maximum Occurrences in clauses of Minimum Size)[DABC93, Pre93] heuristic is an improvement of the previous heuristic. It selects a variable having the maximum number of occurrences in clauses of minimum size. Intuitively, these variables allow to well exploit the power of unit propagation and to augment the chances to reach an empty clause [Fre95].

Example 2.5 *Let ϕ be the CNF formula of Example 2.4. The shortest clauses in ϕ are $(\neg x_1 \vee x_2)$ and $(x_1 \vee \neg x_5)$, hence MOMS heuristic chooses variable x_1 because it occurs twice.*

The two-sided Jeroslow-Wang (JW) heuristic [JW90, HV95] is based on the same principle as MOMS. It gives priority to the variables that appear in the shortest clauses. In contrast to MOMS, the number of occurrences in the rest of clauses is also taken into account. The chances that a variable is selected by JW is inversely proportional to the size of the clauses in which it appears. JW uses a function J that takes as input a literal ℓ and returns a weight for such a literal.

$$J(\ell) = \sum_{\{c \in \phi | \ell \in c\}} 2^{-|c|},$$

where $|c|$ is the number of literals in clause c. JW chooses a variable x that maximizes $J(x) + J(\neg x)$.

Example 2.6 *Let ϕ be the CNF formula of Example 2.4. With function J, one can get: $J(x_1) = 0.25$, $J(\neg x_1) = 0.25$, $J(x_2) = 0.5$, $J(\neg x_2) = 0.25$, $J(x_3) = 0$, $J(\neg x_3) = 0.25$, $J(x_4) = 0.5$, $J(\neg x_4) = 0$, $J(x_5) = 0$, $J(\neg x_5) = 0.25$, $J(x_6) = 0.25$, $J(\neg x_6) = 0$. The variable x_2 with $J(x_2) + J(\neg x_2) = 0.75$ is chosen by heuristic JW.*

Several heuristics based on unit propagation have been proved useful and allow to exploit yet more the power of unit propagation; e.g., the heuristics of POSIT [Fre95], Tableau [CA96] and Satz [LA97a]. A unit propagation heuristic works as follows: Given a variable x, it examines x by respectively adding the unit clause x and $\neg x$ to a CNF formula, and independently computes two unit propagations. The real effect of the unit propagations is then used to weight x and detect failed literals. A *failed literal* is a literal whose addition to a CNF formula brings the empty clause after unit propagation.

Given a CNF formula ϕ, this heuristic requires propagating a literal x to count the clauses reduced in the subproblem obtained, and to follow the same process with the complementary literal $\neg x$. Let $w(x)$ and $w(\neg x)$ be the number of clauses reduced respectively by x and $\neg x$, this heuristic consist in choosing the variable which maximizes at the same time $w(x)$ and $w(\neg x)$, maximizing the following function: $F(x) = w(x) * w(\neg x) * 1024 + w(x) + w(\neg x)$. If literal x $(\neg x)$ is a failed literal, then $\neg x$ (x) is fixed. This approach makes possible to better prevent the consequences that the choice of the literal will produce.

Example 2.7 *Let ϕ be the CNF formula of Example 2.4. The application of function w to each literal produces the following values: $w(x_1) = 3$, $w(\neg x_1) = 1$,*

© CSIC © del autor o autores / Todos los derechos reservados

$w(x_2) = 2$, $w(\neg x_2) = 4$, $w(x_3) = 2$, $w(\neg x_3) = 0$, $w(x_4) = 0$, $w(\neg x_4) = 4$, $w(x_5) = 4$, $w(\neg x_5) = 0$, $w(x_6) = 0$, $w(\neg x_6) = 2$. The variable x_2 with $w(x_2) = 2, w(\neg x_2) = 4$ is chosen by a unit propagation based heuristic.

However, since examining a variable by two unit propagations is time consuming, two major problems remain open: should one examine all the free variable by unit propagation at every node of a search tree? Otherwise, what are the variables to be examined at a search tree node? In [LA97a], the authors try to experimentally address these two questions in order to obtain an optimal exploitation. They define a predicate $PROP_z$ at a search tree node whose meaning is the set of variables that will be examined at the node; i.e., variable x is examined if $PROP_z(x)$ is true.

$PROP_z$ is defined as follows: if there are more than T (parameter empirically set to 10) variables occurring both negatively and positively in binary clauses and having at least 4 binary occurrences, then only all these variables are examined; otherwise, if there are more than T variables occurring both negatively and positively in binary clauses and having at least 3 binary occurrences, then only all these variables are examined; otherwise all the free variables are examined.

Another approach consists in selecting a variable that is likely to be a backbone variable [DD01, KSTW05]. A variable is a backbone variable if the variable has assigned the same value in all the solutions. Given a CNF formula ϕ, this heuristics tries first on variables that belong (or are expected to belong) to the backbone of ϕ. If backbone variables are selected first, the algorithm searches through fewer branches, speeding up the solver. The heuristic of Satz and the heuristics based on the notion of backbone are quite effective on computationally difficult random SAT instances.

The previous heuristics were created without the addition of learning techniques into SAT solvers. The two following heuristics are thought for this kind of solvers, focusing on a kind of locality rather than focusing on formula simplification [Mit05]. For the solver zChaff [MMZ$^+$01, ZM02, Zha03], the authors proposed a branching heuristic called Variable State Independent Decaying Sum (VSIDS). This heuristic keeps a score for each literal. Initially, the scores are the number of occurrences of the literal in the initial problem. Because of the learning mechanism, clauses are added to the formula as the search progresses. VSIDS increases the score of a literal by a constant whenever an added clause contains the literal. More than to develop an accurate heuristic, the motivation was to design a fast and dynamically adapting heuristic.

The SAT solvers BerkMin [GN01] and siege [Rya04] have improved the VSIDS heuristic. BerkMin also measures the age of the clauses and the activity for deciding the next branching variable, whereas siege gives priority to assigning variables on recently recorded clauses.

Efficient data structures

The performance of the DLL procedure critically depends upon the care taken in the implementation. Solvers implementing DLL spent much of their time

Copia gratuita. Personal free copy http://libros.csic.es

applying unit propagation [Zha97, LMS02], and this has motivated the definition of several proposals to reduce the cost of applying unit propagation.

The simplest and intuitive implementation of unit propagation is to keep counters for each clause. This schema is attributed to Crawford and Auton [CA93] by [ZS96]. Similar schemas were since then employed in many solvers such as GRASP [MSS99], Relsat [BS97] and Satz [LA97a]. For example, in GRASP each clause keeps two counters, one for the satisfied literals in the clause and another for the unsatisfied literals in the clause. Each variable has two lists that contain all the clauses in which that variable appears with positive and negative polarity. When a variable is assigned a value, the counters of the clauses that contain this variable are updated. If a counter of unsatisfied literals becomes equal to the total number of literals in the clause, then the clause is conflicting. If a counter of unsatisfied literals is one less than the total number of literals in the clause and the counter of satisfied literals is null, then the clause is a unit clause. A counter-based unit propagation procedure is easy to understand and implement, but this schema may be improved.

As it is pointed out in [ZM02], Zhang and Stickel [ZS96], in order to speed up the application of unit propagation, created a new data structure in the solver SATO: head/tail lists. In this data structure, each clause has two pointers associated with it, called the head and the tail pointer. A clause stores all its literals in an array. Initially, the head pointer points to the first literal of the clause and the tail pointer points to the last literal of the clause. Each variable keeps four linked lists that contain pointers to clauses. Each of these lists contains the pointers to the clauses that have their head/tail literal in positive/negative polarity for a given variable. Whenever a variable is assigned, only two of the four lists are examined. The head/tail list schema is faster than the counter-based schema because when the variable is assigned the value true (false), the clauses that contain the variable with positive (negative) polarity are not visited. For both the counter-based algorithm and the head/tail list-based algorithm, undoing a variable assignment during backtrack has about the same computational complexity as assigning the variable.

The solver zChaff implements a unit propagation algorithm based on the so-called 2-literal watching schema. Similar to the head/tail list schema, 2-literal watching also has two special literals, called watched literals, for each clause. Each variable has two lists containing pointers to all the watched literals containing this variable in either polarity. In contrast to the head/tail list schema of SATO, there is no imposed order on the two pointers within a clause, and each of the pointers can move in either direction. In addition, no references have to be kept to the just assigned literals, since pointers do not move when backtracking is applied. This data structure was also used in the solvers BerkMin [GN01] and MiniSat [ES03].

The main problem of pointer-based data structures is that they cannot keep precise information about clauses with more than two free literals. The inclusion of additional literal references as a solution has been referred in [Gel02], and techniques to rearrange the list of literals have been investigated in [LMS05,

© CSIC © del autor o autores / Todos los derechos reservados

Nad02].

Clause learning and non-chronological backtracking

When a conflicting clause is found, a SAT solver finds out the reason of the conflict and tries to solve it by applying a *conflict analysis* procedure. This procedure tells the SAT solver that there exists no solution in the search space with the current partial assignment, and indicates a new search space to continue the search for a solution. The assigned variables are categorized as decision variables (i.e., picked using a variable selection heuristic) or propagation variables (i.e., assigned using unit propagation). The *decision level* of variable x is the number of decision variables in an assignment that were assigned before x. We call *conflict level* at the decision level at which a conflict occurs.

The simplest conflict analysis procedure is known as *chronological backtracking*, and is applied in the original DLL procedure. When a conflict is detected, the search backtracks to the most recent decision level with a variable that has not been flipped. Chronological backtracking has good performance on random instances and is used in SAT solvers like Satz [LA97a].

Modern SAT solvers apply a more sophisticated conflict analysis procedure, called *non-chronological backtracking* or *conflict direct backjumping*, that can get more information about the conflict. This procedure detects the reason of the conflict and often backtracks to a smaller decision level which produces the conflict. Example 2.8 shows the difference between chronological and non-chronological backtracking.

Example 2.8 *Let us consider a CNF formula ϕ with the following clauses among others:*

$$\begin{array}{ccc} c_1 & : & (x_4 \vee x_8 \vee x_9) \\ c_2 & : & (x_4 \vee x_8 \vee \neg x_9) \\ c_3 & : & (x_4 \vee \neg x_8 \vee x_9) \\ c_4 & : & (x_4 \vee \neg x_8 \vee \neg x_9) \\ \vdots & : & \vdots \\ c_m & : & \ldots \end{array}$$

and lexicographical order as variable selection heuristic. Suppose we assign all the variables to $false$ until decision level 7 without finding conflicts. When we reach the decision level 8, we detect a contradiction when the variable x_8 is set to $false$ between clauses c_1 and c_2, and between clauses c_3 and c_4 if we set variable x_8 to $true$. A chronological backtracking solver would backtrack to decision level 7 because it is the previous decision level with a variable that has not been flipped. However, flipping variables at a decision level greater than 4 does not resolve the conflict. A non-chronological backtracking solver can analyze this particular problem, determine the variable that produces the conflict, and backtrack to its level. In this example, the conflict analysis procedure would resolve to backtrack to level 4 and flip variable x_4 to $true$.

Copia gratuita. Personal free copy http://libros.csic.es

During the conflict analysis process, the information about the current conflict can be stored by means of redundancy [BS94, BGS99]. These redundant clauses do not change the satisfiability of the original formula, and they help prune parts of the search space with conflicts that involve variables of the *learned* conflict. This technique is called *clause learning* or *conflict driven clause learning*, and is used in solvers like BerkMin [GN01], Chaff [MMZ$^+$01], GRASP [MSS99], MiniSat [ES03] and siege [Rya04].

The implication relationships of variable assignments during the SAT solving process can be represented in an *implication graph*. An implication graph is a directed acyclic graph (DAG) in which each node represents a variable assignment, and the incident edges of a vertex are the reasons that imply the variable assignment. Figure 2.2 shows a typical implication graph. The incident edges to node x_5 are from x_1 and $\neg x_4$, which means that if x_1 is *true* and x_4 is $false$, then x_5 must be *true*. A decision vertex has no incident edge. In an implication graph, a variable and its negation only appear when a conflict occurs. Such a variable is called *conflicting variable*. The conflicting variable in Figure 2.2 is x_6.

In SAT solvers, clause learning can be applied by analyzing the implication graph. For example, in Figure 2.2, by examining the incident vertex of the nodes of the conflict variable, it is easy to see that the assignment of x_2 and x_4 to $false$, and x_3 and x_5 to *true* leads to the conflict between nodes x_6 and $\neg x_6$:

$$\neg x_2 \land x_3 \land \neg x_4 \land x_5 \Rightarrow conflict$$

If we want to avoid the conflict we could add the following clause:

$$\neg(\neg x_2 \land x_3 \land \neg x_4 \land x_5) \Leftrightarrow x_2 \lor \neg x_3 \lor x_4 \lor \neg x_5$$

As a result we get the *conflict clause* that is represented by the *cut* labeled as *conflict* in the implication graph.

The example shows that we can generate a conflict clause doing a bipartition of the implication graph. In the bipartition, we have decision variables on one side (the reason side), and conflicting variables on the other side (the conflict side). Each variable on the reason side with an edge to the conflict side belongs to the learned clause. This bipartition is called a *cut*, and different cuts correspond to different learning schemas.

Many learning schemas have been studied in the literature and L. Zhang compares some of them in his Ph.D. Thesis [Zha03]. One of the learning schemas with better performance, and used in modern SAT solvers, is 1-UIP (*first Unique Implication Point*) [MSS99]. A UIP is a vertex that dominates both vertices corresponding to the conflicting variable. The UIP is not unique, and we can find more than one in an implication graph. The 1-UIP learning schema picks the UIP closer to the conflict and cuts just after it. In Figure 2.2, there is only one UIP, represented by vertex $\neg x_4$, and the 1-UIP learning schema is represented by the cut labeled 1-UIP. The learned clause of this cut is $\neg x_1 \lor x_2 \lor \neg x_3 \lor x_4$.

The number of learned clauses can increase drastically the size of the database of clauses. Several clause deletion strategies have been proposed [GN01, BS97] but it is still an open topic.

© CSIC © del autor o autores / Todos los derechos reservados

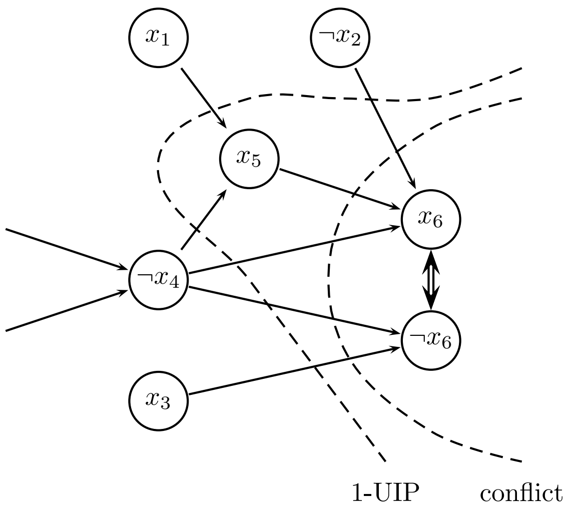


Figure 2.2: Implication graph.

Reasoning on special structures in SAT problems.

Given that many problems like pigeonhole or graph coloring involve a great deal of symmetry in their arguments, there has been suggested to add so-called symmetry-breaking clauses to the original formula [CGLR96, ASM06]; these are clauses that break the existing symmetry without affecting the overall satisfiability of the formula. Rather than modifying the set of clauses in the problem, it is also possible to modify the notion of inference, so that once a particular conflict has been derived, symmetric equivalents can be derived in a single step [Kri85].

Another explored deduction mechanism for special structured instances is equivalence reasoning. Solver eqsatz [Li03] incorporated equivalence reasoning into the Satz solver and found that it is effective on some particular classes of benchmarks. In that work, the equivalence reasoning is accomplished by a pattern-matching schema for equivalence clauses. In particular, finding equivalences of the type $x_1 \leftrightarrow x_2$ can reduce the number of variables and clauses of the formula, since variables x_1 and x_2 can be collapsed into one variable. A related deduction mechanism was proposed in [LMS01]. There, the authors propose to include more patterns in the matching process for simplification purposes in deduction. A more complex equivalence reasoning, with several steps, is performed in [WvM98, HDvMvZ04] as a preprocessing.

Copia gratuita. Personal free copy http://libros.csic.es

2.5 Local search for SAT

One of the drawbacks of complete methods (e.g., DP and DLL) is their inability to solve hard random 3-SAT instances with more than 700 propositional variables within a reasonable amount of time [DD01]. This drawback can be overcomed using incomplete[1] local search methods like GSAT [SLM92] and WalkSAT [SKC94]. These procedures are able to solve hard instances with more than 100,000 variables, though completeness is lost.

Local search (LS) algorithms start typically with some randomly generated complete assignment and try to find a satisfying assignment by iteratively changing the assignment of one propositional variable. Each change of the assignment of a variable is called a *variable flip*, and variables are selected heuristically. Such changes are repeated until either a satisfying assignment is found or a pre-set maximum number of changes is needed. This process is repeated as needed, up to a pre-set number of times. Usually, LS algorithms do not explore the entire search space, and a given assignment may be considered more than once.

The main difference among the different local search algorithms for SAT lies in the strategy used to select the variable to be flipped next. Furthermore, local search algorithms can get trapped in local minima and plateau regions of the search space, leading to premature stagnation of the search. One of the simplest mechanisms for avoiding premature stagnation of the search is *random restart*, which reinitializes the search if no solution has been found after a fixed number of steps. Random restarts are used in almost every local search algorithm for SAT.

A general outline of a local search algorithm for SAT is given in Algorithm 2.4. The generic procedure initializes the search at some complete truth assignment, and then iteratively selects a variable according to the input CNF formula and the current assignment, and flips the selected variable. If after a maximum of $maxSteps$ flips no solution is found, the algorithm restarts from a new randomly generated initial assignment. If after a given number $maxTries$ of such tries still no solution is found, the algorithm terminates unsuccessfully.

We now focus on the GSAT and the WalkSAT algorithms, which have provided a major driving force in the development of local search algorithms for SAT [SHR01, HS04].

2.5.1 GSAT algorithm

The GSAT algorithm was introduced in 1992 [SLM92]. It is based on a rather simple idea: GSAT tries to maximize the number of satisfied clauses by a greedy ascent in the space of truth assignments. The variable selection in GSAT and most of its variants is based on the *score* of a variable x under the current assignment $\mathcal{A}$, which is defined as the difference between the number of clauses

[1]An incomplete method in SAT can find a satisfying assignment, but cannot prove the unsatisfiability of a CNF formula. If a solution is found, the formula is declared satisfiable and the algorithm terminates successfully; but if the algorithm fails to find a solution, no conclusion can be drawn.

© CSIC © del autor o autores / Todos los derechos reservados

Algorithm 2.4: `LocalSearch`(ϕ) : General local search procedure

```
Output: Satisfying assignment of $\phi$ or 'no solution found'
for 1 to maxTries do
    A ← initAssign($\phi$)
    for 1 to maxSteps do
        if A satisfies $\phi$ then return A
        else
            x ← chooseVariable($\phi$, A)
            A ← A with truth value of x flipped
return 'no solution found'
```

falsified by the assignment obtained by flipping x in $\mathcal{A}$ and the number of clauses falsified by $\mathcal{A}$.

The basic GSAT algorithm uses the following instantiation of the procedure *chooseVariable*(ϕ, A): In each local search step, one of the variables with maximal score is flipped. If there are several variables with maximal score, one of them is randomly selected according to a uniform distribution.

2.5.2 WalkSAT algorithm

The WalkSAT algorithm was described by Selman, Kautz, and Cohen in 1994 [SKC94]. It is based on a 2-stage variable selection process focused on the variables occurring in currently unsatisfied clauses. For each local search step, in a first stage a currently unsatisfied clause c' is randomly selected. In a second step, one of the variables appearing in c' is then flipped to obtain the new assignment.

Thus, while the GSAT algorithm is characterized by a static neighborhood relation between assignments with Hamming distance one, the variable to be flipped in WalkSAT is no longer picked among all the variables but from a randomly selected unsatisfied clause [SHR01].

2.5.3 Other local search algorithms

Following the steps of GSAT and WalkSAT, the most relevant local search algorithms developed in the last years are the following ones:

HSAT [GW93] by Ian Gent and Toby Walsh. An improvement of GSAT, which takes the variable that was flipped longest ago for breaking ties.

TSAT [MSG97] by Bertrand Mazure, Lakhdar Saïs and Eric Grégoire. GSAT algorithm with tabu search.

novelty [MSK97] by McAllester, Selman, Kautz. This strategy selects the variable that minimizes the number of clauses which are currently satisfied but

Copia gratuita. Personal free copy http://libros.csic.es

would become violated if the variable is flipped, breaking ties in favor of the least recently flipped variable.

novelty+ [Hoo99] by Holger Hoos. Novelty with random walk and a user fixed noise in the choice of flip. Another variant of the same author is *adapt novelty+*, with self-fixed (adaptative) noise in the choice of flip.

SAPS [HTH02] by Holger Hoos et al. A weight is given to each clause, incrementing the weight to unsatisfied clauses. The variable occurring in more clauses of maximum weight is chosen.

g2wsat [LH05b] by Chu Min Li et al. It exploits promising decreasing paths using a gradient-based approach. When promising decreasing paths exist, the variable to be flipped is no longer selected from a randomly chosen unsatisfied clause but in a deterministic way to ensure that the number of unsatisfied clauses drecreases.

VW by Steve Prestwitch, in SAT 2005 Competition. A weight is given to each variable, which is increased in variables that are often flipped. The variable with minimum weight is chosen.

Most of these techniques can be checked in solver UBCSAT [TH05], from the University of British Columbia (UBC).

2.6 Summary

In this chapter we have presented an overview of the solving techniques most commonly used in SAT. Firstly, basic concepts in satisfiability solving have been introduced. Secondly, complete algorithms for SAT solving such as the DP procedure and the DLL procedure, as well as recent efficient techniques in complete SAT solving, are presented. Finally, the most representative local search algorithms have been described.

© CSIC © del autor o autores / Todos los derechos reservados

Chapter 3

Max-SAT algorithms

This chapter introduces some background knowledge about Max-SAT and reviews the solving techniques that have proved to be useful in terms of performance. In Section 3.1, we define some basic concepts in Max-SAT and Max-CSP. In Section 3.2, we describe how the branch and bound schema can be applied to exact Max-SAT solving. In Section 3.3, we define a complete resolution rule for Max-SAT. In Section 3.4, we review the main Max-SAT approximation algorithms. In Section 3.5, we describe the solving techniques that have been defined for dealing with hard and soft constraints under the formalism of Partial Max-SAT. Finally, in Section 3.6, we present the 2006 and 2007 Max-SAT Evaluations. In this chapter we follow closely the presentation of [LM09].

3.1 Basic concepts in Max-SAT and Max-CSP

3.1.1 Basic concepts in Max-SAT

In propositional logic a variable x_i may take values 0 (for false) or 1 (for true). A *literal* ℓ_i is a variable x_i or its negation $\neg x_i$. The *complementary* of a literal ℓ, denoted by $\bar{\ell}$, is x if $\ell = \neg x$ and is $\neg x$ if $\ell = x$. A *clause* is a disjunction of literals, and a CNF formula is a multiset of clauses. We define CNF formulas as multisets of clauses because repeated clauses cannot be collapsed into a unique clause in Max-SAT. For instance, the multiset $\{x_1, \neg x_1, \neg x_1, x_1 \vee x_2, \neg x_2\}$, where a clause is repeated, has a minimum of two unsatisfied clauses. A *weighted clause* is a pair (C_i, w_i), where C_i is a disjunction of literals and w_i, its weight, is a positive number, and a *weighted CNF formula* is a multiset of weighted clauses. A weighted CNF formula ϕ containing the clauses $(C, w_1), \ldots, (C, w_k)$, can be replaced by $(\phi - \{(C, w_1), \ldots, (C, w_k)\} \cup (C, w_1 + \cdots + w_k))$. The *length* of a (weighted) clause is the total number of literal occurrences in the clause. A (weighted) clause with one literal is called *unit*, with two literals is called *binary*, and with three literals is called *ternary*. The *size* of a (weighted) CNF formula ϕ, denoted by $|\phi|$, is the sum of the lengths of all its clauses.

Copia gratuita. Personal free copy http://libros.csic.es

An *assignment* of truth values to the propositional variables satisfies a literal x_i if x_i takes the value 1 and satisfies a literal $\neg x_i$ if x_i takes the value 0, satisfies a clause if it satisfies at least one literal of the clause, and satisfies a CNF formula if it satisfies all the clauses of the formula. A CNF formula is *satisfiable* if there exists an assignment that satisfies the formula; otherwise, it is *unsatisfiable*. An *empty clause*, denoted by $\square$, contains no literals and cannot be satisfied. A *tautology* is a CNF formula that is satisfied by any truth assignment.

An assignment of truth values to the propositional variables satisfies a weighted clause (C_i, w_i) if it satisfies C_i, and satisfies a weighted CNF formula $\{(C_1, w_1), \ldots, (C_m, w_m)\}$ if it satisfies $C_1, \ldots, C_m$.

An assignment for a CNF formula ϕ is *complete* if all the variables occurring in ϕ have been assigned; otherwise, it is *partial.* A partial truth assignment also partitions the clauses of a CNF formula into three sets: *satisfied* clauses, the clauses that contain at least one satisfied literal; *unsatisfied* clauses, the clauses in which all its literals are unsatisfied, and *unresolved* clauses, the clauses that the partial assignment makes them not to be decided. The unassigned literals of a clause are referred to as its *free literals.* In a search context, an unresolved clause is said to be *unit* if the number of its free literals is one. Similarly, an unresolved clause with two free literals is said to be *binary*, and an unresolved clause with three free literals is said to be *ternary.*

The *Max-SAT problem* for a CNF formula ϕ is the problem of finding an assignment of values to propositional variables that maximizes the number of satisfied clauses. In the sequel, we often use the term Max-SAT meaning Min-UNSAT. This is because, with respect to exact computations, finding an assignment that minimizes the number of unsatisfied clauses is equivalent to finding an assignment that maximizes the number of satisfied clauses. Notice that an upper (lower) bound in Min-UNSAT is greater (smaller) than or equal to the minimum number of clauses that are falsified by an interpretation.

Example 3.1 *Let ϕ be a Max-SAT instance with the following clauses:*

$$\begin{array}{lcl}
c_1 & : & x_1 \\
c_2 & : & \neg x_1 \vee x_2 \\
c_3 & : & x_1 \vee \neg x_2 \vee x_3 \\
c_4 & : & \neg x_1 \vee \neg x_2 \\
c_5 & : & x_1 \vee x_2 \vee \neg x_3 \\
c_6 & : & \neg x_1 \vee x_3 \\
c_7 & : & \neg x_1 \vee \neg x_2 \vee \neg x_3
\end{array}$$

An assignment that satisfies (falsifies) the maximum (minimum) number of clauses is: $x_1 = false$, $x_2 = true$ and $x_3 = true$. The CNF formula ϕ has a maximum (minimum) number of satisfied (falsified) clauses of 6 (1). The falsified clause with this assignment is c_1.

Max-SAT is called Max-k-SAT when all the clauses have at most k literals per clause.

© CSIC © del autor o autores / Todos los derechos reservados

In Max-SAT, two instances ϕ_1 and ϕ_2 are *equivalent* if ϕ_1 and ϕ_2 have the same number of unsatisfied clauses for every complete assignment of ϕ_1 and ϕ_2.

We will also consider three extensions of Max-SAT which are more well-suited for representing and solving over-constrained problems: weighted Max-SAT, Partial Max-SAT and weighted Partial Max-SAT.

The *weighted Max-SAT* problem for a weighted CNF formula ϕ is the problem of finding an assignment of values to propositional variables that maximizes the sum of weights of satisfied clauses (or equivalently, that minimizes the sum of weights of unsatisfied clauses).

Example 3.2 *Let ϕ be a weighted Max-SAT instance with the following clauses:*

$$\begin{array}{lcl}
c_1 & : & (x_1; 5) \\
c_2 & : & (\neg x_1 \vee x_2; 4) \\
c_3 & : & (x_1 \vee \neg x_2 \vee x_3; 3) \\
c_4 & : & (\neg x_1 \vee \neg x_2; 2) \\
c_5 & : & (x_1 \vee x_2 \vee \neg x_3; 4) \\
c_6 & : & (\neg x_1 \vee x_3; 1) \\
c_7 & : & (\neg x_1 \vee \neg x_2 \vee \neg x_3; 2)
\end{array}$$

An assignment that maximizes (minimizes) the sum of weights of satisfied (falsified) clauses is: $x_1 = true$, $x_2 = true$ and $x_3 = false$. The weighted Max-SAT instance ϕ has a maximum (minimum) sum of weights of satisfied (falsified) clauses of 18 (3). The falsified clauses with this assignment are c_4 with weight 2, and c_6 with weight 1.

A *Partial Max-SAT* instance is a CNF formula in which some clauses are *relaxable* or *soft*, and the rest are *non-relaxable* or *hard*. The *Partial Max-SAT problem* for a Partial Max-SAT instance ϕ is the problem of finding an assignment that satisfies all the hard clauses and the maximum number of soft clauses. Hard clauses are represented between square brackets, and soft clauses are represented between round brackets.

Example 3.3 *Let ϕ be a Partial Max-SAT instance with the following clauses:*

$$\begin{array}{lcl}
c_1 & : & [x_1 \vee x_2] \\
c_2 & : & [\neg x_1 \vee \neg x_2] \\
c_3 & : & (x_1 \vee \neg x_2 \vee x_3) \\
c_4 & : & (\neg x_1 \vee x_2 \vee x_3) \\
c_5 & : & (x_2 \vee \neg x_3) \\
c_6 & : & (\neg x_2 \vee \neg x_3) \\
c_7 & : & (x_3)
\end{array}$$

An assignment that satisfies all the hard clauses and maximizes (minimizes) the number of satisfied (falsified) soft clauses is: $x_1 = false$, $x_2 = true$ and $x_3 = true$. The Partial Max-SAT instance ϕ has a maximum (minimum) number of satisfied (falsified) soft clauses of 4 (1). The falsified clause with this assignment is c_6.

Copia gratuita. Personal free copy http://libros.csic.es

The *weighted Partial Max-SAT* problem is the combination of weighted Max-SAT and Partial Max-SAT.

Example 3.4 *Let ϕ be a weighted Partial Max-SAT instance with the following clauses:*

$$\begin{array}{lcl} c_1 & : & [x_1 \vee x_2] \\ c_2 & : & [\neg x_1 \vee \neg x_2] \\ c_3 & : & (x_1 \vee \neg x_2 \vee x_3; 1) \\ c_4 & : & (\neg x_1 \vee x_2 \vee x_3; 2) \\ c_5 & : & (x_2 \vee \neg x_3; 5) \\ c_6 & : & (\neg x_2 \vee \neg x_3; 6) \\ c_7 & : & (x_3; 3) \end{array}$$

An assignment that satisfies all the hard clauses and maximizes (minimizes) the sum of weights of satisfied (falsified) soft clauses is: $x_1 = false$, $x_2 = true$ and $x_3 = false$. The weighted Partial Max-SAT instance ϕ has a maximum (minimum) sum of weights of satisfied (falsified) soft clauses of 13 (4). The falsified clauses with this assignment are c_3 with weight 1, and c_7 with weight 3.

Max-SAT can also be defined as weighted Max-SAT restricted to formulas whose clauses have weight 1, and as Partial Max-SAT in the case that all the clauses are declared to be soft.

Finally, we introduce the integer linear programming (ILP) formulation of weighted Max-SAT, which is used to compute lower and upper bounds. Let $\phi = (C_1, w_1) \wedge \cdots \wedge (C_m, w_m)$ be a weighted Max-SAT instance over the propositional variables $x_1, \ldots, x_n$. With each propositional variable x_i, we associate a variable $y_i \in \{0,1\}$ such that $y_i = 1$ if variable x_i is true and $y_i = 0$, otherwise. With each clause C_j, we associate a variable $z_j \in \{0,1\}$ such that $z_j = 1$ if clause C_j is satisfied and $z_j = 0$, otherwise. Let I_j^+ be the set of indices of unnegated variables in clause C_j, and let I_j^- be the set of indices of negated variables in clause C_j. The ILP formulation of the weighted Max-SAT instance ϕ is defined as follows:

$$\max F(y,z) = \sum_{j=1}^{m} w_j z_j$$

subject to

$$\sum_{i \in I_j^+} y_i + \sum_{i \in I_j^-} (1 - y_i) \geq z_j \quad j = 1, \ldots, m$$
$$y_i \in \{0,1\} \quad i = 1, \ldots, n$$
$$z_j \in \{0,1\} \quad j = 1, \ldots, m$$

If we consider the minimization version of weighted Max-SAT (i.e.; weighted Min-UNSAT), we assume that with each clause C_j, we associate a variable $z_j \in \{0,1\}$ such that $z_j = 1$ if clause C_j is falsified and $z_j = 0$, otherwise. Then, the ILP formulation of the instance ϕ is defined as follows:

$$\min F(y,z) = \sum_{j=1}^{m} w_j z_j$$

© CSIC © del autor o autores / Todos los derechos reservados

subject to

$$\sum_{i \in I_j^+} y_i + \sum_{i \in I_j^-} (1 - y_i) + z_j \geq 1 \quad j = 1, \ldots, m$$

$$y_i \in \{0, 1\} \quad i = 1, \ldots, n$$

$$z_j \in \{0, 1\} \quad j = 1, \ldots, m$$

Example 3.5 *The minimization version of the ILP formulation for the weighted Max-SAT instance ϕ of Example 3.2 is as follows:*

$$\min F(y, z) = 5z_1 + 4z_2 + 3z_3 + 2z_4 + 4z_5 + 1z_6 + 2z_7$$

subject to

$$\begin{array}{l} y_1 + z_1 \geq 1 \\ (1 - y_1) + y_2 + z_2 \geq 1 \\ y_1 + (1 - y_2) + y_3 + z_3 \geq 1 \\ (1 - y_1) + (1 - y_2) + z_4 \geq 1 \\ y_1 + y_2 + (1 - y_3) + z_5 \geq 1 \\ (1 - y_1) + y_3 + z_6 \geq 1 \\ (1 - y_1) + (1 - y_2) + (1 - y_3) + z_7 \geq 1 \end{array}$$

$$y_i \in \{0, 1\} \quad i = 1, 2, 3$$

$$z_j \in \{0, 1\} \quad j = 1, 2, 3, 4, 5, 6, 7$$

An assignment that minimizes $F(y, z)$ is: $y_1 = 1$, $y_2 = 1$ and $y_3 = 0$. The variables z_i that must be assigned to 1 are z_4 and z_6.

The ILP formulation of the weighted Partial Max-SAT instance

$$\phi = [C_1] \wedge \cdots \wedge [C_k] \wedge (C_{k+1}, w_{k+1}) \wedge \cdots \wedge (C_m, w_m)$$

is defined as follows:

$$\max F(y, z) = \sum_{j=k+1}^{m} w_j z_j$$

subject to

$$\sum_{i \in I_j^+} y_i + \sum_{i \in I_j^-} (1 - y_i) \geq 1 \quad j = 1, \ldots, k$$

$$\sum_{i \in I_j^+} y_i + \sum_{i \in I_j^-} (1 - y_i) \geq z_j \quad j = k + 1, \ldots, m$$

$$y_i \in \{0, 1\} \quad i = 1, \ldots, n$$

$$z_j \in \{0, 1\} \quad j = k + 1, \ldots, m$$

Copia gratuita. Personal free copy http://libros.csic.es

If we consider the minimization version of weighted Partial Max-SAT (i.e.; weighted Partial Min-UNSAT), then the ILP formulation of the instance ϕ is defined as follows:

$$\min F(y,z) = \sum_{j=k+1}^{m} w_j z_j$$

subject to

$$\sum_{i \in I_j^+} y_i + \sum_{i \in I_j^-} (1 - y_i) \geq 1 \quad j = 1, \ldots, k$$

$$\sum_{i \in I_j^+} y_i + \sum_{i \in I_j^-} (1 - y_i) + z_j \geq 1 \quad j = k+1, \ldots, m$$

$$y_i \in \{0,1\} \quad i = 1, \ldots, n$$

$$z_j \in \{0,1\} \quad j = k+1, \ldots, m$$

Example 3.6 *The minimization version of the ILP formulation for the weighted Partial Max-SAT instance ϕ of Example 3.4 is as follows:*

$$\min F(y,z) = 1z_1 + 2z_2 + 5z_3 + 6z_4 + 3z_5$$

subject to

$$\begin{array}{l} y_1 + y_2 \geq 1 \\ (1 - y_1) + (1 - y_2) \geq 1 \\ y_1 + (1 - y_2) + y_3 + z_1 \geq 1 \\ (1 - y_1) + y_2 + y_3 + z_2 \geq 1 \\ y_2 + (1 - y_3) + z_3 \geq 1 \\ (1 - y_2) + (1 - y_3) + z_4 \geq 1 \\ y_3 + z_5 \geq 1 \end{array}$$

$$y_i \in \{0,1\} \quad i = 1,2,3$$

$$z_j \in \{0,1\} \quad j = 1,2,3,4,5$$

An assignment that minimizes $F(y,z)$ is: $y_1 = 0$, $y_2 = 1$ and $y_3 = 0$. The variables z_i that must be assigned to 1 are z_1 and z_5.

The linear programming (LP) relaxation of the previous formulations is obtained by allowing the integer variables to take real values in $[0,1]$.

3.1.2 Basic concepts in Max-CSP

A *Constraint Satisfaction Problem (CSP)* instance is defined by a triple $\langle X, D, C \rangle$, where $X = \{x_1, \ldots, x_n\}$ is a set of variables, $D = \{d(x_1), \ldots, d(x_n)\}$ is a set of finite domains containing the values that the variables may take, and $C = \{c_1, \ldots, c_m\}$ is a set of constraints. Each constraint $c_i = \langle S_i, R_i \rangle$ is defined as a relation R_i over a subset of variables $S_i = \{x_{i_1}, \ldots, x_{i_k}\}$, called the *constraint scope*. The relation R_i may be represented extensionally as a subset of the Cartesian product $d(x_{i_1}) \times \cdots \times d(x_{i_k})$.

© CSIC © del autor o autores / Todos los derechos reservados

An *assignment* v for a CSP instance $\langle X, D, C\rangle$ is a mapping that assigns to every variable $x_i \in X$ an element $v(x_i) \in d(x_i)$. An assignment v satisfies a constraint $\langle \{x_{i_1}, \ldots, x_{i_k}\}, R_i\rangle \in C$ iff $\langle v(x_{i_1}), \ldots, v(x_{i_k})\rangle \in R_i$. An assignment satisfies a CSP if it satisfies all its constraints.

The Constraint Satisfaction Problem (CSP) for a CSP instance P consists in deciding whether there exists an assignment that satisfies P.

In the rest of the thesis we assume that all CSPs are unary and binary; i.e., the scope of all the constraints has at most cardinality two.

Example 3.7 *Let P be the CSP instance defined by $\langle X, D, C\rangle$, where $X = \{x_1, x_2, x_3, x_4\}$ is the set of variables, $d(x_1) = d(x_2) = d(x_3) = d(x_4) = \{1, 2, 3\}$, and $C = \{\langle\{x_1, x_2\}, x_1 \neq x_2\rangle, \langle\{x_1, x_3\}, x_1 < x_3\rangle, \langle\{x_2, x_4\}, x_2 = x_4\rangle, \langle\{x_3, x_4\}, x_3 > x_4\rangle\}$ is the set of constraints. An assignment that satisfies P is $x_1 = 1$, $x_2 = 2$, $x_3 = 3$ and $x_4 = 2$; i.e., the CSP instance P has a solution.*

The Max-CSP problem for a CSP instance $\langle X, D, C\rangle$ is the problem of finding an assignment that minimizes (maximizes) the number of violated (satisfied) constraints.

Example 3.8 *Let P be the CSP instance defined by $\langle X, D, C\rangle$, where $X = \{x_1, x_2, x_3\}$ is the set of variables, $d(x_1) = d(x_2) = d(x_3) = \{1, 2, 3\}$, and $C = \{\langle\{x_1, x_2\}, x_1 < x_2\rangle, \langle\{x_2, x_3\}, x_2 < x_3\rangle, \langle\{x_1, x_3\}, x_1 \neq x_3\rangle\}$ is the set of constraints. An assignment that minimizes the number of violated constraints of P is $x_1 = 1$, $x_2 = 2$ and $x_3 = 3$. The number of violated (satisfied) constraints with this assignment is 1 (2).*

3.2 Branch and bound algorithms

Competitive exact Max-SAT solvers —as the ones developed by [AMP03, AMP04, AMP05, AMP08, DDDL07, HL06, HLO07, LHdG08, LMP05, LMP06, LS07, PD07, RG07, SZ04, XZ04, XZ05, ZSM03]— implement variants of the following branch and bound (BnB) schema for solving the minimization version of Max-SAT: Given a CNF formula ϕ, BnB explores the search tree that represents the space of all possible assignments for ϕ in a depth-first manner. At every node, BnB compares the upper bound (UB), which is the best solution found so far for a complete assignment, with the lower bound (LB), which is the sum of the number of clauses unsatisfied by the current partial assignment plus an under-estimation of the number of clauses that will become unsatisfied if the current partial assignment is completed. If $LB \geq UB$, the algorithm prunes the subtree below the current node and backtracks chronologically to a higher level in the search tree. If $LB < UB$, the algorithm tries to find a better solution by extending the current partial assignment by instantiating one more variable. The solution to Max-SAT is the value that UB takes after exploring the entire search tree.

Figure 3.1 shows the pseudo-code of a basic solver for Max-SAT. We use the following notation:

Copia gratuita. Personal free copy http://libros.csic.es

Algorithm 3.1: `Max-SAT`(ϕ, UB) : Basic BnB algorithm for Max-SAT

```
Output: The minimal number of unsatisfied clauses in $\phi$
Function Max-SAT ($\phi$ : CNF formula, $UB$ : upper bound) : Natural
    $\phi \leftarrow$ SimplifyFormula($\phi$)
    if $\phi = \emptyset$ or $\phi$ only contains empty clauses then
        return EmptyClauses($\phi$)
    $LB \leftarrow$ EmptyClauses($\phi$) + Underestimation($\phi$)
    if $LB \geq UB$ then
        return $UB$
    $x \leftarrow$ SelectVariable($\phi$)
    $UB \leftarrow$ Min($UB$, Max-SAT($\phi_{\neg x}$, $UB$))
    return Min($UB$, Max-SAT($\phi_x$, $UB$))
```

- `SimplifyFormula`(ϕ) is a procedure that transforms ϕ into an equivalent and simpler instance by applying inference rules.

- `EmptyClauses`(ϕ) is a function that returns the number of empty clauses in ϕ.

- `Underestimation`(ϕ,UB) is a function that returns an underestimation of the minimum number of non-empty clauses in ϕ that will become unsatisfied if the current partial assignment is extended to a complete assignment.

- LB is a lower bound. We assume that its initial value is 0.

- UB is an upper bound of the number of unsatisfied clauses in an optimal solution. An elementary initial value for UB is the total number of clauses in the input formula, or the number of clauses unsatisfied by an arbitrary interpretation. Another alternative is to solve the LP relaxation of the ILP formulation of the input instance and take as upper bound the number of unsatisfied clauses in the interpretation obtained by rounding variable y_i, for $1 \leq i \leq n$, to an integer solution in a randomized way by interpreting the values of $y_i \in [0, 1]$ as probabilities (set propositional variable x_i to true with probability y_i, and set propositional variable x_i to false with probability $1 - y_i$). Nevertheless, most of the solvers take as initial upper bound the minimum number of unsatisfied clauses that are detected by executing the input formula in a local search solver during a short period of time.

- `SelectVariable`(ϕ) is a function that returns a variable of ϕ following an heuristic.

- ϕ_x ($\phi_{\neg x}$) is the formula obtained by applying the unit clause rule to ϕ using the literal x ($\neg x$).

© CSIC © del autor o autores / Todos los derechos reservados

State-of-the-art Max-SAT solvers implement the basic algorithm augmented with powerful inference techniques, good quality lower bounds, clever variable selection heuristics, learning of hard clauses, non-chronological backtracking, and efficient data structures.

3.2.1 Improving the lower bound with underestimations

The simplest method to compute a lower bound, when solving the minimization version of Max-SAT, consists of just counting the number of clauses unsatisfied by the current partial assignment [BF99]. One step forward is to incorporate an underestimation of the number of clauses that will become unsatisfied if the current partial assignment is extended to a complete assignment. The most basic method was defined by Wallace and Freuder [WF96]:

$$\mathtt{LB}(\phi) = \#emptyClauses(\phi) + \sum_{x \text{ occurs in } \phi} min(ic(x), ic(\neg x)),$$

where ϕ is the CNF formula associated with the current partial assignment, and $ic(x)$ ($ic(\neg x)$) —inconsistency count of x ($\neg x$)— is the number of unit clauses of ϕ that contain $\neg x$ (x). In other words, that underestimation is the number of disjoint inconsistent subformulas in ϕ formed by a unit clause with a literal ℓ and a unit clause with the complementary of ℓ.

Example 3.9 *Given the Max-SAT instance ϕ represented by the following multiset of clauses $\phi = \{\ \Box,\quad \neg x_1, x_1, x_1,\quad \neg x_2, x_2 \vee x_3, x_2 \vee \neg x_3,\quad \neg x_4, \neg x_5, x_6, x_4 \vee x_5 \vee \neg x_6,\quad \neg x_7, x_7\ \}$, the lower bound computed by the simplest method is 1 (number of empty clauses). With the method defined by Wallace and Freuder we have that:*

$$LB(\phi) = 1 + \sum_{x \text{ occurs in } \phi} min(ic(x), ic(\neg x)) = 3$$

Lower bounds dealing with longer clauses include the star rule and UP. In the star rule [SZ04, AMP04], the underestimation of the lower bound is the number of disjoint inconsistent subformulas of the form $\{l_1, \ldots, l_k, \bar{l}_1 \vee \cdots \vee \bar{l}_k\}$. The star rule, when $k = 1$, is equivalent to the inconsistency counts of Wallace and Freuder.

Example 3.10 *Given the Max-SAT instance of Example 3.9, we can detect an additional inconsistent subformula by applying the star rule among the clauses $\{\neg x_4, \neg x_5, x_6, x_4 \vee x_5 \vee \neg x_6\}$ and increase the lower bound by 1.*

In UP [LMP05], the underestimation of the lower bound is the number of disjoint inconsistent subformulas that can be detected with unit propagation. UP works as follows: it applies unit propagation until a contradiction is derived. Then, UP identifies, by inspecting the implication graph, a subset of clauses from which a refutation can be constructed, and tries to identify new contradictions from the remaining clauses. The order in which unit clauses are propagated has a

Copia gratuita. Personal free copy http://libros.csic.es

clear impact on the quality of the lower bound [LMP06]. Shen and Zhang [SZ04] defined a lower bound computation method, called LB4, which is similar to UP but restricted to Max-2-SAT instances and using a static variable ordering.

Example 3.11 *Given the Max-SAT instance of Example 3.9, we can propagate $\neg x_2$ and detect, with unit propagation, the inconsistent subformula $\{\neg x_2, x_2 \vee x_3, x_2 \vee \neg x_3\}$, and increase the lower bound by 1.*

UP can be enhanced with failed literals as follows: Given a Max-SAT instance ϕ and a variable x occurring positively and negatively in ϕ, we apply UP to $\phi \wedge \{x\}$ and $\phi \wedge \{\neg x\}$. If UP derives a contradiction from $\phi \wedge \{x\}$ and $\phi \wedge \{\neg x\}$, then the union of the two inconsistent subsets identified by UP is an inconsistent subset of ϕ. UP enhanced with failed literals does not need the occurrence of unit clauses in the input formula for deriving a contradiction. While UP only identifies unit refutations, UP enhanced with failed literals identifies non-unit refutations too. Since applying detection of failed literals for every variable is time consuming, it is applied to a reduced number of variables in practice.

Example 3.12 *Given the Max-SAT instance ϕ represented by the following multiset of clauses $\phi = \{\, x_1 \vee x_2, x_1 \vee \neg x_2, \neg x_1 \vee x_2, \neg x_1 \vee \neg x_2, \quad \neg x_3, x_4 \vee x_3, x_5\}$, we can detect an inconsistent subformula using failed literals as follows: we derive a contradiction when we apply UP to $\phi \vee \{x_1\}$, and also when we apply UP to $\phi \vee \{\neg x_1\}$. The union of the two detected inconsistent subsets is an inconsistens subset of ϕ formed by the clauses $\{x_1 \vee x_2, x_1 \vee \neg x_2, \neg x_1 \vee x_2, \neg x_1 \vee \neg x_2\}$.*

Modern Max-SAT solvers like LB-SAT, MaxSatz and MiniMaxsat apply either UP or UP enhanced with failed literal detection. Recently, Darras et al. [DDDL07] have developed a version of UP in which the computation of the lower bound is made more incremental by saving some of the small size disjoint inconsistent subformulas detected by UP. They avoid to redetect the saved inconsistencies if they remain in subsequent nodes of the proof tree, and are able to solve some types of instances faster.

Marques-Silva and Planes [MSP08] developed a solver, called msu4, which solves Max-SAT using a SAT solver equipped with unsatisfiability cores detection and cardinality constraints. In ms4, the number of detected disjoint unsatisfiable cores is taken as a lower bound.

Another approach for computing underestimation is based on first reducing the Max-SAT instance we want to solve to an instance of another problem, and then solve a relaxation of the obtained instance. For example, two solvers of the 2007 Max-SAT Evaluation, Clone [PD07] and SR(w) [RG07], reduce Max-SAT to the minimum cardinality problem. Since the minimum cardinality problem is NP-hard for a CNF formula ϕ and can be solved in time linear in the size of a deterministic decomposable negation normal form (d-DNNF) compilation of ϕ, Clone and SR(w) solve the minimum cardinality problem of a d-DNNF compilation of a relaxation of ϕ. The worst-case complexity of a d-DNNF compilation of ϕ is exponential in the treewidth of its constraint graph, and Clone and SR(w)

© CSIC © del autor o autores / Todos los derechos reservados

obtain a relaxation of ϕ with bounded treewidth by renaming different occurrences of some variables.

Xing and Zhang [XZ05] reduce the Max-SAT instance to the ILP formulation of the minimization version of Max-SAT (c.f. Section 1.2), and then solve the LP relaxation. An optimal solution of the LP relaxation provides an underestimation of the lower bound because the LP relaxation is less restricted than the ILP formulation. In practice, they apply that lower bound computation method only to nodes containing unit clauses. If each clause in the Max-SAT instance has more than one literal, then $y_i = \frac{1}{2}$ for all $1 \leq i \leq n$ and $z_j = 0$ for all $1 \leq j \leq m$ is an optimal solution of the LP relaxation. In this case, the underestimation is 0. Nevertheless, LP relaxations do not seem to be so competitive as the rest of approaches.

3.2.2 Improving the lower bound with inference

Another approach to improve the quality of the lower bound consists in applying inference rules that transform a Max-SAT instance ϕ into an *equivalent* but simpler Max-SAT instance ϕ'. In the best case, inference rules produce new empty clauses in ϕ' that allow to increment the lower bound. In contrast with the empty clauses derived when computing underestimations, the empty clauses derived with inference rules do not have to be recomputed at every node of the current subtree so that the lower bound computation is more incremental.

A Max-SAT inference rule is *sound* if it transforms an instance ϕ into an equivalent instance ϕ'. It is not sufficient to preserve satisfiability as in SAT, ϕ and ϕ' must have the same number of unsatisfied clauses for every possible assignment. Unfortunately, unit propagation, which is the most powerful inference technique applied in DPLL-style SAT solvers, is unsound for Max-SAT as the next example shows: The set of clauses $\{x_1, \neg x_1 \vee x_2, \neg x_1 \vee \neg x_2, \neg x_1 \vee x_3, \neg x_1 \vee \neg x_3\}$ has a minimum of one unsatisfied clause (setting x_1 to false), but performing unit propagation with x_1 leads to a non-optimal assignment falsifying two clauses.

Max-SAT inference rules are also called *transformation rules* in the literature because the premises of the rule are replaced with the conclusion when a rule is applied. If the conclusion is added to the premises as in SAT, the number of clauses unsatisfied by an assignment might increase.

The amount of inference performed by existing BnB Max-SAT solvers at each node of the proof tree is poor compared with the inference performed in DPLL-style SAT solvers. The simplest inference enforced, when branching on literal ℓ, is the following: the clauses containing ℓ are removed from the instance and the occurrences of $\bar{\ell}$ are removed from the clauses in which $\bar{\ell}$ appears, but the new unit clauses derived as a consequence of removing the occurrences of $\bar{\ell}$ are not propagated as in unit propagation. That inference is typically enhanced with Max-SAT inference rules as the ones described in the rest of this section.

We now present simple inference rules that have proved to be useful in a number of solvers [AMP03, AMP05, BF99, SZ04, XZ05], and then some more sophisticated inferences rules which are implemented in solvers like Max-

Copia gratuita. Personal free copy http://libros.csic.es

DPLL [LHdG08], MaxSatz [LMP07], and MiniMaxsat [HLO07]. Some simple inference rules are:

- The pure literal rule [BF99]: If a literal only appears with either positive or negative polarity in a MaxSAT instance, all the clauses containing that literal are removed.
- The dominating unit clause rule [NR00]: If the number of clauses (of any length) in which a literal l appears is not greater than the number of unit clauses in which $\bar{l}$ appears, all the clauses containing l and all the occurrences of $\bar{l}$ are removed.
- The complementary unit clause rule [NR00]: If a Max-SAT instance contains a unit clause with the literal l and a unit clause with the literal $\bar{l}$, these two clauses are replaced with one empty clause.
- The almost common clause rule [BR99]: If a Max-SAT instance contains a clause $x \vee D$ and a clause $\neg x \vee D$, where D is a disjunction of literals, then both clauses are replaced with D. In practice, this rule is applied when D contains at most one literal.

The resolution rule applied in SAT (i.e., derive $D \vee D'$ from $x \vee D$ and $\neg x \vee D'$) preserves satisfiability but not equivalence, and therefore cannot be applied to Max-SAT instances, except for some particular cases like the almost common clause rule. We refer the reader to Section 3.3 for a complete resolution calculus for Max-SAT, and devote the rest of this section to present some sound Max-SAT resolution rules that can be applied in polynomial time.

We next present the *star rule*: If $\phi_1{=}\{l_1,\ \bar{l}_1{\vee}\bar{l}_2,\ l_2\}{\cup}\phi'$, then $\phi_2{=}\{\Box,\ l_1{\vee}l_2\}{\cup}\ \phi'$ is equivalent to ϕ_1. This rule, which can be seen as the inference counterpart of the underestimation of the same name, can also be presented as follows:

$$\left\{\begin{array}{c} l_1 \\ \bar{l}_1 \vee \bar{l}_2 \\ l_2 \end{array}\right\} \Longrightarrow \left\{\begin{array}{c} \Box \\ l_1 \vee l_2 \end{array}\right\}$$

Notice that the rule detects a contradiction from $l_1, \bar{l}_1 \vee \bar{l}_2, l_2$ and, therefore, replaces these clauses with an empty clause. In addition, the rule adds the clause $l_1 \vee l_2$ to ensure the equivalence between ϕ_1 and ϕ_2. For any assignment containing either $l_1 = 0, l_2 = 1$, or $l_1 = 1, l_2 = 0$, or $l_1 = 1, l_2 = 1$, the number of unsatisfied clauses in $\{l_1, \bar{l}_1 \vee \bar{l}_2, l_2\}$ is 1, but for any assignment containing $l_1 = 0, l_2 = 0$, the number of unsatisfied clauses is 2. Notice that even when any assignment containing $l_1 = 0, l_2 = 0$ is not the best assignment for the subset $\{l_1, \bar{l}_1 \vee \bar{l}_2, l_2\}$, it can be the best for the whole formula. By adding $l_1 \vee l_2$, the rule ensures that the number of unsatisfied clauses in ϕ_1 and ϕ_2 is also the same when $l_1 = l_2 = 0$.

MaxSatz implements the almost common clause rule, the star rule, and additional inference rules presented in Section 5.2.4.

© CSIC © del autor o autores / Todos los derechos reservados

Independently and in parallel to the definition of the rules of MaxSatz, similar inference rules were defined for weighted Max-SAT by Heras and Larrosa [LH05b, HL06], and were implemented in Max-DPLL [LHdG08]. These rules were inspired by the soft local consistency properties defined in the Constraint Programming community [dGHZL05, CdGS07]. The rules implemented in Max-DPLL are the almost common clause rule, chain resolution and cycle resolution. Chain resolution, which allows one to derive a new empty clause, is defined as follows:

$$\left\{\begin{array}{l}(l_1, w_1),\\ (\bar{l}_i \vee l_{i+1}, w_{i+1})_{1\le i<k},\\ (\bar{l}_k, w_{k+1})\end{array}\right\} \Longrightarrow \left\{\begin{array}{l}(l_i, m_i - m_{i+1})_{1\le i\le k},\\ (\bar{l}_i \vee l_{i+1}, w_{i+1} - m_{i+1})_{1\le i<k},\\ (l_i \vee \bar{l}_{i+1}, m_{i+1})_{1\le i<k},\\ (\bar{l}_k, w_{k+1} - m_{k+1}),\\ (\Box, m_{k+1})\end{array}\right\}$$

where w_i, $1 \le i \le k+1$, is the weight of the corresponding clause, and $m_i = \min(w_1, w_2, \ldots, w_i)$.

Cycle resolution, which allows one to derive a new unit clause and whose application is restricted to $k = 3$ in Max-DPLL, is defined as follows:

$$\left\{\begin{array}{l}(\bar{l}_i \vee l_{i+1}, w_i)_{1\le i<k},\\ (\bar{l}_1 \vee \bar{l}_k, w_k)\end{array}\right\} \Longrightarrow \left\{\begin{array}{l}(\bar{l}_1 \vee l_i, m_{i-1} - m_i)_{2\le i\le k},\\ (\bar{l}_i \vee l_{i+1}, w_i - m_i)_{2\le i<k},\\ (\bar{l}_1 \vee l_i \vee \bar{l}_{i+1}, m_i)_{2\le i<k},\\ (l_1 \vee \bar{l}_i \vee l_{i+1}, m_i)_{2\le i<k},\\ (\bar{l}_1 \vee \bar{l}_k, w_k - m_k),\\ (\bar{l}_1, m_k)\end{array}\right\}$$

A more general inference schema is implemented in MiniMaxsat [HLO07]. It detects a contradiction with unit propagation and identifies an unsatisfiable subset. Then, it creates a refutation for that unsatisfiable subset, and applies the Max-SAT resolution rule defined in Section 3.3 if the size of the largest resolvent in the refutation is less than 4.

3.2.3 Variable selection heuristics

Most of the exact Max-SAT solvers incorporate variable selection heuristics that take into account the number of literal occurrences in such a way that each occurrence has an associated weight that depends on the length of the clause that contains the literal. Max-SAT heuristics give priority to literals occurring in binary clauses instead of literals occurring in unit clauses as SAT heuristics do. The goal is to generate as many unit clauses as possible.

Let us see as an example the variable selection heuristic of MaxSatz [LMP07]: Let $neg1(x)$ ($pos1(x)$) be the number of unit clauses in which x is negative (positive), $neg2(x)$ ($pos2(x)$) be the number of binary clauses in which x is negative (positive), and let $neg3(x)$ ($pos3(x)$) be the number of clauses containing three or more literals in which x is negative (positive). MaxSatz selects the variable x

Copia gratuita. Personal free copy http://libros.csic.es

such that $(neg1(x)+4*neg2(x)+neg3(x))*(pos1(x)+4*pos2(x)+pos3(x))$ is the largest. Once a variable x is selected, MaxSatz applies the following value selection heuristic: If $neg1(x)+4*neg2(x)+neg3(x) < pos1(x)+4*pos2(x)+pos3(x)$, set x to true; otherwise, set x to false.

Other Max-SAT solvers (AMP [AMP03], Lazy [AMP08], MaxSolver [XZ05], Max-DPLL [LHdG08], MiniMaxsat [HLO07], ...) incorporate variants of the two-sided Jeroslow-Wang rule that give priority to variables occurring often in binary clauses. MaxSolver changes the weights as the search proceeds.

MiniMaxsat incorporates VSIDS [MMZ+01] and the two-sided Jeroslow-Wang rule. It automatically changes to the two-sided Jeroslow-Wang heuristic if the problem does not contain any literal ℓ such that ℓ and $\bar{\ell}$ appear in some hard clauses.

3.2.4 Data structures

Data structures for SAT have been naturally extended to Max-SAT. We can divide the solvers into two classes: solvers like BF and MaxSatz representing formulas with adjacency lists, and solvers like Lazy and MiniMaxsat which use data structures with watched literals. Lazy data structures are particularly good when there is a big number of clauses; for example, in Partial Max-SAT solvers with clause learning.

It is also worth to point out that the lower bound computation methods based on unit propagation represent the different derivations of unit clauses in a graph, called the implication graph [LMP07]. Looking at that graph, solvers identify the clauses which are involved in the derivation of a contradiction. Furthermore, in some solvers the implication graph is used to decide whether the clauses involved in a contradiction match with the premises of some inference rule.

3.3 Complete inference in Max-SAT

A natural extension to Max-SAT of the resolution rule applied in SAT was defined by Larrosa and Heras [LH05b]:

$$\begin{array}{c} x \vee A \\ \neg x \vee B \\ \hline\hline A \vee B \\ x \vee A \vee \neg B \\ \neg x \vee \neg A \vee B \end{array}$$

However, two of the conclusions of this rule are not in clausal form, and the application of distributivity results into an unsound rule. Independently and in parallel, Bonet et al. [BLM06, BLM07], and Heras and Larrosa [HL06] defined a sound version of the rule with the conclusions in clausal form:

© CSIC © del autor o autores / Todos los derechos reservados

$$\begin{array}{l}
x \vee a_1 \vee \cdots \vee a_s \\
\overline{x} \vee b_1 \vee \cdots \vee b_t \\
\hline\hline
a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_t \\
x \vee a_1 \vee \cdots \vee a_s \vee \overline{b_1} \\
x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \overline{b_2} \\
\cdots \\
x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_{t-1} \vee \overline{b_t} \\
\overline{x} \vee b_1 \vee \cdots \vee b_t \vee \overline{a_1} \\
\overline{x} \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \overline{a_2} \\
\cdots \\
\overline{x} \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \cdots \vee a_{s-1} \vee \overline{a_s}
\end{array}$$

This inference rule concludes, apart from the conclusion where a variable has been cut, some additional clauses that contain one of the premises as subclause. We say that the rule *cuts* the variable x. The tautologies concluded by the rule are removed, and the repeated literals in a clause are collapsed into one.

Notice that an instance of Max-SAT resolution not only depends on the two premises and the cut variable (like in resolution), but also on the order of the literals in the premises. Notice also that, like in resolution, this rule concludes a new clause not containing the variable x, except when this clause is a tautology.

Moreover, Bonet et al. [BLM06, BLM07] proved the completeness of Max-SAT resolution: By *saturating* successively w.r.t. all the variables, one derives as many empty clauses as the minimum number of unsatisfied clauses in the the Max-SAT input instance. Saturating w.r.t. a variable amounts to apply the Max-SAT resolution rule to clauses containing that variable until every possible application of the inference rule only introduces clauses containing that variable (since tautologies are eliminated). Once a Max-SAT instance is saturated w.r.t. a variable, all the clauses containing that variable are not considered to saturate w.r.t. another variable. We refer to [BLM07] for further technical details and for the weighted version of the rule.

3.4 Approximation algorithms

Heuristic local search algorithms are often quite effective at finding near-optimal solutions. Actually, most of the exact Max-SAT solvers use a local search algorithm to compute an initial upper bound. However, these algorithms, in contrast with approximation algorithms, do not come with rigorous guarantees concerning the quality of the final solution or the required maximum runtime. Informally, an algorithm approximately solves an optimization problem if it always returns a feasible solution whose measure is close to optimal, for example, within a factor bounded by a constant or by a slowly growing function of the input size. Given a constant α, an algorithm is an α-approximation algorithm for a maximization (minimization) problem if it provides a feasible solution in polynomial time which is at least (most) α times the optimum, considering all the possible instances of the problem.

Copia gratuita. Personal free copy http://libros.csic.es

The first Max-SAT approximation algorithm, with a performance guarantee of $\frac{1}{2}$, is a greedy algorithm that was devised by Johnson in 1974 [Joh74]. This result was improved in 1994 by Yannakakis [Yan94], and Goemans and Williamson [GW94a], who described 3/4–approximation algorithms for Max-SAT. Then, Goemans and Williamson [GW94a] proposed a .878-approximation algorithm for Max2SAT (which gives a .7584-approximation for Max-SAT [GW95]) based on semidefinite programming [GW94b]. Since then other improvements have been achieved, but there is a limit on approximability: Hastad [Hås97] proved that, unless $P = NP$, no approximation algorithm for Max-SAT (even for Max3SAT) can achieve a performance guarantee better than 7/8. Interestingly, Karloff and Zwick [KZ97] gave a 7/8 approximation algorithm for Max3SAT, showing that the constant 7/8 is tight. The most promising approaches from a theoretical and practical point of view [GvHL06] are based on semidefinite programming. We refer the reader to the survey of Anjos [Anj05] to learn more about how approximate Max-SAT with semidefinite programming.

3.5 Partial Max-SAT and soft constraints

Partial Max-SAT allows to encode combinatorial problems with hard and soft constraints in a more natural and compact way than Max-SAT. Moreover, Partial Max-SAT is particularly interesting from an algorithmic point of view because the solvers exploit the distinction between hard and soft constraints. Such a structural property has a great impact on performance, and is crucial for deciding the solving techniques that can be applied at each node of the search space.

Inference in Partial Max-SAT tends to be more powerful than in Max-SAT. Since any optimal solution has to satisfy all the hard clauses, it can be applied the satisfiability preserving inference of SAT solver to hard clauses, and the equivalence preserving inference of Max-SAT solvers to soft clauses. This implies, for example, that the unit clause rule can be enforced when a unit hard clause is derived, allowing to eliminate one variable not just without introducing any additional clause, but reducing the size of the current instance. Another example of exploiting the fact of knowing which clauses are hard consists of pruning a subtree as soon as a hard clause is violated.

Computing underestimations of the lower bound based on unit propagation also benefits from the distinction between hard ans soft constraints. The hard clauses used to detect an inconsistent subset can be used again to detect further inconsistent subsets. It is specially advantageous to be able to reuse unit hard clauses to get lower bounds of better quality.

Learning of hard clauses, first introduced in [AM06b], is another feature of modern Partial Max-SAT solvers that lacks in (weighted) Max-SAT solvers. This kind of learning consists of recording a clause every time a hard clause is violated, and there is experimental evidence that it is particularly useful when solving structured instances [AM07, HLO07].

Fu and Malik [FM06] implemented two Partial Max-SAT solvers, ChaffBS and ChaffLS, on top of the SAT solver zChaff. In order to translate a Max-SAT

© CSIC © del autor o autores / Todos los derechos reservados

instance into a SAT one, they append a distinct slack variable to every Max-SAT clause. A true slack variable essentially means that the corresponding Max-SAT clause can be left unsatisfied. It then constructs a hierarchical tree adder using three-at-a-time adders (i.e., full adders). The hierarchical tree adder sums up the number of true slack variables and presents the summation in binary format to a logic comparator, which returns true if and only if the binary number is less than or equal to any given number k. At this point, the Max-SAT instance can be translated into a SAT instance, which consists of the Max-SAT clauses with slack variables and the SAT clauses correspond to the hierarchical tree adder and the logic comparator for a given k value. Obviously, k is greater than or equal to 0 and smaller than or equal to the total number of slack variables. In order to find the minimum k, i.e. the minimum number of true slack variables, we can either do Binary Search (ChaffBS) or Linear Search (ChaffLS) within the possible range of k. For ChaffLS, it starts with $k = 0$ and increase k by one until it finds the translated SAT instance satisfiable.

A similar SAT-based approach for solving Partial Max-SAT was developed by Daniel Le Berre in SAT4Jmaxsat [Ber]. In this case, cardinality constraints are used instead of a circuit as in ChaffBS and ChaffLS.

Partial Max-SAT solving dealing with blocks of soft clauses was considered in [AM06a]. In this case, satisfiability preserving inference rules can also be applied locally to each soft block.

In addition to the previous solvers, the state-of-the-art solvers dealing with Partial Max-SAT instances are:

Clone [PD07] by Knot Pipatsrisawat and Adnan Darwiche.

LB-PSAT [LS07] by Han Li and Kaile Su.

MiniMaxsat [HLO07] by Federico Heras, Javier Larrosa, Albert Oliveras and Simon de Givry.

msuncore [MSP08] by João P. Marques-Silva and Jordi Planes.

SR(w) [RG07] by Miquel Ramírez and Hector Geffner.

Toolbar [LHdG08] by Simon de Givry, Federico Heras, Javier Larrosa and Thomas Schiex.

3.6 Evaluations of Max-SAT solvers

The First and Second Max-SAT Evaluations were organized as affiliated events of the 2006 and 2007 editions of the International Conference on Theory and Applications of Satisfiability Testing (SAT-2006 and SAT-2007) with the aim of assessing the advancements in the field of Max-SAT solvers through a comparison of their performances, identifying successful solving techniques and encouraging researchers to develop new ones, and creating a publicly available collection of challenging Max-SAT benchmarks. In the 2006 Max-SAT Evaluation 6 solvers

Copia gratuita. Personal free copy http://libros.csic.es

participated and there were two categories (Max-SAT and weighted Max-SAT), while in the 2007 Max-SAT Evaluation 12 solvers participated and there were two additional categories (Partial Max-SAT and weighted Partial Max-SAT).

The solvers participating in the evaluation can be classified into three classes: (i) solvers like ChaffBS, ChaffLS and SAT4Jmaxsat that solve Max-SAT using a state-of-the-art SAT solver; (ii) solvers like LB-SAT, MaxSatz, MaxSatz14, MiniMaxsat and PMS that implement a branch and bound schema and apply inference rules and compute unit propagation-based underestimations of the lower bound at each node of the proof tree; and (iii) solvers like Clone and SR(w) that implement a branch and bound schema and compute an underestimation by solving a relaxation of a d-DNNF compiled translation of the Max-SAT instance into a minimum cardinality instance. Most of the solvers solving Partial Max-SAT, independently of the class to which they belong, incorporate learning of hard clauses. Solvers of the second class were the ones with better performance profile in both evaluations. Next, we describe all the solvers that have participated in the evaluations. The descriptions below correspond to the descriptions provided by the authors of the solvers that participated in the evaluations.

– **ChaffBS & ChaffLS (Zhaohui Fu and Sharad Malik):** Both ChaffBS and ChaffLS are implemented on top of the SAT solver zChaff. In order to translate a Max-SAT instance into a SAT one, they append a distinct slack variable to every Max-SAT clause. A true slack variable essentially means the corresponding Max-SAT clause can be left unsatisfied. They then construct a hierarchical tree adder using three-at-a-time adders (i.e., full adders). The hierarchical tree adder sums up the number of true slack variables and presents the summation in binary format to a logic comparator, which returns true if and only if the binary number is less than or equal to any given number k. At this point, the Max-SAT instance can be translated into a SAT instance, which consists of the Max-SAT clauses with slack variables and the SAT clauses correspond to the hierarchical tree adder and the logic comparator for a given k value. Obviously, k is greater than or equal to 0 and less than or equal to the total number of slack variables. In order to find the minimum k, i.e., the minimum number of true slack variables, one can either do Binary Search (ChaffBS) or Linear Search (ChaffLS) within the possible range of k. For ChaffLS, it starts with $k = 0$ and increase k by one until it finds the translated SAT instance satisfiable. As a side effect, ChaffLS is not able to produce any sub-optimal solution as the first solution it finds is the optimal solution. For further details see [FM06].

– **Clone (Knot Pipatsrisawat, Mark Chavira, Arthur Choi and Adnan Darwiche):** Clone is an exact Max-SAT solver that uses branch and bound search to find optimal solutions. The method for computing bounds used by Clone is rather different from those of contemporary Max-SAT solvers. Clone relaxes some constraints in the original CNF and turns it into an approximate formula, which is then compiled into a d-DNNF

© CSIC © del autor o autores / Todos los derechos reservados

(Deterministic Decomposable Negation Normal Form). The Max-SAT solution of the approximate formula, which can be computed very efficiently, can be used as a bound on the solution of the original problem. Once every variable involved in the relaxation is assigned a value, the solution of the conditioned approximate formula is no longer a bound, it becomes exact. Thus, Clone only needs to perform branch and bound search on the search space of those variables involved in the relaxation of constraints, resulting in a smaller search space. For further details see [PD07].

– **Lazy (Teresa Alsinet, Felip Manyà and Jordi Planes):** It is a branch and bound solver for both Max-SAT and Weighted Max-SAT that uses very simple lazy data structures and a static variable selection heuristic. As preprocessing it applies a refinement of binary resolution (replaces $x \vee D_1$ and $\neg x \vee D_2$ with $D_1 \vee D_2$). The initial upper bound is computed with a local search algorithm. At each node of the proof tree it applies the complementary unit clause rule, and applies unit propagation whenever the difference between the lower bound and the upper bound is one. Lazy implements the star rule as lower bound computation method (it increments the lower bound by one for every detected disjoint subset of the form $x, y, \neg x \vee \neg y$). For further details see [AMP08].

– **LB-SAT (Han Lin and Kaile Su):** LB-SAT is a two-stage solver for MAX-SAT. At the first stage, it invokes a local search procedure to calculate an approximate optimal solution. At the second stage, taking the approximate value as an initial upper bound, a branch and bound routine is called to find the exact solution. At each search node, like UP and Maxsatz, LB-SAT exploits unit propagation to compute a lower bound. The lower bound is computed in an incremental style; i.e., at each node, instead of computing the lower bound from scratch, LB-SAT reuses the information from the previous search nodes to boost the computation and improve the lower bound. Other techniques incorporated into LB-SAT can be found in [LS07].

– **Maxsatz (Chu Min Li, Felip Manyà and Jordi Planes):** It is a branch and bound solver for Max-SAT that incorporates into a Max-SAT solver some of the technology developed for Satz. At each node of the proof tree it transforms the formula into an equivalent formula that preserves the number of unsatisfied clauses by applying some efficient refinements of unit resolution that the authors have defined for Max-SAT (e.g., it replaces $x, y, \neg x \vee \neg y$ with $\square, x \vee y$, it replaces $x, \neg x \vee y, \neg x \vee z, \neg y \vee \neg z$ with $\square, \neg x \vee y \vee z, x \vee \neg y \vee \neg z$). MaxSatz implements a lower bound computation method that consists in incrementing the lower bound by one for every disjoint inconsistent subset that can be detected by unit propagation. Moreover, the lower bound computation method is enhanced with failed literal detection. The variable selection heuristics takes into account the number of positive and negative occurrences in binary and ternary clauses. For further details see [LMP06, LMP07].

Copia gratuita. Personal free copy http://libros.csic.es

– **MaxSatz14 (Sylvain Darras, Gilles Dequen, Laure Devendeville and Chu Min Li):** This solver is based on the last release of Maxsatz solver, built and improved by Chu Min Li, Felip Manyà and Jordi Planes. The main contribution has been to speed up the two look-ahead functions by selecting and storing useful conflictual subformulas in order to avoid their recomputation at each node of the search tree. For further details see [DDDL07].

– **MiniMaxsat (Federico Heras, Javier Larrosa and Albert Oliveras):** MiniMaxsat incorporates the best current SAT and Max-SAT techniques. It can handle hard clauses (clauses of mandatory satisfaction as in SAT), soft clauses (clauses whose falsification is penalized by a cost as in Max-SAT) as well as pseudo-Boolean objective functions and constraints. Its main features are: learning and backjumping on hard clauses; resolution-based and subtraction-based lower bounding; and lazy propagation with the two-watched literals schema. For further details see [HLO07].

– **SAT4Jmaxsat (Daniel Le Berre):** SAT4Jmaxsat translates a Max-SAT instance $S = \{C1, C2, \ldots, Cm\}$ with n variables into the following optimization problem: For each clause C_i in the original problem, a new variable V_i is created and added. Some people call those variables selector variables because satisfying such a variable disables a clause. So solving Max-SAT on the original problem is equivalent to solve the optimization problem: Minimize the number of V_i's satisfied in $S' = \{C_1 \vee V_1, C_2 \vee V_2, \ldots, C_n \vee V_n\}$. Since SAT4Jmaxsat supports cardinality constraints, it simply asks a SAT solver to solve S', and each time a model M is found, it tries to find a better one, by adding a cardinality constraints $SUM(V_i) < number\_of\_V_i\_satisfied\_in\_M$. Once S' and all the cardinality constraints are inconsistent, the latest model is the optimal solution. For further details see [Ber].

– **PMS (Josep Argelich and Felip Manyà):** PMS is the version of our Partial Max-SAT solver described in Section 5.5.1.

– **SR(w) (Miquel Ramírez and Héctor Geffner):** SR(w) is a Min-CostSAT solver which uses explicit structural relaxation to derive lower bounding functions that allow a Branch & Bound DLL-style search procedure to potentially prune vast tracts of the search space. SR(w) is built on top of two off-the-shelf tools: the d-SDNNF compiler by Darwiche [Dar] and the state-of-the-art SAT solver MiniSAT 2.0 [ES]. For further details see [RG07].

– **ToolBar (Simon de Givry, Federico Heras, Javier Larrosa and Thomas Schiex):** A DPLL-like algorithm is used to find a better solutions or proving optimality. After each assignment the current subproblem is transformed to an equivalent (and simpler) one. The transformations are based on the resolution rule for Max-SAT [LH05a]. Note that these

© CSIC © del autor o autores / Todos los derechos reservados

transformations can be explained as different levels of local consistency for WCSP. It is easy to see that a Max-SAT instance can be represented as a WCSP problem where all variables have two values (Boolean variables) and forbidden tuples represent weighted clauses. Examples of such transformations are (and its related WCSP local consistencies): (i) clauses $(x \vee y, 2), (\neg x \vee y, 1)$ are replaced by $(x \vee y, 1), (y, 1)$ (This is detected by AC* in WCSP). (ii) clauses $(x, 1), (\neg x \vee y, 2), (\neg y \vee z, 1), (\neg z, 1)$ are replaced by $(\square, 1), (\neg x \vee y, 1), (x \vee \neg y, 1), (y \vee \neg z, 1)$ where $(\square, 1)$ represents an increment of the lower bound (this is detected by EDAC* in WCSP).

– **W-MaxSatz (Josep Argelich, Chu Min Li and Felip Manà):** W-MaxSatz is the version of our solver described in Section 5.5.2.

These are the solving techniques that were identified as powerful and promising:

- Resolution-style inference rules that transform Max-SAT instances into equivalent Max-SAT instances have a dramatic impact on the the performance profile of solvers. Solvers implementing powerful inference rules include MaxSatz, MaxSatz14, MiniMaxsat, Toolbar and W-MaxSatz.

- Despite the dramatic improvements achieved by applying inference rules, the computation of good quality underestimations of the lower bound is decisive to speed up solvers. The two more powerful techniques that have been identified are the detection of disjoint inconsistent subsets of clauses via unit propagation and failed literal detection (LB-SAT, MaxSatz, MaxSatz14, MiniMaxsat, PMS, W-MaxSatz), and transforming the Max-SAT instance into a minimum cardinality instance and solving a relaxation of this new instance after compiling it with a d-DNNF compiler (Clone, SR(w)).

- Learning of hard clauses produces significant performance improvements on several types of Partial Max-SAT instances.

- The selection of suitable data structures is decisive for producing fast implementations. Solvers incorporating lazy data structures include ChaffBS, ChaffLS, Lazy, MiniMaxsat and SAT4Jmaxsat.

- The formalism used to encode problems has a remarkable impact on performance. When there are hard and soft constraints, the Partial Max-SAT formalism allows to exploit structural information.

3.7 Summary

We have presented an overview about Max-SAT and reviewed the solving techniques that have proved to be useful in terms of performance. This is the basis for all discussions in the rest of the thesis, with special attention to the advances

Copia gratuita. Personal free copy http://libros.csic.es

on combining techniques for SAT and Max-SAT for problems with hard and soft constraints, new learning schemas for soft clauses, preprocessing techniques, and efficient implementations of Partial Max-SAT solvers.

© CSIC © del autor o autores / Todos los derechos reservados

Chapter 4

The Soft-SAT formalism

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 thesis we will consider that all the constraints are *crisp* (i.e., they are either completely satisfied or completely violated), but we distinguish between *hard* constraints (i.e., they must be satisfied by any solution) and *soft* constraints (i.e., they can be violated by some solutions). 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 [MBB$^+$03, MRS06] for surveys about constraint programming approaches for 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 truth assignment satisfies a constraint if it satisfies all those clauses, and violates a constraint if it falsifies at least one of those clauses. Thus, in contrast to the usual approach, the concept of satisfaction for SAT-encoded over-constrained problems that we propose in this chapter refers to blocks of clauses instead of individual clauses. This leads 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 chapter we present a new generic problem solving approach for over-constrained problems based on Max-SAT. We first define a Boolean clausal form formalism that deals with blocks of clauses instead of individual clauses, and

Copia gratuita. Personal free copy http://libros.csic.es

that allows one to declare a block of *hard* clauses (i.e., it must be satisfied by any solution) and several blocks of *soft* clauses (i.e., they can be violated by some solutions). 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 the hard block of clauses and maximizes the number of satisfied soft blocks. Our solvers are branch and bound algorithms equipped with original lazy data structures, powerful inference techniques, good quality lower bounds, and original variable selection heuristics. Finally, we report an experimental investigation on a representative sample of instances (random Soft-2-SAT, Max-CSP, graph coloring, pigeon hole, and quasigroup completion) which provides experimental evidence that our approach is competitive compared with the state-of-the-art approaches developed in the CSP and SAT communities.

Solving over-constrained problems with Max-SAT local search algorithms was investigated in [JKS95, CIKM97]. 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 our best knowledge, the treatment of soft constraints with exact Max-SAT solvers was not considered before the introduction of Soft-SAT.

It is important to point out that our work on the Soft-SAT formalism was done before the Max-SAT community was interested in developing exact Partial Max-SAT solvers, which also capture in a natural way the distinction between hard and soft constraints, and exploit this distinction. Nevertheless, we would like to highlight some advantages of Soft-SAT over Partial Max-SAT:

1. It allows to represent over-constrained problems in a more natural and compact way.

2. It provides to the user information about constraint violations in a more intuitive way.

3. It allows us to get more propagation at certain nodes (this point is discussed in Section 4.2).

4. It allows to define variable selection heuristics that exploit the fact that a variable occurs in a hard or in a soft block.

5. It allows to identify clauses of violated blocks that are not relevant for further checks. Once a clause in a block is violated, the remaining clauses of the block are ignored.

6. It avoids to use auxiliary variables. The idea of blocks is usually captured in Partial Max-SAT with the introduction of auxiliary variables. Since most inference rules in Partial Max-SAT deal with short clauses, the introduction of auxiliary variables delays the application of inference techniques and slows the identification of optimal solutions.

© CSIC © del autor o autores / Todos los derechos reservados

The chapter is structured as follows. In Section 4.1 we introduce the formalism of soft CNF formulas. In Section 4.2 we describe solvers for soft CNF formulas with both static and dynamic variable selection heuristics. In Section 4.3 we report the experimental investigation we performed to assess the performance of our formalism and solvers.

4.1 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 problems with hard and soft constraints.

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 *or* s_i *for some* $i \in \mathbb{N}$*. The hard block of a soft CNF formula is formed by all the pairs* $(clause, label)$ *with label* h*, 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 s_i model the same soft constraint.

Definition 2 *A truth assignment satisfies the 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 the hard block of* ϕ*. We say then that* ϕ *is satisfiable. A soft CNF formula* ϕ *is unsatisfiable if there is no truth assignment that satisfies the hard block 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 the hard block 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 4.1 *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* $\{(v_1, v_2), (v_1, v_3), (v_2, v_3)\}$ *(see Figure 4.1), that problem is encoded as a Soft-SAT instance as follows:*

(i) The set of propositional variables is $\{x_1^1, x_1^2, x_2^1, x_2^2, x_3^1, x_3^2\}$*; the intended meaning of variable* x_i^j *is that vertex* v_i *is colored with color* j*.*

(ii) The hard block is formed by the following at-least-one and at-most-one clauses:

$$(x_1^1 \vee x_1^2, h), (\neg x_1^1 \vee \neg x_1^2, h), (x_2^1 \vee x_2^2, h), (\neg x_2^1 \vee \neg x_2^2, h), (x_3^1 \vee x_3^2, h), (\neg x_3^1 \vee \neg x_3^2, h).$$

Copia gratuita. Personal free copy http://libros.csic.es

(iii) There is a soft block for every edge:

$$(\neg x_1^1 \vee \neg x_2^1, s_1), (\neg x_1^2 \vee \neg x_2^2, s_1),$$
$$(\neg x_1^1 \vee \neg x_3^1, s_2), (\neg x_1^2 \vee \neg x_3^2, s_2),$$
$$(\neg x_2^1 \vee \neg x_3^1, s_3), (\neg x_2^2 \vee \neg x_3^2, s_3).$$

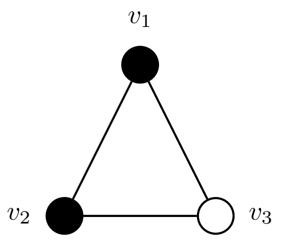


Figure 4.1: Graph coloring example with three vertices and two colors.

4.2 Soft-SAT algorithms

We start by describing how a basic Soft-SAT solver works. Based on that description, we then introduce the two solvers we have designed and implemented: Soft-SAT-S and Soft-SAT-D. Soft-SAT-S uses static variable selection heuristics while Soft-SAT-D uses dynamic variable selection heuristics.

4.2.1 A basic Soft-SAT algorithm

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. Figure 4.2 shows this tree representation.

A basic Soft-SAT solver explores that search tree following a depth-first branch and bound schema. At each node, the algorithm backtracks if the current partial assignment violates some clause of the hard block, and applies the unit clause rule to the literals that occur in unit clauses of hard block; i.e., given a literal $\neg x$ (x), it deletes all the clauses containing the literal $\neg x$ (x) and removes all the occurrences of the literal x ($\neg x$). If the current partial assignment does not violate any clause of the hard block, the algorithm compares the number of soft blocks falsified by the best complete assignment found so far, called upper bound (UB), with the number of soft blocks falsified 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

© CSIC © del autor o autores / Todos los derechos reservados

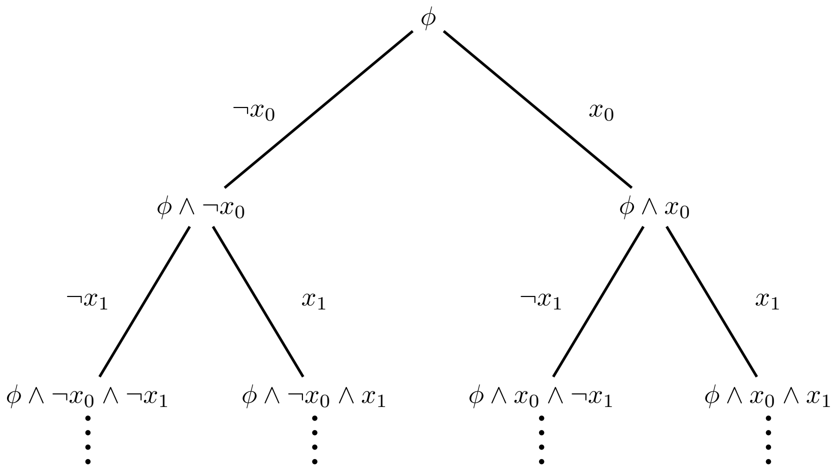

ϕ
$\neg x_0$
x_0
$\phi \wedge \neg x_0$
$\phi \wedge x_0$
$\neg x_1$
x_1
$\neg x_1$
x_1
$\phi \wedge \neg x_0 \wedge \neg x_1$
$\phi \wedge \neg x_0 \wedge x_1$
$\phi \wedge x_0 \wedge \neg x_1$
$\phi \wedge x_0 \wedge x_1$

Figure 4.2: Search tree of a soft CNF formula ϕ.

tree. If $UB > LB$, it extends the current partial assignment by instantiating one more variable, say x, which leads to the creation of two branches from the current branch: the left branch corresponds to instantiating x to false, and the right branch corresponds to instantiating x 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 unit clause rule using the literal $\neg x$ (x). The value that UB takes after exploring the entire search tree is the minimum number of soft blocks that are falsified by a complete assignment. The algorithm 4.1 shows the pseudocode of a basic Soft-SAT solver.

4.2.2 Soft-SAT-S

Soft-SAT-S implements the basic Soft-SAT solver augmented with a number of improvements that we describe below.

Upper bound and lower bound computation

In Soft-SAT-S, as in modern Max-SAT solvers, the initial upper bound is obtained with a GSAT-like [SLM92] local search algorithm. The search begins with a randomly generated complete truth assignment and, at each step, the value of one variable is flipped taking into account its score. The score of a variable is the sum of weights that we associate with unsatisfied clauses; we associate a weight one to an unsatisfied clause of a soft block and a weight equal to the number of clauses to an unsatisfied clause of the hard block. Local minima are avoided by occasionally performing a random walk.

Copia gratuita. Personal free copy http://libros.csic.es

Algorithm 4.1: `Soft-SAT-Basic`(ϕ, UB) : Basic Soft-SAT solver

```
Output: The minimum number of blocks of the soft CNF formula $\phi$ that
        are falsified by an assignment
Function Soft-SAT-Basic ($\phi$ : soft CNF formula, $UB$ : upper bound) :
Natural
    if not hard block is satisfied then
        return $\infty$
    $\phi \leftarrow$ HardUnitPropagation($\phi$)
    if $UB$ = LowerBound($\phi$)+1 then
        $\phi \leftarrow$ SoftUnitPropagation($\phi$)
    if hard block is satisfied and $UB$>LowerBound($\phi$) then
        if $\phi = \emptyset$ or $\phi$ only contains empty blocks then
            return EmptyBlocks($\phi$)
        $x \leftarrow$ SelectVariable($\phi$)
        $UB \leftarrow$ Min($UB$, Soft-SAT-Basic($\phi_{\neg x}$, $UB$))
        return Min($UB$, Soft-SAT-Basic($\phi_x$, $UB$))
    else
        return $\infty$
```

In branch and bound Max-SAT solvers incorporating the lower bound computation method based on inconsistency counts [WF96], the lower bound is the sum of the number of clauses unsatisfied 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. Such an underestimation is calculated taking into account the inconsistency counts of the variables not yet instantiated; i.e., the underestimation is the number of disjoint inconsistent subformulas formed by a unit clause with a literal ℓ and a unit clause with the complementary literal of ℓ.

For instances with CSP variables with domain size greater than two, we defined an original lower bound for soft CNF formulas that incorporates an underestimation of the number of soft blocks that will become unsatisfied if we extend the current partial assignment into a complete assignment. Each CSP variable with a domain of size k is represented by a set of k Boolean variables $x_1, \ldots, x_k$ in a SAT encoding. The inconsistency count associated with a Boolean variable x_i ($1 \leq i \leq k$) is the number of soft blocks violated when x_i is set to true. The inconsistency count associated with a CSP variable X, which is encoded by the Boolean variables $x_1, \ldots, x_k$, is the minimum of the inconsistency counts of x_i ($1 \leq i \leq k$). As underestimation for the lower bound, we consider exactly one CSP variable for each soft block and take the sum of the inconsistency counts of such variables. If we consider more than one CSP variable for each soft block, the inconsistency of a soft block can be computed more than once, and lead to a wrong lower bound.

© CSIC © del autor o autores / Todos los derechos reservados

Example 4.2 *Let ϕ be the following soft CNF formula:*

$$\begin{gathered}
(x_1^1 \vee x_1^2 \vee x_1^3, h), (\neg x_1^1 \vee \neg x_1^2, h), (\neg x_1^1 \vee \neg x_1^3, h), (\neg x_1^2 \vee \neg x_1^3, h),\\
(x_2^1 \vee x_2^2 \vee x_2^3, h), (\neg x_2^1 \vee \neg x_2^2, h), (\neg x_2^1 \vee \neg x_2^3, h), (\neg x_2^2 \vee \neg x_2^3, h),\\
(x_3^1 \vee x_3^2 \vee x_3^3, h), (\neg x_3^1 \vee \neg x_3^2, h), (\neg x_3^1 \vee \neg x_3^3, h), (\neg x_3^2 \vee \neg x_3^3, h),\\
(\neg x_1^1, s_1), (\neg x_2^1, s_1), (\neg x_3^2, s_1), (\neg x_1^1 \vee \neg x_2^2, s_1), (x_3^2 \vee x_1^3, s_1),\\
(\neg x_1^3, s_2), (\neg x_2^2, s_2), (\neg x_3^3, s_2), (\neg x_1^3, s_2), (x_2^1 \vee \neg x_3^1, s_2),\\
(\neg x_1^2, s_3), (\neg x_2^1, s_3), (\neg x_3^3, s_3), (x_1^1 \vee x_2^3, s_3).
\end{gathered}$$

Each CSP variable x_i $(1 \le i \le 3)$ with domain of size 3 is represented by the set of Boolean variables x_i^1, x_i^2 and x_i^3. The inconsistency counts associated with each variable are the following:

CSP variables	*Boolean variables*			$ic(x_i)$
x_1	$x_1^1 = 1$	$x_1^2 = 1$	$x_1^3 = 1$	1
x_2	$x_2^1 = 2$	$x_2^2 = 1$	$x_2^3 = 0$	0
x_3	$x_3^1 = 0$	$x_3^2 = 1$	$x_3^3 = 2$	0

The sum of the inconsistency counts of each CSP variable is 1; i.e., we can increase the lower bound by 1.

Inference

When branching is done, algorithms for Max-SAT like MaxSatz [LMP07], Max-Solver [XZ05] and MiniMaxsat [HLO07], apply the unit clause rule (simplifying with the branching literal) instead of applying unit propagation as in the Davis-Putnam-style [DLL62] 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 $\{x, \neg y, \neg x \vee y, \neg x\}$ using the unit clause $\neg x$, we derive one empty clause while if we use the unit clause x, 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 [BF99]. Soft-SAT-S performs unit propagation in that case too.

Moreover, as pointed out in the description of the basic Soft-SAT solver, Soft-SAT-S applies the unit clause rule when a clause of the hard block becomes unit. This propagation, which leads to substantial performance improvements, cannot be safely applied in Max-SAT solvers, and is a key feature of our approach, as well as of Partial Max-SAT solvers.

Data structures

Since Soft-SAT-S uses static variable selection heuristics, we were able to implement extremely simple and 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 either unit or empty. Thus, if we have a clause with four variables, we do not perform any

Copia gratuita. Personal free copy http://libros.csic.es

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 have also a pointer to the last variable of the clause. When a variable x is fixed to true (false), only the clauses whose penultimate variable in the static order is $\neg x$ (x) are evaluated. This approach has two advantages: the cost of backtracking is constant because we do not have to undo pointers like in adjacency lists and, at each step, we evaluate a minimum number of clauses.

Example 4.3 *Given the clause $\neg x_1 \vee x_2 \vee x_3 \vee \neg x_4 \vee x_5$ and the static order $x_5 = false, x_4 = true, x_3 = false, x_2 = false, x_1 = true$, the solver performs the following steps:*

1. *Initially, all the literals are free.*

2. *It does nothing when the solver instantiates the variables $x_5 = false$, $x_4 = true$ and $x_3 = false$ because these variables neither are referenced as penultimate nor are the last variable of the clause.*

3. *When it instantiates $x_2 = false$, it follows the pointer to the penultimate literal x_2. Then, it checks all the instantiated literals of the clause. Since all the instantiated literals are falsified, the clause becomes unit and is added to the stack of unit clauses.*

4. *Finally, when it instantiates $x_1 = true$, it follows the pointer to the last literal $\neg x_1$ of the clause. Since the clause is unit, the clause becomes unsatisfied.*

Figure 4.3 shows the described steps.

In contrast to the lazy data structures used in Chaff [MMZ$^+$01], where a dynamic variable selection heuristic is used, we do not have to deal with two watched literals. It is enough to have a pointer to the penultimate variable, and a clause is not visited until that variable is instantiated.

Variable selection heuristics

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

- **MO:** It instantiates first the variables that appear Most Often (MO). Ties are broken using the lexicographical order.

© CSIC © del autor o autores / Todos los derechos reservados

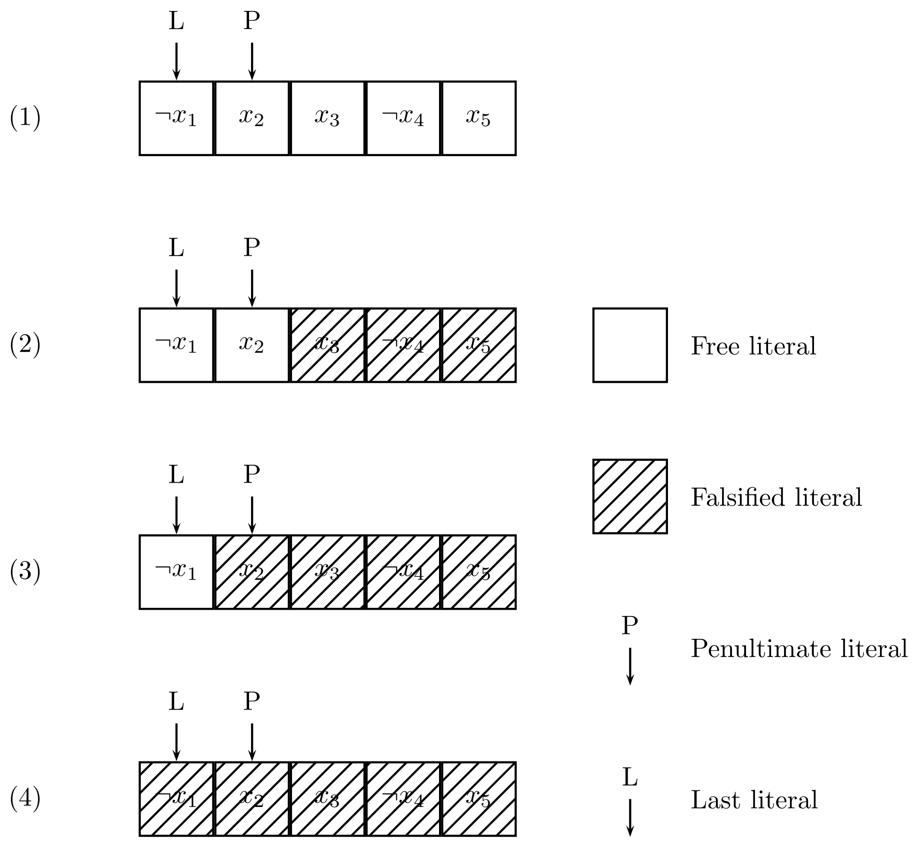


Figure 4.3: Data structures behaviour for static variable ordering.

- **MOH:** It instantiates first the variables that appear most often, but it takes into account if the variable occurs in the hard block or in a soft block. The score assigned to each variable is the number of occurrences in soft blocks plus five times the number of occurrences in the hard block. Ties are broken using the lexicographical order.

  It gives a bigger score to variables in the hard block because Soft-SAT-S applies the unit clause rule to the unit clauses of the hard block.

- **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 $x_1, \ldots, x_k$. This heuristic associates a weight to each one of these sets: the sum of the total number of occurrences of each variable of the set. It orders the sets according to such a weight. Heuristic **csp** instantiates, first and in lexico-

Copia gratuita. Personal free copy http://libros.csic.es

graphical 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 (e.g., Max-CSP, graph coloring, pigeon hole and quasigroup completion). 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 an n-ary branching.

4.2.3 Soft-SAT-D

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.

Data structures

The fact of having dynamic variable selection heuristic 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 zChaff [MMZ+01] described in Section 4.2.2. They are also lazy data structures, but are not so efficient because here we need to maintain the two watched literals.

We are only interested in knowing when a clause has become unit or empty. With a static order, we know which variables of a clause will be the penultimate and the last to be instantiated, but we do not have this information with a dynamic order. The two-watched literals data structure has two pointers to literals per clause, but they are not fixed as in Soft-SAT-S. We denote by p_1 and p_2 the two pointers to literals l_{p_1} and l_{p_2}, respectively. When we instantiate the literal referenced by one of this pointers, say l_{p_1}, we can observe three cases:

1. l_{p_2} is a true literal: The clause is satisfied; we do nothing.
2. l_{p_2} is a false literal: We check if all the literals of the clause are falsified. If they are, the clause is unsatisfied and we update the lower bound, but we do nothing with the two-watched literals data structures.
3. l_{p_2} is a free literal: In this case not all the literals of the clause are instantiated. We will search another free literal to know whether the clause is unit. If l_{p_2} is the only free literal and the clause has no true literals, it becomes unit. Otherwise, if we found another free literal, we move p_1 on it.

Example 4.4 *Given the clause $\neg x_1 \vee x_2 \vee x_3 \vee \neg x_4 \vee x_5$, the dynamic order $x_2 = false, x_4 = true, x_1 = true, x_5 = false, x_3 = false$, and the pointers p_1 on $\neg x_1$ and p_2 on x_3, the solver performs the following steps:*

© CSIC © del autor o autores / Todos los derechos reservados

1. *Initially, all the literals are free.*

2. *It does nothing for the instantiation of $x_2 = false$ and $x_4 = true$ because they are not watched literals.*

3. *When it instantiates $x_1 = true$, it has to move the reference of $\neg x_1$ to another free litereal, if it exists. There is the literal x_5 that is not referenced by any pointer. Then, it moves the pointer of $\neg x_1$ to x_5.*

4. *The next variable to instantiate is $x_5 = false$. It has to move the reference to x_5 but only one free literal remains and is referenced by the other pointer. As all the literals of the clause but one are falsified, the clause becomes unit.*

5. *Finally, it instantiates $x_3 = false$. The pointer to x_5 references a falsified literal. It checks that all the literals of the clause are falsified and deduces that the clause is empty.*

This process is shown graphically in Figure 4.4.

The main advantage of two-watched literals with respect to other pointer based data structures (e.g., the data structures of SATO [ZS96]) is that the time complexity of backtracking is constant. We do not need to restore the watched literals to previous positions when backtracking because the watched literals are the last literals to be instantiated.

Variable selection heuristics

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

- **MO:** It instantiates first the variables that appear Most Often (MO). Ties are broken using the lexicographical order. Observe that it does not use the variable that appears most often in minimum size clauses (MOMS heuristic) because it is difficult to know the current size of a clause with two-watched literals data structures. However, most of the instances which were solved in the experimental investigation contain a significant number of binary clauses.

- **MO-csp:** This is the dynamic version of heuristic *csp* of Soft-SAT-S. It associates 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. It selects the set with the highest weight and instantiates its variables in lexicographical order. We emulate an n-ary branching as in heuristic *csp*.

Copia gratuita. Personal free copy http://libros.csic.es

4.3 Experimental investigation

We now report the experimental investigation we conducted to evaluate the performance of our problem solving approach. We used a representative sample of instances (random Soft-2-SAT, Max-CSP, graph coloring, pigeon hole and quasigroup completion), and compared our solvers with the best performing state-of-the-art solvers for over-constrained problems developed in the CSP and SAT communities.

All the experiments of this section were performed on a cluster with the following specification:

- Number of hosts: 4
- Operating System: Rocks Cluster 4.0.0 (Linux 2.6.9)
- Processor: Intel(R) Pentium(R) 4 CPU, 2 GHz
- Memory: 1 GB
- Cache: 512 KB

4.3.1 Solvers

The solvers used are the following ones:

- Soft-SAT-S with heuristics MO, MOH and csp.
- Soft-SAT-D with heuristics MO and MO-csp.
- WMax-SAT: It is a weighted Max-SAT solver that we have implemented. WMax-SAT uses the code of Soft-SAT but does not take into account the notion of hard and soft block. Conceptually, WMax-SAT is like Soft-SAT but every clause is treated as a different soft block. The lower bound of WMax-SAT is different from the lower bound of Soft-SAT; the underestimation is calculated taking into account the inconsistency counts of the variables not yet instantiated. WMax-SAT incorporates the following variable selection heuristic: It instantiates the variables taking into account the number of occurrences in decreasing order (MO). We use this solver in the experimental investigation to check if the performance of Soft-SAT solvers is due either to the formalism and its techniques or to the engineering of the solver.
- PFC-MPRDAC [LM99]: This is a highly optimized solver from the Constraint Programming community for solving binary Max-CSP problems. It is the successor of PFC-RDAC and PFC-MRDAC [LMS99] augmented by adding to the lower bound global contributions of disjoint subsets of unassigned variables, which requires a partition of the set of unassigned variables.

© CSIC © del autor o autores / Todos los derechos reservados

Toolbar [LHdG08]: It is a DPLL-like algorithm used to find optimal solutions. The branching Heuristic is similar to two-sided Jeroslow-Wang heuristic. After each assignment, the current subproblem is transformed into an equivalent (and simpler) one. The transformations applied by Toolbar are chain resolution and cycle resolution, that are based on the resolution rule for Max-SAT. We used version 3.0.

- Toolbar-CSP [LHdG08]: This is the Toolbar 3.0 version that works as a weighted Max-CSP solver. Toolbar includes several algorithms maintaining some form of local consistency for solving weighted CSP: node, arc, directional arc and full directional arc consistencies, as well as algorithms such as tree decomposition, bucket elimination, dominance testing and singleton arc consistency which are part of this platform. We refer to this option as Toolbar-CSP, while when we say Toolbar we refer to the option in which the solver works as a weighted Max-SAT solver.

For this experimental investigation, we also considered two of the best performing weighted Partial Max-SAT solvers in the last Max-SAT Evaluation (Max-SAT 2007):

- MiniMaxsat [HLO07]: It is a branch and bound solver that can handle hard clauses (clauses of mandatory satisfaction as in SAT), and soft clauses (clauses whose falsification is penalized by a cost as in Max-SAT) as well as pseudo-Boolean objective functions and constraints. Its main features are: learning and backjumping on hard clauses; resolution-based and subtraction-based lower bounding; and lazy propagation with the two-watched literals schema. MiniMaxsat was the best performing solvers on structured benchmarks in Max-SAT 2007 in the categories of weighted and unweighted Partial Max-SAT.

- W-MaxSatz: W-MaxSatz is an extension of MaxSatz [LMP07] to solve weighted Partial Max-SAT. It is a branch and bound weighted Max-SAT solver that incorporates all the features of MaxSatz adapted to deal with weights. This implies the modification of the data structures to dynamically add and remove clauses without a significant overhead in CPU time. W-MaxSatz implements a dynamic variable selection heuristic, advanced inference rules and a lower bound based on unit propagation and failed literal detection. W-MaxSatz was the best performing solver on random benchmarks in Max-SAT 2007 in the categories of weighted and unweighted Partial Max-SAT.

4.3.2 Benchmarks and encodings

The benchmarks and the encodings used in the experimental investigation are described in detail below.

Copia gratuita. Personal free copy http://libros.csic.es

Random Soft-2-SAT instances

We generated random 2-SAT instances to which we then assigned, randomly and uniformly, a label corresponding to the hard block or to a soft block. The generator has as parameter the number of blocks: one block is declared to be hard and the rest of blocks are declared to be soft. Hard and soft blocks are intended to have the same number of clauses.

Max-CSP instances

We used SAT-encoded random binary CSPs and solved the Max-CSP problem. Max-CSP instances have a natural representation using the formalism of soft CNF formulas.

The instances were generated with a generator of uniform random binary CSPs[1] —designed and implemented by Frost, Bessière, Dechter and Regin— that implements the so-called model B [SD96]: in the class $\langle n, d, p_1, p_2 \rangle$ with n variables of domain size d, we choose a random subset of exactly $p_1 n(n-1)/2$ constraints (rounded to the nearest integer), each with exactly $p_2 d^2$ conflicts (rounded to the nearest integer); p_1 may be thought of as the *density* of the problem and p_2 as the *tightness* of constraints.

The instances were encoded using the support encoding [Kas90, Gen02]. The idea behind the encoding is to encode into clauses the *support* for a value instead of encoding conflicts. The support for a value j of a CSP variable X_i across a constraint is the set of values of the other variable in the constraint which allow $X_i = j$. If $v_1, v_2, \ldots, v_k$ are the supporting values of variable X_l for $X_i = j$, we add the clause $\neg x_{ij} \vee x_{lv_1} \vee x_{lv_2} \vee \cdots \vee x_{lv_k}$ (called *support* clause). There is one support clause for each pair of variables X_i, X_l involved in a constraint, and for each value in the domain of X_i. We need a similar clause in each direction, one for the pair X_i, X_l and one for X_l, X_i. Besides, we need to add the at-least-one and at-most-one clauses for each CSP variable to ensure that each CSP variable takes exactly one value of its domain. All the at-least-one and at-most-one clauses were encoded as the hard block, and each set of clauses that encodes a CSP constraint was encoded as a different soft block.

Example 4.5 *Let P be a CSP instance defined by $\langle X, D, C \rangle$, where $X = \{x_1, x_2, x_3\}$ is the set of variables, $d(x_1) = d(x_2) = d(x_3) = \{1, 2, 3\}$ is the domain for all the variables and $C = \{\langle \{x_1, x_2\}, x_1 \neq x_2 \rangle, \langle \{x_1, x_3\}, x_1 < x_3 \rangle\}$ is the set of constraints. P is encoded as a Soft-SAT instance as follows:*

(i) The set of propositional variables is $\{x_1^1, x_1^2, x_1^3, x_2^1, x_2^2, x_2^3, x_3^1, x_3^2, x_3^3\}$; the intended meaning of variable x_i^j is that variable i has assigned value j.

(ii) The hard block is formed by the following at-least-one and at-most-one clauses:

$$(x_1^1 \vee x_1^2 \vee x_1^3, h), (\neg x_1^1 \vee \neg x_1^2, h), (\neg x_1^1 \vee \neg x_1^3, h), (\neg x_1^2 \vee \neg x_1^3, h),$$
$$(x_2^1 \vee x_2^2 \vee x_2^3, h), (\neg x_2^1 \vee \neg x_2^2, h), (\neg x_2^1 \vee \neg x_2^3, h), (\neg x_2^2 \vee \neg x_2^3, h),$$
$$(x_3^1 \vee x_3^2 \vee x_3^3, h), (\neg x_3^1 \vee \neg x_3^2, h), (\neg x_3^1 \vee \neg x_3^3, h), (\neg x_3^2 \vee \neg x_3^3, h),$$

[1] http://www.lirmm.fr/~bessiere/generator.html

© CSIC © del autor o autores / Todos los derechos reservados

(iii) There is a soft block for every constraint:

$$\begin{gathered}(\neg x_1^1 \vee x_2^2 \vee x_2^3, s_1), (\neg x_1^2 \vee x_2^1 \vee x_2^3, s_1), (\neg x_1^3 \vee x_2^1 \vee x_2^2, s_1),\\ (\neg x_2^1 \vee x_1^2 \vee x_1^3, s_1), (\neg x_2^2 \vee x_1^1 \vee x_1^3, s_1), (\neg x_2^3 \vee x_1^1 \vee x_1^2, s_1),\\ (\neg x_1^1 \vee x_3^2 \vee x_3^3, s_2), (\neg x_1^2 \vee x_3^3, s_2), (\neg x_1^3, s_2),\\ (\neg x_3^1, s_2), (\neg x_3^2 \vee x_1^1, s_2), (\neg x_3^3 \vee x_1^1 \vee x_1^2, s_2).\end{gathered}$$

Notice that, if we would like to encode this instance into Partial Max-SAT, we should introduce auxiliary variables because there are two violated clauses for every violated constraint. The Partial Max-SAT encoding would be as follows:

$$\begin{gathered}\left[x_1^1 \vee x_1^2 \vee x_1^3\right], \left[\neg x_1^1 \vee \neg x_1^2\right], \left[\neg x_1^1 \vee \neg x_1^3\right], \left[\neg x_1^2 \vee \neg x_1^3\right],\\ \left[x_2^1 \vee x_2^2 \vee x_2^3\right], \left[\neg x_2^1 \vee \neg x_2^2\right], \left[\neg x_2^1 \vee \neg x_2^3\right], \left[\neg x_2^2 \vee \neg x_2^3\right],\\ \left[x_3^1 \vee x_3^2 \vee x_3^3\right], \left[\neg x_3^1 \vee \neg x_3^2\right], \left[\neg x_3^1 \vee \neg x_3^3\right], \left[\neg x_3^2 \vee \neg x_3^3\right],\\ (\neg x_1^1 \vee x_2^2 \vee x_2^3 \vee s_1), (\neg x_1^2 \vee x_2^1 \vee x_2^3 \vee s_1), (\neg x_1^3 \vee x_2^1 \vee x_2^2 \vee s_1),\\ (\neg x_2^1 \vee x_1^2 \vee x_1^3 \vee \neg s_1), (\neg x_2^2 \vee x_1^1 \vee x_1^3 \vee \neg s_1), (\neg x_2^3 \vee x_1^1 \vee x_1^2 \vee \neg s_1),\\ (\neg x_1^1 \vee x_3^2 \vee x_3^3 \vee s_2), (\neg x_1^2 \vee x_3^3 \vee s_2), (\neg x_1^3 \vee s_2),\\ (\neg x_3^1 \vee \neg s_2), (\neg x_3^2 \vee x_1^1 \vee \neg s_2), (\neg x_3^3 \vee x_1^1 \vee x_1^2 \vee \neg s_2).\end{gathered}$$

We add an auxiliary variable for each soft constraint, which is positive in one direction of its support clauses and negative in the other direction. This is so because there is one violated clause in each direction of each violated constraint.

Graph coloring instances

We used unsatisfiable graph coloring instances and the problem we solved was to find a coloring that minimizes the number of adjacent vertices with the same color. We used individual instances from the Graph Coloring Symposium celebrated at CP-2002[2], and randomly generated instances using the generator of Culberson [Cul]. We used the generator with option IID (independent random edge assignment). The parameters of the generator are: number of vertices (n), optimum number of colors to get a valid coloring (k), and number of colors we use to color the graph (c).

The set of clauses that encode that each vertex is colored with exactly one color forms the hard block of the Soft-SAT instance. For every two adjacent vertices, the set of clauses that encode that those vertices have different colors forms a soft block. We can see an example of a Soft-SAT encoded graph in Section 4.1.

Pigeon hole instances

Given $m+1$ pigeons and m holes, the problem we solved was to determine the minimum number of holes with more than one pigeon taking into account that there is at least one pigeon in each hole. The set of clauses that encode that each pigeon is assigned exactly to one hole, together with the set of clauses that encode that there is at least one pigeon in each hole, form the hard block of the

[2]http://mat.gsia.cmu.edu/COLORING02/

Copia gratuita. Personal free copy http://libros.csic.es

Soft-SAT instance. There is a soft block for each set of clauses that encode that two different pigeons cannot be in the same hole.

Example 4.6 *A pigeon hole instance with 3 pigeons and 2 holes is encoded as a Soft-SAT instance as follows:*

(i) The set of propositional variables is $\{x_1^1, x_1^2, x_2^1, x_2^2, x_3^1, x_3^2\}$*; the intended meaning of variable* x_i^j *is that pigeon* i *is in hole* j*.*

(ii) The hard block is formed by the following clauses:

$$\begin{gathered}(x_1^1 \vee x_1^2, h), (\neg x_1^1 \vee \neg x_1^2, h),\\ (x_2^1 \vee x_2^2, h), (\neg x_2^1 \vee \neg x_2^2, h),\\ (x_3^1 \vee x_3^2, h), (\neg x_3^1 \vee \neg x_3^2, h),\\ (x_1^1 \vee x_2^1 \vee x_3^1, h), (x_1^2 \vee x_2^2 \vee x_3^2, h).\end{gathered}$$

(iii) There is a soft block for every hole:

$$\begin{gathered}(\neg x_1^1 \vee \neg x_2^1, s_1), (\neg x_1^1 \vee \neg x_3^1, s_1), (\neg x_2^1 \vee \neg x_3^1, s_1),\\ (\neg x_1^2 \vee \neg x_2^2, s_2), (\neg x_1^2 \vee \neg x_3^2, s_2), (\neg x_2^2 \vee \neg x_3^2, s_2).\end{gathered}$$

Quasigroup completion instances

We considered unsatisfiable instances of the quasigroup (or Latin square) completion problem (QCP) that were generated as indicated in [GS97]. Given n colors, a quasigroup, or Latin square, is defined by an $n \times n$ table, where each entry has a color and where there are no repeated colors in any row or any column; n is called the *order* of the quasigroup. The problem of whether a partially colored quasigroup can be completed into a full quasigroup by assigning colors to the open entries of the table is called the QCP. The problem we solved was to minimize the number of violated row and column constraints in QCP instances. By a row (column) constraint we mean that no color is repeated in the same row (column).

The set of clauses that encode that each entry is colored with exactly one color plus the set of clauses that encode the preassigned colors form the hard block of the Soft-SAT instances. For each row (column), the clauses that encode the row (column) constraints form a soft block. Therefore, the total number of soft blocks is $2n$.

Example 4.7 *A QCP instance of order 2 with color 1 in position* $\{1,1\}$ *and holes in positions* $\{1,2\}$*,* $\{2,1\}$ *and* $\{2,2\}$ *is encoded as a Soft-SAT instance as follows:*

(i) The set of propositional variables is $\{x_{1,1}^1, x_{1,1}^2, x_{1,2}^1, x_{1,2}^2, x_{2,1}^1, x_{2,1}^2, x_{2,2}^1, x_{2,2}^2\}$*; the intended meaning of variable* $x_{i,j}^c$ *is that position i,j of the table has color c.*

© CSIC © del autor o autores / Todos los derechos reservados

(ii) The hard block is formed by the following clauses:

$$\begin{gathered}(x^1_{1,1}, h),\\ (x^1_{1,1} \vee x^2_{1,1}, h), (\neg x^1_{1,1} \vee \neg x^2_{1,1}, h),\\ (x^1_{1,2} \vee x^2_{1,2}, h), (\neg x^1_{1,2} \vee \neg x^2_{1,2}, h),\\ (x^1_{2,1} \vee x^2_{2,1}, h), (\neg x^1_{2,1} \vee \neg x^2_{2,1}, h),\\ (x^1_{2,2} \vee x^2_{2,2}, h), (\neg x^1_{2,2} \vee \neg x^2_{2,2}, h).\end{gathered}$$

(iii) There is a soft block for every row and column of the table:

$$\begin{gathered}(\neg x^1_{1,1} \vee \neg x^1_{1,2}, s_1), (\neg x^2_{1,1} \vee \neg x^2_{1,2}, s_1),\\ (\neg x^1_{2,1} \vee \neg x^1_{2,2}, s_2), (\neg x^2_{2,1} \vee \neg x^2_{2,2}, s_2),\\ (\neg x^1_{1,1} \vee \neg x^1_{2,1}, s_3), (\neg x^2_{1,1} \vee \neg x^2_{2,1}, s_3),\\ (\neg x^1_{1,2} \vee \neg x^1_{2,2}, s_4), (\neg x^2_{1,2} \vee \neg x^2_{2,2}, s_4).\end{gathered}$$

The formula can be simplified using the assigned positions of the table. In this example, the unit clause rule can be applied using the clause $(x^1_{1,1}, h)$.

4.3.3 Weighted Partial Max-SAT and Max-CSP encodings

All the benchmarks encoded as soft CNF formulas were also encoded as Boolean weighted Partial Max-SAT instances in order to compare our solvers with Boolean weighted Partial Max-SAT solvers. The encoding used is defined as follows: A soft block s_i formed by a set of clauses $\{C_1, \ldots, C_m\}$ is replaced with the set of clauses $\{(s_i; 1), (C_1 \vee \neg s_i; 2), \ldots, (C_m \vee \neg s_i; 2)\}$, where s_i is a new Boolean variable and 1 and 2 are weights associated with the clauses. The hard block of the soft CNF formula is encoded as the hard block in the weighted Partial Max-SAT instance. A solution of a soft CNF formula corresponds to a feasible solution of its weighted Partial Max-SAT encoding with the minimum sum of weights of unsatisfied clauses. Actually, with our encoding, the minimum sum of weights of unsatisfied clauses is identical to the minimum number of soft blocks that can be falsified by a truth assignment that satisfies the hard block.

Example 4.8 *Given the following soft CNF formula of Example 4.1:*

$$\begin{gathered}(x^1_1 \vee x^2_1, h), (\neg x^1_1 \vee \neg x^2_1, h),\\ (x^1_2 \vee x^2_2, h), (\neg x^1_2 \vee \neg x^2_2, h),\\ (x^1_3 \vee x^2_3, h), (\neg x^1_3 \vee \neg x^2_3, h),\\ (\neg x^1_1 \vee \neg x^1_2, s_1), (\neg x^2_1 \vee \neg x^2_2, s_1),\\ (\neg x^1_1 \vee \neg x^1_3, s_2), (\neg x^2_1 \vee \neg x^2_3, s_2),\\ (\neg x^1_2 \vee \neg x^1_3, s_3), (\neg x^2_2 \vee \neg x^2_3, s_3).\end{gathered}$$

We derive the following weighted Partial Max-SAT instance:

$$\begin{gathered}[x^1_1 \vee x^2_1], [\neg x^1_1 \vee \neg x^2_1],\\ [x^1_2 \vee x^2_2], [\neg x^1_2 \vee \neg x^2_2],\\ [x^1_3 \vee x^2_3], [\neg x^1_3 \vee \neg x^2_3],\\ (s_1; 1), (\neg x^1_1 \vee \neg x^1_2 \vee \neg s_1; 2), (\neg x^2_1 \vee \neg x^2_2 \vee \neg s_1; 2),\\ (s_2; 1), (\neg x^1_1 \vee \neg x^1_3 \vee \neg s_2; 2), (\neg x^2_1 \vee \neg x^2_3 \vee \neg s_2; 2),\\ (s_3; 1), (\neg x^1_2 \vee \neg x^1_3 \vee \neg s_3; 2), (\neg x^2_2 \vee \neg x^2_3 \vee \neg s_3; 2).\end{gathered}$$

Copia gratuita. Personal free copy http://libros.csic.es

The Max-CSP instances and the graph coloring instances were also encoded as binary CSPs using the format used by PFC-MPRDAC and Toolbar-CSP, which consists of defining a constraint network by means of a list of nogoods. We believe that it is important to compare our approach with the problem solving approach for over-constrained problems developed in the Constraint Programming community because they have worked more intensively on this topic [MRS06].

4.3.4 Experimental results

Experiments with random Soft-2-SAT instances

We compared Soft-SAT-S with heuristic MO, Soft-SAT-S with heuristic MOH, Soft-SAT-D with heuristic MO, Toolbar and WMax-SAT on random Soft-2-SAT instances.

Figure 4.5 shows the results for instances with 50 variables, with a number of clauses ranging from 200 to 430, where 20 clauses are in the hard block and the rest of clauses are randomly distributed among 100 soft blocks; Figure 4.6 shows the results for instances with a number of variables ranging from 50 to 100 and with 300 clauses, where 50 clauses are in the hard block and the rest of clauses are randomly distributed among 50 soft blocks; and Figure 4.7 shows the results for instances with 60 variables and 300 clauses, where the number of clauses in the hard block ranges from 10 to 50 and the rest of clauses are randomly distributed among 50 soft blocks. In Figure 4.7 we do not display the results for WMax-SAT because they were not competitive. In all the figures we give mean and median time, and each data point corresponds to the mean and median time needed to solve a set of 100 instances.

In Figure 4.5 and Figure 4.7, we observe that the best performing solver is Soft-SAT-S with heuristic MOH, while in Figure 4.6 is Soft-SAT-D with heuristic MO. It is worth mentioning the good behaviour of heuristic MOH that takes into account the distinction between variables appearing in the hard block and in soft blocks.

Figure 4.8 compares our best performing solver of Figure 4.5, Soft-SAT-S (MOH), with Toolbar and the state-of-the-art weighted Partial Max-SAT solvers W-MaxSatz and MiniMaxsat. We observe that the best performing solver is MiniMaxsat and that Soft-SAT-S and W-MaxSatz have a similar behaviour.

Experiments with Max-CSP instances

In this section we describe a number of experiments we performed on random binary CSP. In Table 4.1 we compare Soft-SAT-S without underestimation in the lower bound with Soft-SAT-S with underestimation for sets of 100 instances of a representative sample of Max-CSP instances. The first column shows the parameters given to the generator of random binary CSPs, and the remaining columns show the experimental results obtained. For each set we give the mean and median time needed to solve an instance of the set. The variable selection

© CSIC © del autor o autores / Todos los derechos reservados

heuristic used is csp. Table 4.2 shows the number of backtracks instead of the CPU time for the same instances. In both cases we observe that the fact of adding a lower bound of better quality leads to dramatic performance improvements. In the rest of the experimental investigation, all the results reported are with underestimation.

	Soft-SAT-S (with underestimation)		Soft-SAT-S (without underestimation)	
$\langle n, d, p_1, p_2 \rangle$	mean	median	mean	median
$\langle 10, 15, 45/45, 190/225 \rangle$	**12.25**	**10.31**	605.03	547.52
$\langle 12, 13, 60/66, 130/169 \rangle$	**17.94**	**16.21**	2256.91	2010.26
$\langle 13, 8, 78/78, 50/64 \rangle$	**12.51**	**11.28**	1028.91	973.41
$\langle 15, 10, 50/105, 75/100 \rangle$	**1.63**	**1.39**	77.54	57.97
$\langle 17, 5, 110/136, 18/25 \rangle$	**3.35**	**2.83**	394.82	343.61
$\langle 18, 5, 80/153, 18/25 \rangle$	**0.86**	**0.76**	53.64	46.52
$\langle 20, 5, 90/190, 18/25 \rangle$	**3.06**	**2.42**	406.53	378.00
$\langle 22, 6, 70/231, 28/36 \rangle$	**7.10**	**4.13**	910.97	493.15
$\langle 23, 4, 150/253, 12/16 \rangle$	**16.25**	**13.59**	4615.67	3797.04
$\langle 25, 3, 160/300, 7/9 \rangle$	**2.66**	**2.09**	142.80	112.51

Table 4.1: Comparison of Soft-SAT-S without underestimation and Soft-SAT-S with underestimation on Max-CSP instances. Time in seconds.

	Soft-SAT-S (with underestimation)		Soft-SAT-S (without underestimation)	
$\langle n, d, p_1, p_2 \rangle$	mean	median	mean	median
$\langle 10, 15, 45/45, 190/225 \rangle$	**2.619.160**	**2.257.644**	807.841.884	735.579.551
$\langle 12, 13, 60/66, 130/169 \rangle$	**3.432.624**	**3.005.897**	>2.000.000.000	>2.000.000.000
$\langle 13, 8, 78/78, 50/64 \rangle$	**2.450.851**	**2.129.608**	1.093.257.769	1.168.573.259
$\langle 15, 10, 50/105, 75/100 \rangle$	**339.848**	**267.922**	141.343.132	96.429.278
$\langle 17, 5, 110/136, 18/25 \rangle$	**611.488**	**521.378**	564.618.781	520.298.372
$\langle 18, 5, 80/153, 18/25 \rangle$	**175.118**	**145.393**	114.017.436	92.915.266
$\langle 20, 5, 90/190, 18/25 \rangle$	**681.346**	**516.087**	601.459.493	631.196.627
$\langle 22, 6, 70/231, 28/36 \rangle$	**1.750.568**	**934.992**	416.141.039	513.696.823
$\langle 23, 4, 150/253, 12/16 \rangle$	**2.513.565**	**2.075.907**	>2.000.000.000	>2.000.000.000
$\langle 25, 3, 160/300, 7/9 \rangle$	**424.359**	**318.227**	337.120.904	262.771.567

Table 4.2: Comparison of Soft-SAT-S without underestimation and Soft-SAT-S with underestimation on Max-CSP instances. The variable selection heuristic used is csp. Mean and median number of backtracks.

In the second experiment we compared Soft-SAT-S with heuristic csp with a version of Soft-SAT-S with heuristic csp in which we do not apply the unit clause rule to unit clauses that appear in the hard block. We generated sets

Copia gratuita. Personal free copy http://libros.csic.es

of 100 instances of random binary CSPs with 14 variables, domain size 8, 91 constraints and a number of nogoods ranging from 10 to 63. Figure 4.9 shows the experimental results obtained; we give mean time (upper plot) and median time (lower plot). We see that applying this inference technique that exploits the fact of knowing whether a variable belongs to the hard block leads to significant performance improvements. The same behavior was observed for the rest of Soft-SAT heuristics and solvers that we have developed.

In the third experiment we compared Soft-SAT-S with heuristic csp (it is the best performing Soft-SAT solver on Max-CSP instances), PFC-MPRDAC, Toolbar-CSP and WMax-SAT on Max-CSP instances. The results obtained are shown in Table 4.3. We observe that solvers Toolbar-CSP and PFC-MPRDAC, which are specialized on solving Max-CSP instances, are faster than Soft-SAT-S, but the weighted Partial Max-SAT approach is much worse. We do not display results with Toolbar because they are worse than the results of WMax-SAT. Even when our solver is not the best, the differences with weighted Partial Max-SAT are substantial.

	Soft-SAT-S		PFC-MPRDAC		Toolbar-CSP		WMax-SAT	
$\langle n, d, p_1, p_2 \rangle$	mean	median	mean	median	mean	median	mean	median
$\langle 10, 8, 45/45, 48/64 \rangle$	0.33	0.32	0.19	0.19	**0.05**	**0.05**	11.95	11.41
$\langle 12, 6, 66/66, 27/36 \rangle$	0.48	0.47	0.23	0.23	**0.06**	**0.06**	45.50	45.48
$\langle 14, 5, 91/91, 18/25 \rangle$	0.82	0.77	0.35	0.35	**0.12**	**0.11**	189	193
$\langle 16, 4, 120/120, 12/16 \rangle$	0.37	0.32	0.22	0.22	**0.10**	**0.9**	275	276
$\langle 18, 3, 153/153, 6/9 \rangle$	1.00	0.93	0.31	0.31	**0.04**	**0.04**	68.04	62.71
$\langle 15, 6, 60/105, 27/36 \rangle$	0.25	0.24	0.20	0.20	**0.03**	**0.03**	778	539
$\langle 18, 5, 80/153, 18/25 \rangle$	0.67	0.58	0.33	0.30	**0.05**	**0.04**	5383	3364
$\langle 20, 5, 70/190, 18/25 \rangle$	0.54	0.44	0.33	0.31	**0.03**	**0.03**	4701	2715
$\langle 14, 8, 91/91, 50/64 \rangle$	44.27	43.95	8.37	7.68	**3.02**	**2.91**	>7200	>7200
$\langle 23, 4, 200/253, 12/16 \rangle$	102	79.61	8.91	7.80	**1.71**	**1.54**	>7200	>7200

Table 4.3: Comparison of Soft-SAT-S, PFC-MPRDAC, Toolbar-CSP and WMax-SAT on Max-CSP instances. Time in seconds.

In the fourth experiment, whose results are shown in Table 4.4, we solved the same instances of the previous experiment with Soft-SAT-D with heuristic MO-csp and with Soft-SAT-D with heuristic MO in order to compare the n-ary branching with the binary branching. We see that the fact of using an n-ary branching allows us to solve the instances up to 3 times faster. Also observe that Soft-SAT-S (which also uses an n-ary branching) is about 2 times faster than Soft-SAT-D with heuristic MO-csp, and up to 6 times faster than Soft-SAT-D with heuristic MO.

In the last experiment with Max-CSP instances we compared our best Soft-SAT solver for this benchmark, Soft-SAT-S, with the weighted Partial Max-SAT solvers MiniMaxsat and W-MaxSatz. We solved the same instances of the previous experiments. The results are shown in Table 4.5, where we observe that the best performing solver is Soft-SAT-S in all the sets of tested instances. This shows that the Soft-SAT formalism is a good approach for this benchmark

© CSIC © del autor o autores / Todos los derechos reservados

	Soft-SAT-D (MO-csp)		Soft-SAT-D (MO)	
$\langle n,d,p_1,p_2\rangle$	mean	median	mean	median
$\langle 10,8,45/45,48/64\rangle$	**0.69**	**0.68**	1.83	1.76
$\langle 12,6,66/66,27/36\rangle$	**1.20**	**1.11**	2.92	2.66
$\langle 14,5,91/91,18/25\rangle$	**2.55**	**2.33**	6.82	6.27
$\langle 16,4,120/120,12/16\rangle$	**2.69**	**2.54**	5.44	5.11
$\langle 18,3,153/153,6/9\rangle$	**0.69**	**0.65**	1.40	1.28
$\langle 15,6,60/105,27/36\rangle$	**1.16**	**1.01**	2.21	1.92
$\langle 18,5,80/153,18/25\rangle$	**2.97**	**2.48**	5.98	4.38
$\langle 20,5,70/190,18/25\rangle$	**1.85**	**1.52**	3.80	2.82

Table 4.4: Comparison of Soft-SAT-D with heuristic MO-csp and Soft-SAT-D with heuristic MO on Max-CSP instances. Time in seconds.

compared with the best performing weighted Partial Max-SAT solvers. We believe that the introduction of auxiliary variables limits the application of the lower bounding techniques incorporated into W-MaxSatz and MiniMaxsat.

	Soft-SAT-S		MiniMaxsat		W-MaxSatz	
$\langle n,d,p_1,p_2\rangle$	mean	median	mean	median	mean	median
$\langle 10,8,45/45,48/64\rangle$	**0.33**	**0.32**	2.05	1.96	79.78	81.62
$\langle 12,6,66/66,27/36\rangle$	**0.48**	**0.47**	3.97	3.89	234	232
$\langle 14,5,91/91,18/25\rangle$	**0.82**	**0.77**	8.83	8.61	785	782
$\langle 16,4,120/120,12/16\rangle$	**0.37**	**0.32**	7.92	7.22	765	750
$\langle 18,3,153/153,6/9\rangle$	**1.00**	**0.93**	1.64	1.50	153	143
$\langle 15,6,60/105,27/36\rangle$	**0.25**	**0.24**	1.36	1.23	1653	1618
$\langle 18,5,80/153,18/25\rangle$	**0.67**	**0.58**	2.63	2.16	>7200	>7200
$\langle 20,5,70/190,18/25\rangle$	**0.54**	**0.44**	1.29	1.21	>7200	>7200
$\langle 14,8,91/91,50/64\rangle$	**44.27**	**43.95**	571	546	>7200	>7200
$\langle 23,4,200/253,12/16\rangle$	**102**	**79.61**	378	335	>7200	>7200

Table 4.5: Comparison of Soft-SAT-S, MiniMaxsat and W-MaxSatz on Max-CSP instances. Time in seconds.

Experiments with graph coloring instances

Another benchmark of our empirical investigation was graph coloring. In this case, when solving the weighted Partial Max-SAT instances, we can either use the encoding provided by the reduction of Soft-SAT to weighted Partial Max-SAT that we have defined or we can use a simpler encoding that does not use additional variables. In that encoding, the hard block is encoded in the same way, and, in the soft blocks, the weight associated with each clause is one, and

Copia gratuita. Personal free copy http://libros.csic.es

	Soft-SAT-S		Soft-SAT-D		Toolbar-CSP		PFC-MPRDAC	
$\langle n,k,c\rangle$	mean	median	mean	median	mean	median	mean	median
$\langle 15,15,8\rangle$	**104**	**10.47**	319	26.73	180	36.77	133	21.29
$\langle 15,15,10\rangle$	**103**	**0.05**	262	0.06	268	0.08	140	0.15
$\langle 16,14,6\rangle$	197	49.00	987	225	**141**	**40.13**	234	78.63
$\langle 16,14,8\rangle$	**165**	**19.38**	392	29.45	267	43.44	208	26.26
$\langle 16,16,6\rangle$	208	130	950	545	**142**	**81.60**	250	181
$\langle 16,16,8\rangle$	**91.87**	**23.33**	225	37.11	199	51.74	147	37.01

Table 4.6: Comparison between Soft-SAT-S with heuristic csp, Soft-SAT-D with heuristic MO-csp, Toolbar-CSP and PFC-MPRDAC on randomly generated graph coloring instances. Time in seconds.

the unit clauses containing the additional variable, as well as the occurrences of that variable in the remaining clauses, are not included. The correctness of that encoding follows from the fact that there is at most one violated clause in each soft block. We used this encoding because leads to better performance profiles.

In the first experiment we considered 6 sets of randomly generated instances, where each set had 100 instances. We solved the instances with Soft-SAT-S with heuristic csp, Soft-SAT-D with heuristic MO-csp, Toolbar-CSP and PFC-MPRDAC.[3] The results obtained are shown in Table 4.6: the first column displays the parameters given to the generator, and the rest of columns display the mean and median time needed to solve an instance of the set with each one of the used solvers.

We repeated the previous experiments but using a representative sample of individual instances from the graph coloring symposium celebrated as a co-located event of CP-2002. The results obtained are shown in Table 4.7: the first column displays the name of the instance, the optimum number of colors to get a valid coloring (k), and the number of colors we used to color the graph (c); the second column displays the number of violated constraints; and the rest of columns display the time needed to solve the instance with each one of the used solvers. We observe in both experiments that Soft-SAT is very competitive with respect to Toolbar-CSP and superior to PFC-MPRDAC.

To conclude the experimentation with the graph coloring benchmark, we compared our Soft-SAT solvers with MiniMaxsat and W-MaxSatz. As we can observe in Table 4.8, the weighted Partial Max-SAT solvers do not improve the results of Soft-SAT-S with these instances. Table 4.9 shows the results for the individual instances of graph coloring; we can observe that Soft-SAT-D is competitive compared with MiniMaxsat and W-MaxSatz.

[3]We do not give results with some weighted Max-SAT solvers because they are not competitive with the solvers used.

© CSIC © del autor o autores / Todos los derechos reservados

$\langle$Instance, $k, c\rangle$	vc	Soft-SAT-S	Soft-SAT-D	Toolbar-CSP	PFC-MPRDAC
$\langle$myciel5.col, 6, 3$\rangle$	16	11.04	46.39	**0.66**	12.11
$\langle$myciel5.col, 6, 4$\rangle$	4	78.50	226.59	**6.28**	96.41
$\langle$myciel5.col, 6, 5$\rangle$	1	3178	31.87	**26.02**	44.34
$\langle$GEOM30a.col, 6, 3$\rangle$	11	9.31	27.22	**0.87**	14.33
$\langle$GEOM30a.col, 6, 4$\rangle$	4	4.48	**2.35**	2.42	22.89
$\langle$GEOM30a.col, 6, 5$\rangle$	1	0.49	**0.15**	0.17	0.18
$\langle$GEOM40.col, 6, 2$\rangle$	22	3.89	20.58	**0.08**	4.42
$\langle$GEOM40.col, 6, 3$\rangle$	7	**10.83**	30.63	25.20	770
$\langle$GEOM40.col, 6, 4$\rangle$	3	95.18	**14.67**	1981	>7200
$\langle$GEOM40.col, 6, 5$\rangle$	1	1.58	**0.51**	1186	1574
$\langle$queen5_5.col, 5, 3$\rangle$	29	57.60	168	**9.22**	27.27
$\langle$queen5_5.col, 5, 4$\rangle$	12	37.50	124	**13.18**	73.67

Table 4.7: Comparison between Soft-SAT-S, Soft-SAT-D, Toolbar-CSP and PFC-MPRDAC on individual graph coloring instances.Time in seconds.

	Soft-SAT-S		Soft-SAT-D		MiniMaxsat		W-MaxSatz	
$\langle n, k, c\rangle$	mean	median	mean	median	mean	median	mean	median
$\langle 15, 15, 8\rangle$	**104**	**10.47**	319	26.73	290	36.14	2615	547
$\langle 15, 15, 10\rangle$	**103**	**0.05**	262	0.06	417	0.08	4428	0.07
$\langle 16, 14, 6\rangle$	**197**	**49.00**	987	225	860	147	3596	1042
$\langle 16, 14, 8\rangle$	**165**	**19.38**	392	29.45	390	40.88	5077	1016
$\langle 16, 16, 6\rangle$	**208**	**130**	950	545	833	362	3754	2284
$\langle 16, 16, 8\rangle$	**91.87**	**23.33**	225	37.11	275	55.35	3664	1877

Table 4.8: Comparison between Soft-SAT-S with heuristic csp, Soft-SAT-D with heuristic MO-csp, MiniMaxsat and W-MaxSatz on randomly generated graph coloring instances. Time in seconds.

Copia gratuita. Personal free copy http://libros.csic.es

$\langle$Instance, $k, c\rangle$	vc	Soft-SAT-S	Soft-SAT-D	MiniMaxsat	W-MaxSatz
$\langle$myciel5.col, 6, 3$\rangle$	16	11.04	46.39	2.24	**2.16**
$\langle$myciel5.col, 6, 4$\rangle$	4	78.50	226.59	**13.79**	4143.96
$\langle$myciel5.col, 6, 5$\rangle$	1	3178	**31.87**	>7200	439.98
$\langle$GEOM30a.col, 6, 3$\rangle$	11	9.31	27.22	2.03	**1.53**
$\langle$GEOM30a.col, 6, 4$\rangle$	4	4.48	2.35	**0.84**	1011.28
$\langle$GEOM30a.col, 6, 5$\rangle$	1	0.49	**0.15**	>7200	4128.63
$\langle$GEOM40.col, 6, 2$\rangle$	22	3.89	20.58	0.17	**0.15**
$\langle$GEOM40.col, 6, 3$\rangle$	7	10.83	30.63	**3.73**	37.27
$\langle$GEOM40.col, 6, 4$\rangle$	3	95.18	14.67	**14.14**	>7200
$\langle$GEOM40.col, 6, 5$\rangle$	1	1.58	**0.51**	>7200	>7200
$\langle$queen5_5.col, 5, 3$\rangle$	29	57.60	168	34.36	**24.51**
$\langle$queen5_5.col, 5, 4$\rangle$	12	37.50	124	**21.80**	292.64

Table 4.9: Comparison between Soft-SAT-S, Soft-SAT-D, MiniMaxsat and W-MaxSatz on individual graph coloring instances.Time in seconds.

Experiments with pigeon hole instances

We solved pigeon hole instances with a number of holes ranging from 7 to 12 in order to study the scaling behaviour on Soft-SAT solvers (Soft-SAT-S with heuristic csp and Soft-SAT-D with heuristic MO-csp), weighted Max-SAT solvers (Toolbar and WMax-SAT) and weighted Partial Max-SAT solvers (MiniMaxSAT and W-MaxSatz). The results obtained are shown in Table 4.10. We observe that the solver with best scaling behaviour is Soft-SAT-D and then Soft-SAT-S. The weighted Max-SAT solvers scale worse than the Soft-SAT solvers. Also, we can see that the newest solving techniques incorporated into weighted Partial Max-SAT solvers do not help improve the results in this benchmarks.

N	Soft-SAT-S	Soft-SAT-D	Toolbar	WMax-SAT	MiniMaxsat	W-MaxSatz
7	**0.07**	0.10	0.43	0.08	0.25	0.08
8	**0.19**	0.28	4.05	0.54	0.75	0.83
9	**1.13**	1.55	43	5.32	5.53	8.77
10	**11**	12	521	75	60	102
11	133	**103**	6741	705	765	1317
12	1784	**990**	>7200	>7200	>7200	>7200

Table 4.10: Comparison between Soft-SAT-S with heuristic csp, Soft-SAT-D with heuristic MO-csp, Toolbar and WMax-SAT on pigeon hole instances. Time in seconds.

© CSIC © del autor o autores / Todos los derechos reservados

Experiments with QCP instances

QCP instances were the last benchmark considered. We solved sets of 100 unsatisfiable instances ranging from quasigroups of order 6 to quasigroups of order 10, and with 40% of preassigned entries. The results obtained are shown in Table 4.11. We observe that the best performing solver is Soft-SAT-D and then Soft-SAT-S when the order increases. The weighted Max-SAT solvers scale worse than the Soft-SAT solvers, and this also holds for the best performing weighted Partial Max-SAT solvers of the 2007 Max-SAT Evaluation.

order	holes	Soft-SAT-D	Soft-SAT-S	Toolbar	WMax-SAT	MiniMaxsat	W-MaxSatz
6	21	0.35	**0.33**	34	1.26	0.46	66
7	29	0.97	1.11	17474	90	**0.75**	>20000
8	38	5.12	20	>20000	4806	**4.72**	>20000
9	48	**137**	2963	>20000	>20000	162	>20000
10	60	**8050**	>20000	>20000	>20000	>20000	>20000

Table 4.11: Comparison between Soft-SAT-D with heuristic MO-csp, Soft-SAT-S with heuristic csp, Toolbar and WMax-SAT on QCP instances. Time in seconds.

4.4 Summary

We have presented a new generic problem solving approach for over-constrained problems based on a formalism that deals with hard and soft blocks of clauses. The distinction between hard and soft blocks allows us to model problems in a more natural and compact way, and to design Max-SAT-like solvers that traverse efficiently the search space of all possible truth assignments. In particular, we have provided experimental evidence that exploiting the fact of knowing whether variables and clauses appear in the hard block or in a soft blocks is relevant for devising good performing variable selection heuristics and inference methods:

- Variable selection heuristics: we can define heuristics like MOH that take into account whether a variable occurrence belongs to the hard block or to a soft block.
- Inference methods: we get an extra level of propagation by applying the unit clause rule to unit clauses that appear in the hard block. Moreover, the inference applied in SAT can be locally applied inside each soft block.

We have also exploited the structure which is hidden in the encoding to define a lower bound of better quality and an n-ary branching, and defined extremely efficient lazy data structures for the Soft-SAT-S solver.

Moreover, we have shown that our approach exhibits a better performance profiles than reducing over-constrained problems to weighted Partial Max-SAT for some classes of problems. We believe that the introduction of auxiliary

Copia gratuita. Personal free copy http://libros.csic.es

variables limits the application of the lower bounding techniques incorporated into modern weighted Partial Max-SAT solvers.

The empirical investigation provides evidence that our approach is very competitive compared with solving over-constrained problems by reducing them to Max-CSP problems. Taking into account the amount of efforts devoted in the Constraint Programming community on investigating methods for solving over-constrained problems, we believe that Soft-SAT is a suitable alternative to solve over-constrained problems.

We would also like to comment the good results we obtained with Soft-SAT-S on some instances of the empirical investigation. The extremely efficient data structures that we have implemented are a key factor of its success. We believe that the incorporation of more sophisticated variable selection heuristics into Soft-SAT-D will provide us with faster Soft-SAT-D solvers.

It is worth mentioning that, when we started our research on Soft-SAT, we did not found in the SAT literature any approach of solving problems with hard and soft constraints using exact Max-SAT algorithms. All the papers we found refered to local search algorithms, and did not incorporate the notion of block of clauses. Currently, Soft-SAT solvers can be improved by adapting some techniques implemented in the last Partial Max-SAT solvers like hard and soft learning, lower bound computation using unit propagation, transforming the formula to a simpler one using inference rules, or incorporating failed literal detection. For example, the computation of the lower bound using unit propagation could be implemented taking into account that, in Soft-SAT, we have to detect disjoint inconsistent subsets of blocks instead of detecting disjoint inconsistent subsets of clauses.

© CSIC © del autor o autores / Todos los derechos reservados

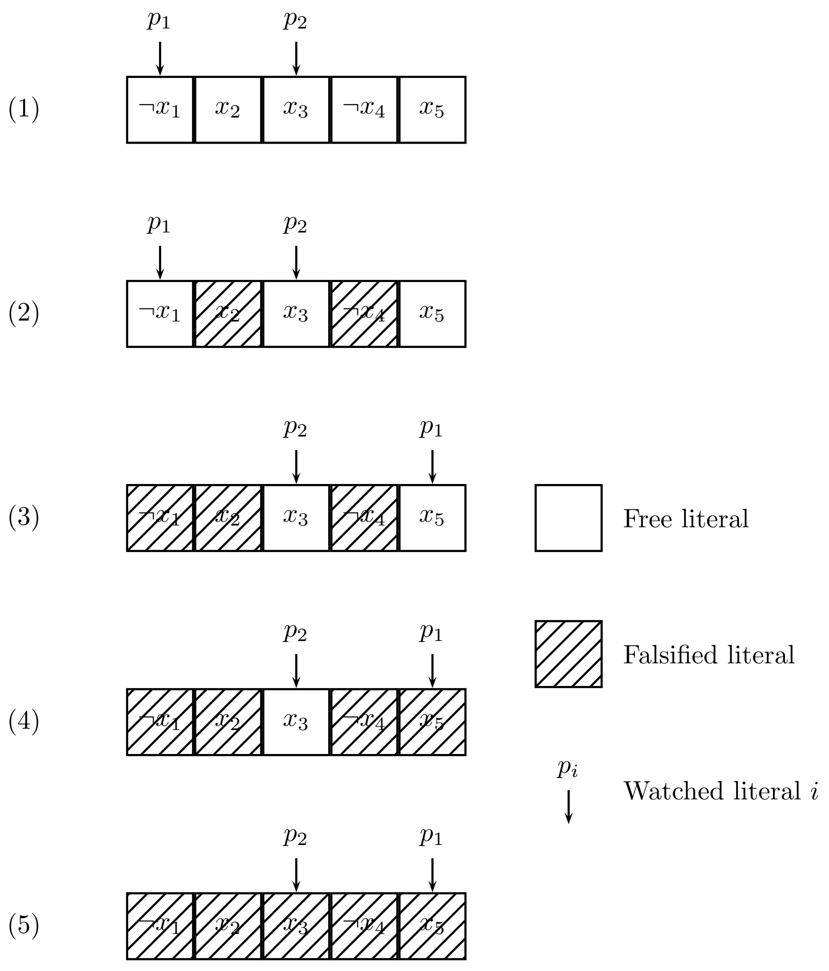
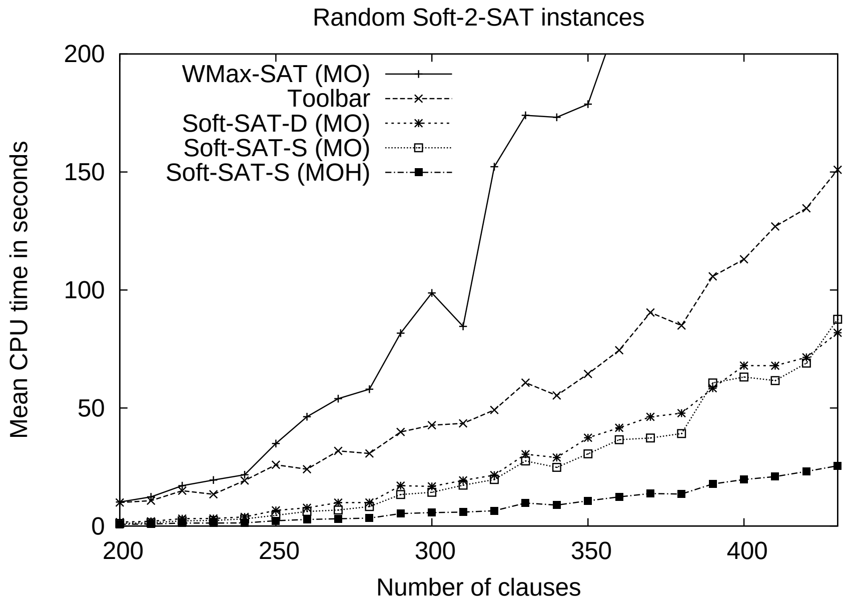


Figure 4.4: Data structures behaviour for dynamic variable ordering.

Copia gratuita. Personal free copy http://libros.csic.es



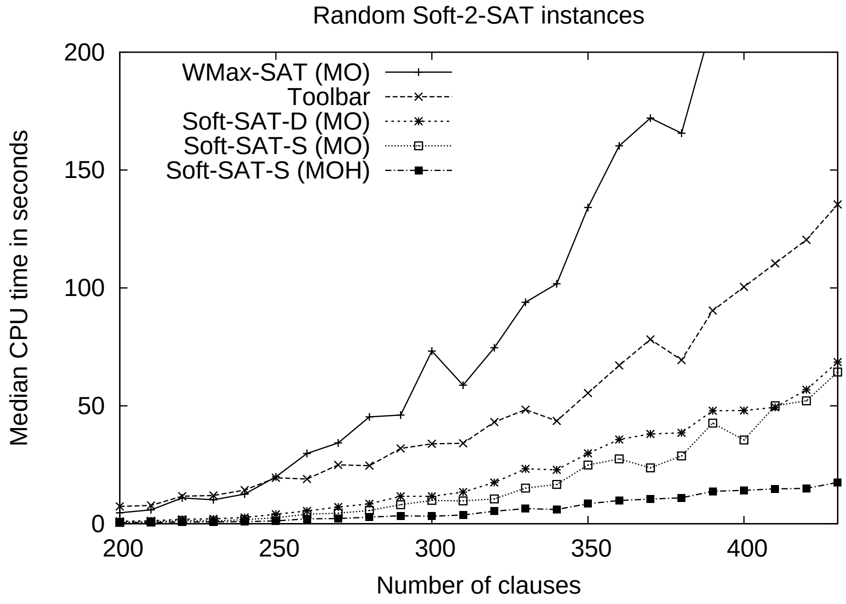
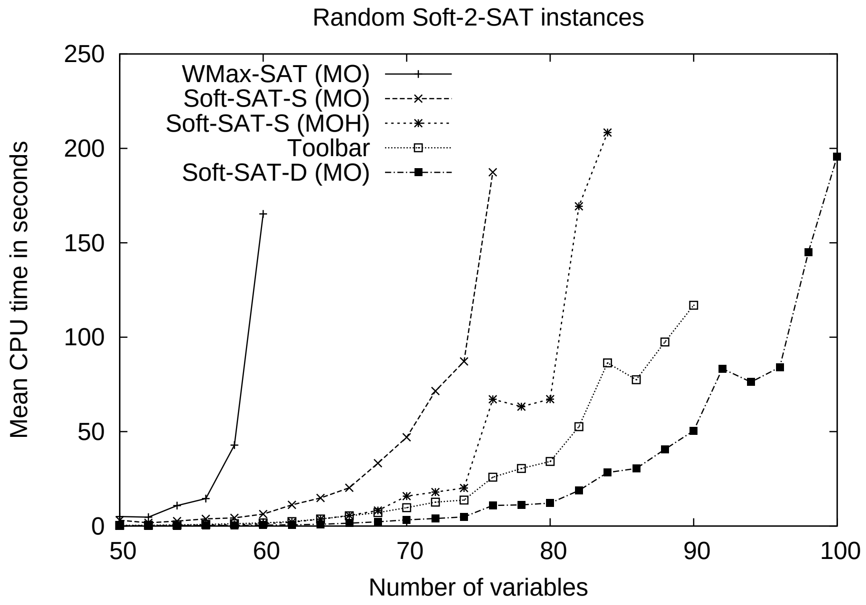


Figure 4.5: Random Soft-2-SAT instances with 50 variables, with a number of clauses ranging from 200 to 430, where 20 clauses are in the hard block and the rest of clauses are randomly distributed among 100 soft blocks. Mean time (upper plot) and median time (lower plot) in seconds.

© CSIC © del autor o autores / Todos los derechos reservados



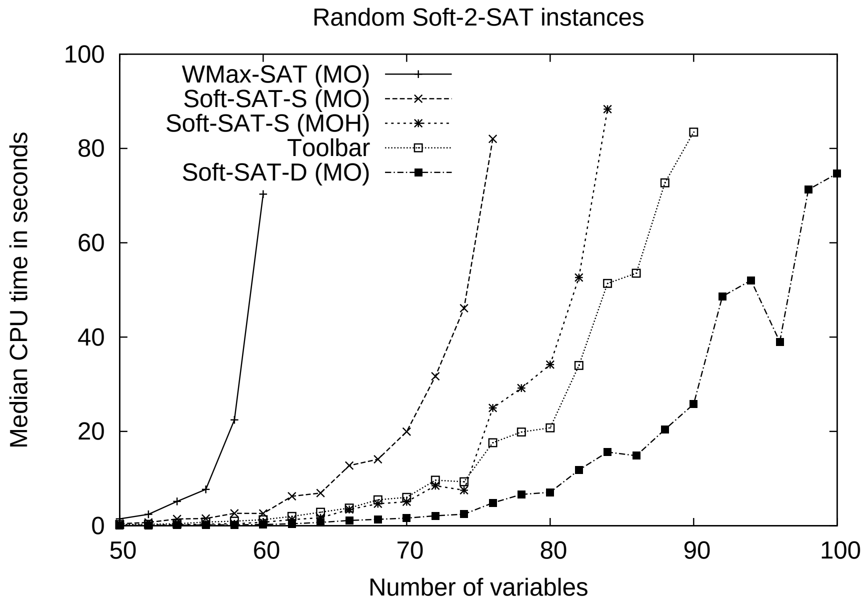
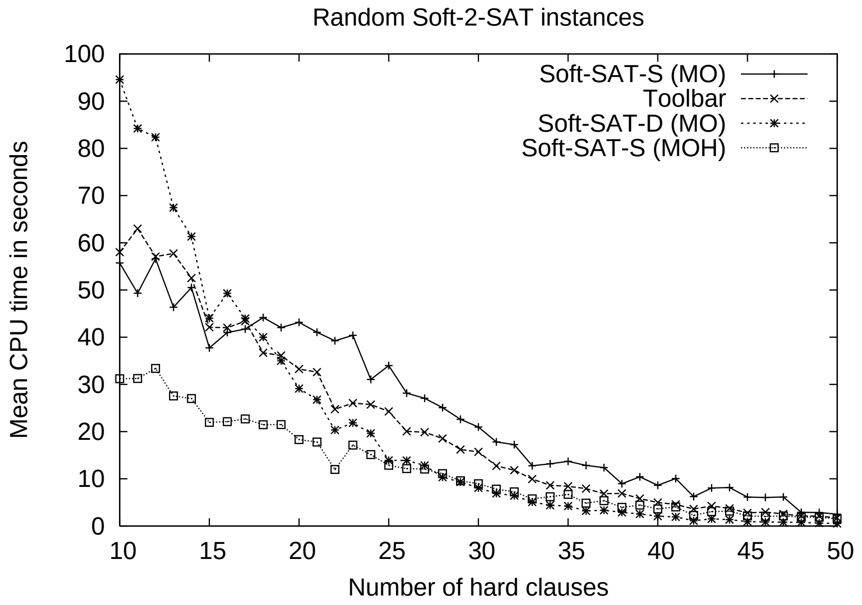


Figure 4.6: Random Soft-2-SAT instances with a number of variables ranging from 50 to 100 and with 300 clauses, where 50 clauses are in the hard block and the rest of clauses are randomly distributed among 50 soft blocks. Mean time (upper plot) and median time (lower plot) in seconds.

Copia gratuita. Personal free copy http://libros.csic.es



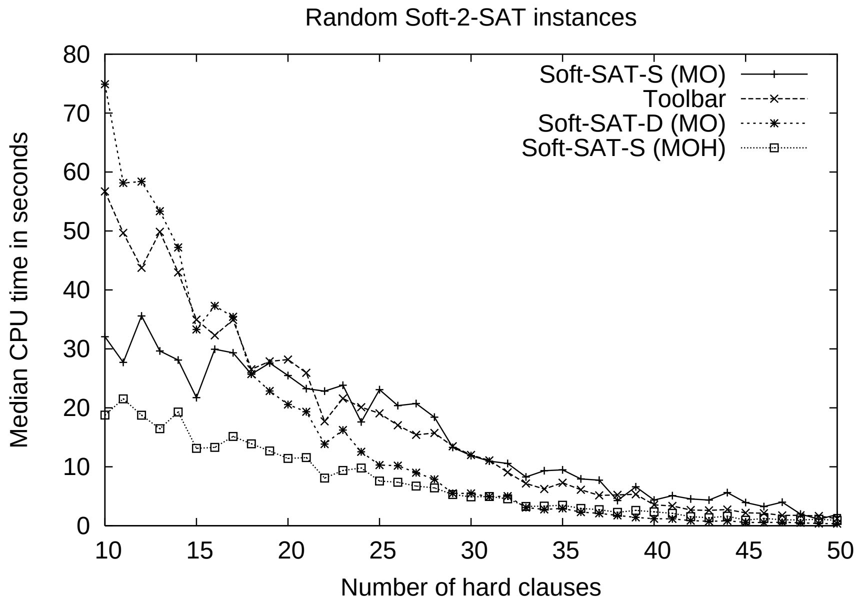
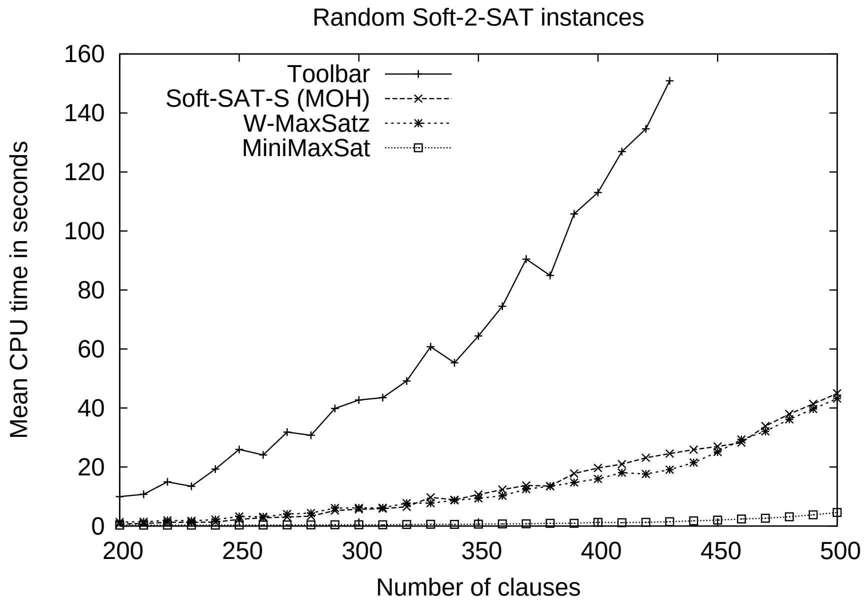


Figure 4.7: Random Soft-2-SAT instances with 60 variables and 300 clauses, where the number of clauses in the hard block ranges from 10 to 50 and the rest of clauses are randomly distributed among 50 soft blocks. Mean time (upper plot) and median time (lower plot) in seconds.

© CSIC © del autor o autores / Todos los derechos reservados



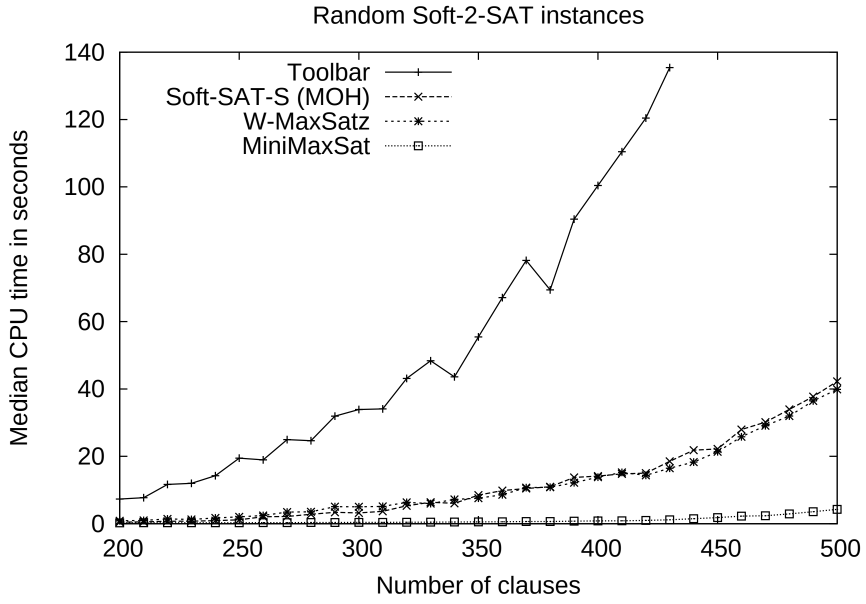
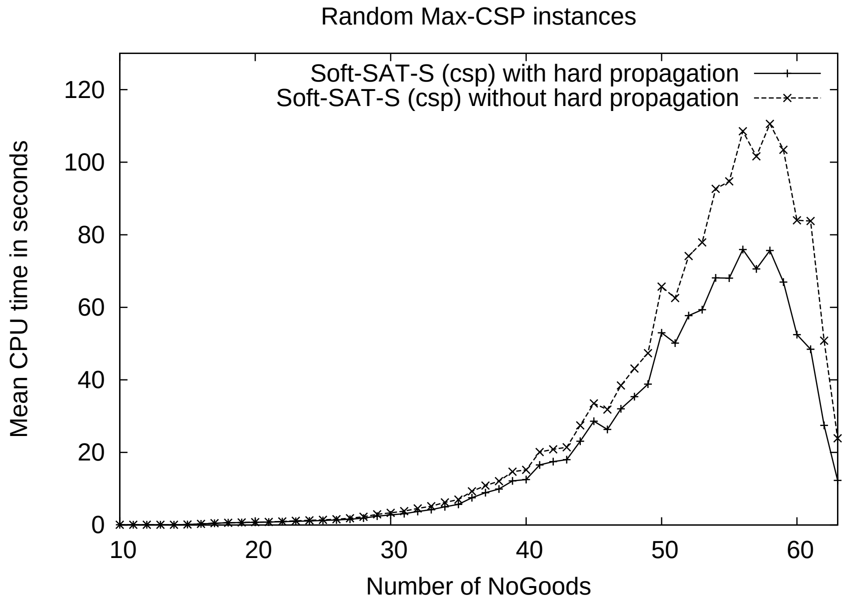


Figure 4.8: Random Soft-2-SAT instances with 50 variables, with a number of clauses ranging from 200 to 430, where 20 clauses are in the hard block and the rest of clauses are randomly distributed among 100 soft blocks. Mean time (upper plot) and median time (lower plot) in seconds.

Copia gratuita. Personal free copy http://libros.csic.es



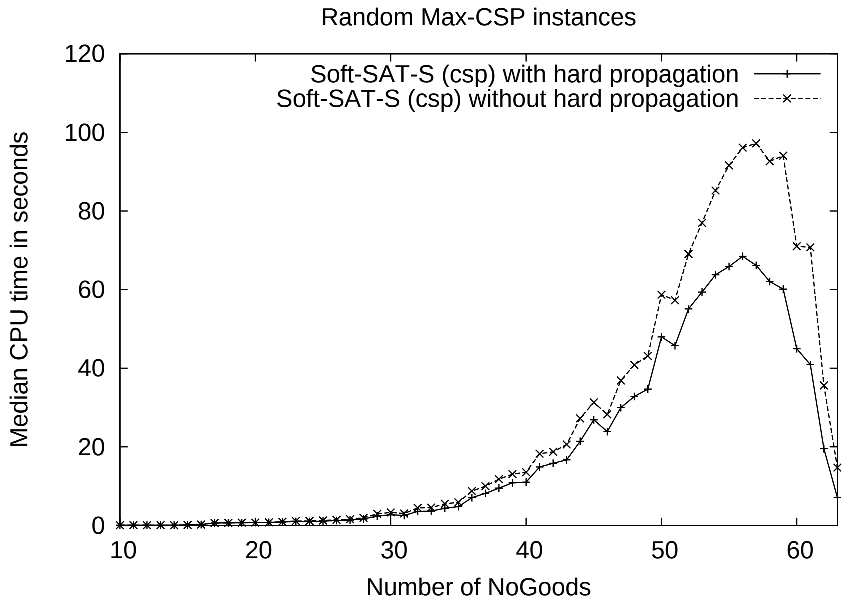


Figure 4.9: Comparison of Soft-SAT-S with a version of Soft-SAT-S in which the unit clause rule is not applied to unit clauses that appear in the hard block. Mean time (upper plot) and median time (lower plot) in seconds.

© CSIC © del autor o autores / Todos los derechos reservados

Chapter 5

The Partial Max-SAT formalism

In this chapter we focus on Partial Max-SAT, which is a problem between SAT and Max-SAT. Partial Max-SAT is well-suited for representing and solving over-constrained problems, and has become a standard in recent years. As a proof of the growing interest in Partial Max-SAT we would like to emphasize the two new categories added to the 2007 Max-SAT Evaluation: weighted Partial Max-SAT and unweighted Partial Max-SAT, as well as the number of Partial Max-SAT solvers that have been recently developed (ChaffBS, ChaffLS, Clone, LB-SAT, MiniMaxsat, SAT4Jmaxsat, SR(w), Toolbar,…).

The chapter is structured as follows. In Section 5.1 we present an overview of the Partial Max-SAT problem. In Section 5.2 we define novel techniques for Partial Max-SAT solving, and introduce the solving techniques that incorporate the modern Partial Max-SAT solvers. In Section 5.3 we present some efficient and original preprocessing techniques for Partial Max-SAT. In Section 5.4 we describe the two Partial Max-SAT solvers that we have designed and implemented: PMS and W-MaxSatz. Finally, in Section 5.5 we report an experimental investigation that we conducted in order to assess the performance of our solvers and preprocessing techniques.

5.1 The Partial Max-SAT problem

Partial Max-SAT was first defined in 1996 by Miyazaki et al. [MIK96] in the context of optimization of database queries. A Partial Max-SAT instance is a CNF formula in which some clauses are *relaxable* or *soft* and the rest are *non-relaxable* or *hard*. Solving a Partial Max-SAT instance amounts to find an assignment that satisfies all the hard clauses and the maximum number of soft clauses.

Let us illustrate the expressive power of Partial Max-SAT by showing how to encode a Max-Clique instance into Partial Max-SAT.

Copia gratuita. Personal free copy http://libros.csic.es

Example 5.1 *Given a graph, the Max-Clique problem consists in finding a clique*[1] *of maximum size. If we consider the graph with set of vertices* $V = \{v_1, v_2, v_3, v_4\}$ *and set of edges* $E = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_2, v_4)\}$, *the Max-Clique problem for this graph can be encoded as a Partial Max-SAT instance as follows:*

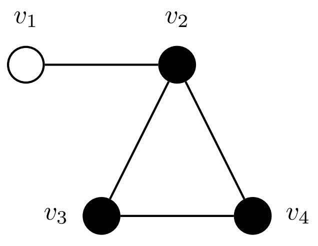


Figure 5.1: Max-Clique example with four vertices and four edges.

1. *The set of propositional variables is* $\{x_1, x_2, x_3, x_4\}$. *The variable* x_i *is true if vertex* v_i *belongs to the clique.*
2. *The following hard clauses encode that any two vertices not connected by an edge cannot belong to the same clique:*

$$[\neg x_1 \vee \neg x_3], [\neg x_1 \vee \neg x_4];$$

3. *The following soft clauses encode the function to maximize; i.e., the number of variables assigned to true which belong to the maximum clique. Since all the variables set to true belong to the clique, the aim is to maximize the number of satisfied positive literals. Therefore, the soft clauses are:*

$$x_1, x_2, x_3, x_4.$$

The optimal solutions for this example is $x_1 = false$, *and* $x_2 = x_3 = x_4 = true$.

Note that we write hard clauses between square brackets in order to distinguish hard clauses from soft clauses.

5.2 Partial Max-SAT algorithms

We first define a basic branch and bound Partial Max-SAT solver and then explain the main features we added to the basic solver in order to obtain the

[1]A clique in a graph is a set of pairwise adjacent vertices, or in other words, an induced subgraph which is a complete graph.

© CSIC © del autor o autores / Todos los derechos reservados

different versions of our Partial Max-SAT solvers. In Section 5.4, we will enumerate the techniques described in this section which are incorporated into our Partial Max-SAT solvers (PMS and W-MaxSatz).

5.2.1 A basic Partial Max-SAT algorithm

The space of all possible assignments for a Partial Max-SAT instance ϕ can be represented as a search tree, where internal nodes represent partial assignments and leaf nodes represent complete assignments. A branch and bound (BnB) algorithm explores that search tree in a depth-first manner. At each node, the algorithm backtracks if the current partial assignment violates some hard clause, and applies the unit clause rule to the literals that occur in unit hard clauses; i.e., given a literal $\neg x$ (x), it deletes all the clauses containing the literal $\neg x$ (x) and removes all the occurrences of the literal x ($\neg x$). If the current partial assignment does not violate any hard clause, the algorithm compares the number of soft clauses falsified by the best complete assignment found so far, called upper bound (UB), with the number of soft clauses falsified 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 x. The instantiation of x leads to the creation of two branches from the current branch: the left branch corresponds to instantiating x to false, and the right branch corresponds to instantiating x 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 unit clause rule using the literal $\neg x$ (x). The value that UB takes after exploring the entire search tree is the minimum number of soft clauses that are falsified by a complete assignment that satisfies all the hard clauses.

As in Soft-SAT solvers, Partial Max-SAT solvers enforce unit propagation on unit hard clauses, and prune a branch of the search tree as soon as a hard clause is violated. Algorithm 5.1 shows the pseudocode of a basic Partial Max-SAT solver.

As we can see, the basic Partial Max-SAT algorithm is quite similar to the basic Soft-SAT algorithm of Section 4.2.1. The main difference between them is that in Partial Max-SAT we deal with soft clauses instead of soft blocks of clauses. Most of the improvements of the following sections are not applied in Soft-SAT because the notion of block does not allow their application, or the adaptation to blocks reduces drastically the performance of the technique and becomes inefficient.

In the following sections we describe the solving techniques that have shown to be effective for Partial Max-SAT, and present new techniques and improvements that we have incorporated into our solvers. Our contributions can be summarized as follows:

- **Variable selection heuristic:** We have adapted to Partial Max-SAT

Copia gratuita. Personal free copy http://libros.csic.es

Algorithm 5.1: `Partial-Max-SAT-Basic`(ϕ, UB) : Basic Partial Max-SAT solver

```
Output: The minimum number of soft clauses of Partial Max-SAT
        instance $\phi$ that are falsified by an assignment
Function Partial-Max-SAT-Basic ($\phi$ : Partial Max-SAT instance, $UB$ : upper bound) : Natural
    if hard clause is unsatisfied then
        return $\infty$
    $\phi \leftarrow$ HardUnitPropagation($\phi$)
    if $UB =$ LowerBound($\phi$)$+1$ then
        $\phi \leftarrow$ SoftUnitPropagation($\phi$)
    if hard clauses are satisfied and $UB >$ LowerBound($\phi$) then
        if $\phi = \emptyset$ or $\phi$ only contains empty clauses then
            return EmptyClauses($\phi$)
        $x \leftarrow$ SelectVariable($\phi$)
        $UB \leftarrow$ Min($UB$, Partial-Max-SAT-Basic($\phi_{\neg x}$, $UB$))
        return Min($UB$, Partial-Max-SAT-Basic($\phi_x$, $UB$))
    else
        return $\infty$
```

two of the best performing variable selection heuristics for Max-SAT: CS heuristic and the MaxSatz heuristic.

- **Bounds computation:** We have implemented a GSAT-like algorithm for Partial Max-SAT in order to obtain a good initial upper bound. Regarding lower bounds, we have defined new sound inference rules that have been incorporated into our solvers with the aim of deriving as early as possible empty clauses, and have adapted UP (c.f. Section 3.2.1) to Partial Max-SAT for both computing underestiamtions and guiding the application of the inference rules.
- **Hard learning:** We have incorporated into our Partial Max-SAT solvers a learning module that performs a conflict analysis, and learns a hard clause, when a hard clause is violated by the current assignment of a branch and bound solver.
- **Soft learning:** We have incorporated a module that analyzes the conflicts detected when at least one of the conflict clauses is soft.

5.2.2 Variable selection heuristic

We have adapted two of the best dynamic variable selection heuristics for Max-SAT to Partial Max-SAT to take advantage of having hard and soft clauses.

– **CS heuristic:** Clause Size (CS) heuristic assigns a score to each variable. Such a score is computed taking into account the lenght of the clause in

© CSIC © del autor o autores / Todos los derechos reservados

which the variable appears and whether the clause is hard or soft. For soft clauses, it gives a score of 16 to variables appearing in binary clauses, a score of 4 to variables appearing in ternary clauses and a score of 1 to variables appearing in the remaining clauses. For hard clauses, it adds the same score as for soft clauses multiplied by the number of soft clauses. CS selects a variable with maximum score.

– **MaxSatz heuristic:** It is a variable/value selection heuristic introduced in [LMP06]. We have extended the heuristic of MaxSatz to solve Partial Max-SAT problems. It associates a weight equal to one to soft clauses and a weight w to hard clauses, where w is the total number of soft clauses. Let $B(\ell)$ and $C(\ell)$ be the sum of weights associated to binary clauses containing the literal ℓ, and the sum of weights associated with the remaining clauses containing the literal ℓ, respectively. The heuristics are defined as follows:

  – Variable selection heuristic: It selects the variable x such that $(4B(\neg x)+C(\neg x)) * (4B(x)+C(x))$ is the largest.

  – Value selection heuristic: Let x be the selected branching variable. If $4B(\neg x)+C(\neg x) < 4B(x)+C(x)$, set x to true. Otherwise, set x to false.

5.2.3 Bounds computation

Upper bound

The initial upper bound is computed with a local search solver. We have tried several available implementations of local search algorithms, and we have also implemented a Partial Max-SAT version of GSAT [SLM92].

The local search solvers that we have incorporated into our Partial Max-SAT solvers are:

– **P-GSAT:** It is a variant of GSAT that we have adapted to solve Partial Max-SAT. The search begins with a randomly generated complete truth assignment and, at each step, the value of one variable is flipped taking into account its score. The score of a variable is the sum of weights that we associate with unsatisfied clauses; we associate a weight one to an unsatisfied soft clause and a weight equal to the total number of clauses to an unsatisfied hard clause. Local minima are avoided by occasionally performing a random walk.

– **UBCSAT:** UBCSAT [TH05] is a repository of local search algorithms for SAT and Max-SAT. It is provided with some tools designed to solve weighted Max-SAT problems. Among all the algorithms available in the repository, we use *IROTS (Iterated Robust Tabu Search)* [SHS03] for the computation of the upper bound. This is the algorithm with better general performance in our tests with Partial Max-SAT instances. To solve Partial

Copia gratuita. Personal free copy http://libros.csic.es

Max-SAT instances with this solver, we use the tools for weighted Max-SAT instances associating a weight one to soft clauses and a weight equal to the total number of clauses to hard clauses.

Lower bound

We adapted lower bound UP [LMP05, LMP06] to Partial Max-SAT. In lower bound UP for Max-SAT, the lower bound is the current number of unsatisfied clauses plus an underestimation of the minimum number of clauses that will become unsatisfied if the current partial assignment is extended to a complete assignment. Such an underestimation is the number of disjoint unsatisfiable subsets that can be detected using unit propagation.

In lower bound UP for Partial Max-SAT, the underestimation is the number of unsatisfiable subsets that can be derived by applying unit propagation in such a way that soft clauses appear only in one subset. In UP for Max-SAT, the clauses in unsatisfiable subsets can appear just in one subset. In Partial Max-SAT, hard clauses can appear in more than one subset. This is a crucial point for obtaining a better performance profile than in Max-SAT for some instances.

There are several ways to implement UP:

– Single queue (UP_{SQ}): Older unit clauses are preferred to more recent unit clauses.

– Stack (UP_{St}): It stores the unit clauses in a stack. The last inserted unit clause is used first.

– Double queue (UP_{DQ}): It maintains two queues: Q_1 and Q_2. When UP_{DQ} starts to search for an inconsistent subformula, Q_1 contains all the unit clauses of the formula under consideration (more recently derived unit clauses are at the end of Q_1), and Q_2 is empty. The unit clauses derived during the application of unit propagation are stored in Q_2, and unit propagation does not use any unit clause from Q_1 until Q_2 is empty.

Generally speaking, UP_{SQ} creates the implication graph in a breadth-first manner, UP_{St} in a depth-first manner, and UP_{DQ} in a kind of locality and breadth-first manner.

We can incorporate into lower bound UP an additional level of forward look-ahead based on the detection of failed literals [LMP06]. Let ϕ be a Partial MAX-SAT instance, and let ϕ' be the formula resulting from ϕ after replacing every inconsistent subformula detected by UP with an empty clause. Obviously, unit propagation in ϕ' cannot derive any additional empty clause. However, if unit propagation is applied to $\phi' \cup \{x\}$ and $\phi' \cup \{\neg x\}$, for any variable x occurring in ϕ', and produces an empty clause in each formula (i.e., x and $\neg x$ are *failed literals* in ϕ'), then $(\varphi_1 \cup \varphi_2) \setminus \{x, \neg x\}$ is an inconsistent subformula of ϕ', where φ_1 is the inconsistent subformula detected by UP in $\phi' \cup \{x\}$, and φ_2 is the inconsistent subformula detected by UP in $\phi' \cup \{\neg x\}$.

© CSIC © del autor o autores / Todos los derechos reservados

As introducing an additional level of look-ahead is time consuming, only a subset of the variables occurring in the formula are used to detect failed literals. The propositional variables used to detect failed literals do not have to appear in unit clauses, and they must have at least two positive occurrences and two negatives occurrences in binary clauses. For further details about the implementation of UP enhanced with failed literals see [LMP06].

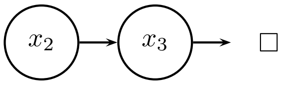
5.2.4 Inference rules

The inference rules that one can apply in Max-SAT have to transform the current instance ϕ into another instance ϕ' in such a way that ϕ and ϕ' have the same number of unsatisfied clauses for every possible assignment; in other words, the inference rules have to be *sound.*

To transform ϕ into ϕ', we replace a set of clauses S with a set of clauses S' in such a way that the number of unsatisfied clauses in S and S' is the same for every assignment.

The applicability of most of the inference rules described in this chapter can be decided when we apply lower bound UP. Once we reach a conflict using UP, we can follow the implication graph and check if it matches with the premises of an inference rule. If so, we apply the inferencer rule. This helps to speed up considerably the run time of the solver.

Example 5.2 *Given the following formula* $\phi = \{x_1 \vee x_2, x_2, \neg x_2 \vee x_3, \neg x_3, x_4 \vee \neg x_5\}$*, if we apply lower bound* UP *to* ϕ*, we get the following implication graph:*



Analyzing the implication graph, we derive the unsatisfiable subset $\{x_2, \neg x_2 \vee x_3, \neg x_3\}$*. This subset matches with the premises of the star rule, and we can replace this subset by an empty clause* $\square$ *and a binary clause* $x_2 \vee \neg x_3$*. After the application of the inferencer rule, the resulting formula is:*

$$\phi = \{\square, x_1 \vee x_2, x_2 \vee \neg x_3, x_4 \vee \neg x_5\}$$

To simplify the description of the inference rules, we describe the inference rules only for Max-SAT. In Max-SAT, the set of clauses in the premises of the rule is replaced by the set of clauses in the conclusions of the rule. For Partial Max-SAT, if we have hard clauses in the premises, they will remain after adding the conclusions of the rule.

Example 5.3 *Given the formula* $\phi = \{x_1 \vee x_2, [x_2], \neg x_2 \vee x_3, \neg x_3, x_4 \vee \neg x_5\}$ *of Example 5.2 with one hard clause, if we apply lower bound* UP *to* ϕ*, we get the unsatisfiable subset* $\{[x_2], \neg x_2 \vee x_3, \neg x_3\}$*. This subset can be replaced by* $\{\square, x_2 \vee \neg x_3\} \cup \{[x_2]\}$*. Note that the hard clause* $[x_2]$ *can be used again to apply lower bound* UP*.*

Copia gratuita. Personal free copy http://libros.csic.es

The existing inference rules for Max-SAT that we have implemented for Partial Max-SAT are the following:

Rule 5.1 (ACC) *[BR99] If* $\phi_1 = \{l_1 \vee l_2 \vee \cdots \vee l_k,\ \bar{l}_1 \vee l_2 \vee \cdots \vee l_k\} \cup \phi'$ *and* $\phi_2 = \{l_2 \vee \cdots \vee l_k\} \cup \phi'$*, then* ϕ_1 *and* ϕ_2 *are equivalent.*

We pay special attention to the case $k = 2$, where the resolvent is a unit clause, and to the case $k = 1$, where the resolvent is the empty clause. The case $k = 1$ is described in the following rule.

Rule 5.2 (CUC) *[NR00] If* $\phi_1 = \{l,\ \bar{l}\} \cup \phi'$ *and* $\phi_2 = \{\Box\} \cup \phi'$*, then* ϕ_1 *and* ϕ_2 *are equivalent.*

Rule 5.2 is used to replace two complementary unit clauses with an empty clause. The new empty clause contributes to the lower bounds of the search space below the current node by incrementing the number of unsatisfied clauses, but not by incrementing the underestimation. Therefore, this contradiction has not to be detected again. In practice, that simple rule gives rise to considerable gains.

Rule 5.3 *If* $\phi_1 = \{l_1,\ \bar{l}_1 \vee \bar{l}_2,\ l_2\} \cup \phi'$ *and* $\phi_2 = \{\Box,\ l_1 \vee l_2\} \cup \phi'$*, then* ϕ_1 *and* ϕ_2 *are equivalent.*

Rule 5.3 replaces three clauses with an empty clause, and adds a new binary clause to keep the equivalence between ϕ_1 and ϕ_2.

Rule 5.4 *If* $\phi_1 = \{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3,\ \cdots,\ \bar{l}_k \vee l_{k+1},\ \bar{l}_{k+1}\} \cup \phi'$*,* $\phi_2 = \{\Box,\ l_1 \vee \bar{l}_2,\ l_2 \vee \bar{l}_3,\ \cdots,\ l_k \vee \bar{l}_{k+1}\} \cup \phi'$*, then* ϕ_1 *and* ϕ_2 *are equivalent.*

$$\begin{tikzcd} l_1 \arrow[r] & l_2 \arrow[r] & l_3 \arrow[r] & \cdots \arrow[r] & l_k \arrow[r] & l_{k+1} \arrow[r] & \Box \end{tikzcd}$$

Figure 5.2: Rule 5.4 implication graph.

Rule 5.4 generalizes Rule 5.2 and Rule 5.3. It captures linear unit resolution refutations in which clauses and resolvents are used exactly once. The rule simply eliminates the unit and binary clauses used in the refutation, and adds an empty clause and k new binary clauses that are obtained by negating the literals of the eliminated binary clauses. So, all the operations involved can be performed efficiently. Figure 5.2 shows the implication graph for Rule 5.4.

Rule 5.5 *If* $\phi_1 = \{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_1 \vee l_3,\ \bar{l}_2 \vee \bar{l}_3\} \cup \phi'$ *and* $\phi_2 = \{\Box,\ l_1 \vee \bar{l}_2 \vee \bar{l}_3,\ \bar{l}_1 \vee l_2 \vee l_3\} \cup \phi'$*, then* ϕ_1 *and* ϕ_2 *are equivalent.*

Rule 5.5 captures unit resolution refutations in which there is a linear derivation but the unit clause is used twice in the derivation of the empty clause.

© CSIC © del autor o autores / Todos los derechos reservados

Rule 5.6 *If* $\phi_1=\{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3,\ \cdots,\ \bar{l}_k \vee l_{k+1},\ \bar{l}_{k+1} \vee l_{k+2}, \bar{l}_{k+1} \vee l_{k+3},\ \bar{l}_{k+2} \vee \bar{l}_{k+3}\} \cup \phi'$ *and* $\phi_2=\{\Box,\ l_1 \vee \bar{l}_2,\ l_2 \vee \bar{l}_3,\ \cdots,\ l_k \vee \bar{l}_{k+1},\ l_{k+1} \vee \bar{l}_{k+2} \vee \bar{l}_{k+3},\ \bar{l}_{k+1} \vee l_{k+2} \vee l_{k+3}\} \cup \phi'$*, then* ϕ_1 *and* ϕ_2 *are equivalent.*

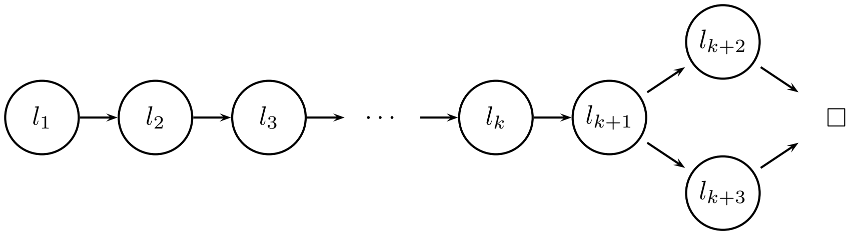


Figure 5.3: Rule 5.6 implication graph.

Rule 5.6 is a combination of a linear derivation and Rule 5.5. Figure 5.3 shows the implication graph for Rule 5.6.

We now define four original inference rules and prove their soundness. These rules have been incorporated into a Partial Max-SAT solver and tested empirically. The experiments provide evidence that they produce substantial speedups on some classes of instances.

Rule 5.7 *If* $\phi_1=\{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3,\ \cdots,\ \bar{l}_k \vee l_{k+1},\ \bar{l}_{k+1} \vee l_{k+2}, \bar{l}_{k+2} \vee l_{k+3}, \bar{l}_{k+1} \vee l_{k+4},\ \bar{l}_{k+3} \vee \bar{l}_{k+4}\} \cup \phi'$ *and* $\phi_2=\{\Box,\ l_1 \vee \bar{l}_2,\ l_2 \vee \bar{l}_3,\ \cdots,\ l_k \vee \bar{l}_{k+1},\ l_{k+1} \vee \bar{l}_{k+2} \vee l_{k+3},\ \bar{l}_{k+1} \vee l_{k+2} \vee \bar{l}_{k+3},\ l_{k+1} \vee \bar{l}_{k+3} \vee \bar{l}_{k+4},\ \bar{l}_{k+1} \vee l_{k+3} \vee l_{k+4}\} \cup \phi'$*, then* ϕ_1 *and* ϕ_2 *are equivalent.*

Proof We prove the soundness of the rule by induction on k. Applying Max-SAT resolution, using as premises the clauses in bold face, when $k=1$,

$$\begin{aligned}
\phi_1 &= \{\mathbf{l_1}, \mathbf{\bar{l}_1} \vee \mathbf{l_2}, \bar{l}_2 \vee l_3, \bar{l}_3 \vee l_4, \bar{l}_2 \vee l_5, \bar{l}_4 \vee \bar{l}_5\} \cup \phi' \\
&= \{l_2, l_1 \vee \bar{l}_2, \bar{l}_2 \vee l_3, \bar{l}_3 \vee l_4, \mathbf{\bar{l}_2} \vee \mathbf{l_5}, \mathbf{\bar{l}_4} \vee \mathbf{\bar{l}_5}\} \cup \phi' \\
&= \{l_2, l_1 \vee \bar{l}_2, \bar{l}_2 \vee l_3, \mathbf{\bar{l}_3} \vee \mathbf{l_4}, \mathbf{\bar{l}_2} \vee \mathbf{\bar{l}_4}, \bar{l}_2 \vee l_4 \vee l_5, l_2 \vee \bar{l}_4 \vee \bar{l}_5\} \cup \phi' \\
&= \{l_2, l_1 \vee \bar{l}_2, \mathbf{\bar{l}_2} \vee \mathbf{l_3}, \mathbf{\bar{l}_2} \vee \mathbf{\bar{l}_3}, \bar{l}_2 \vee l_3 \vee \bar{l}_4, l_2 \vee \bar{l}_3 \vee l_4, \bar{l}_2 \vee l_4 \vee l_5, l_2 \vee \bar{l}_4 \vee \bar{l}_5\} \cup \phi' \\
&= \{\mathbf{l_2}, l_1 \vee \bar{l}_2, \mathbf{\bar{l}_2}, \bar{l}_2 \vee l_3 \vee \bar{l}_4, l_2 \vee \bar{l}_3 \vee l_4, \bar{l}_2 \vee l_4 \vee l_5, l_2 \vee \bar{l}_4 \vee \bar{l}_5\} \cup \phi' \\
&= \{\Box, l_1 \vee \bar{l}_2, \bar{l}_2 \vee l_3 \vee \bar{l}_4, l_2 \vee \bar{l}_3 \vee l_4, \bar{l}_2 \vee l_4 \vee l_5, l_2 \vee \bar{l}_4 \vee \bar{l}_5\} \cup \phi' \\
&= \phi_2
\end{aligned}$$

Assume that Rule 5.7 is sound for $k=n$. Let us prove that it is sound for $k=n+1$. In that case:

$$\phi_1 = \{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3,\ \cdots,\ \bar{l}_{n+1} \vee l_{n+2},\ \bar{l}_{n+2} \vee l_{n+3}, \bar{l}_{n+3} \vee l_{n+4}, \bar{l}_{n+2} \vee l_{n+5},\ \bar{l}_{n+4} \vee \bar{l}_{n+5}\} \cup \phi'$$

By applying Max-SAT resolution between l_1 and $\bar{l}_1 \vee l_2$, we get:

$$\phi_1 = \{l_1 \vee \bar{l}_2,\ l_2,\ \bar{l}_2 \vee l_3,\ \cdots,\ \bar{l}_{n+1} \vee l_{n+2},\ \bar{l}_{n+2} \vee l_{n+3}, \bar{l}_{n+3} \vee l_{n+4}, \bar{l}_{n+2} \vee l_{n+5},\ \bar{l}_{n+4} \vee \bar{l}_{n+5}\} \cup \phi'$$

By applying the induction hypothesis, we get:

$$\phi_1 = \{l_1 \vee \bar{l}_2,\ \Box, l_2 \vee \bar{l}_3,\ \cdots,\ l_{n+1} \vee \bar{l}_{n+2}, \bar{l}_{n+2} \vee l_{n+3} \vee \bar{l}_{n+4}, l_{n+2} \vee \bar{l}_{n+3} \vee l_{n+4},$$

Copia gratuita. Personal free copy http://libros.csic.es

$$, \bar{l}_{n+2} \vee l_{n+4} \vee l_{n+5}, l_{n+2} \vee \bar{l}_{n+4} \vee \bar{l}_{n+5}\} \cup \phi'$$

which is ϕ_2 when $k = n+1$. Therefore, ϕ_1 and ϕ_2 are equivalent and the rule is sound. ■

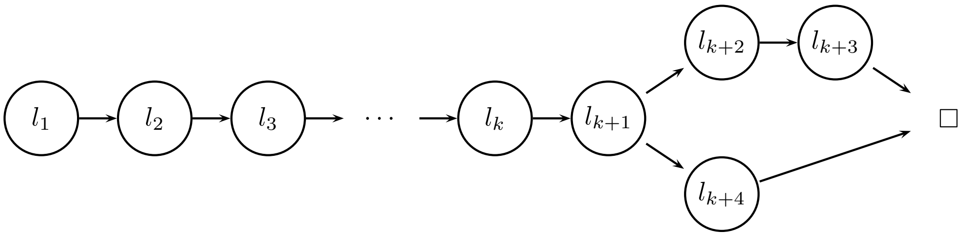


Figure 5.4: Rule 5.7 implication graph.

Rule 5.7 is an extension of Rule 5.6 with one implication more in one of the final implication lines. Figure 5.4 shows the implication graph for Rule 5.7.

Rule 5.8 *If* $\phi_1 = \{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3,\ \cdots,\ \bar{l}_k \vee l_{k+1},\ \bar{l}_{k+1} \vee l_{k+2}, \bar{l}_{k+2} \vee l_{k+3}, \bar{l}_{k+1} \vee l_{k+4}, \bar{l}_{k+4} \vee l_{k+5},\ \bar{l}_{k+3} \vee \bar{l}_{k+5}\} \cup \phi'$ *and* $\phi_2 = \{\Box,\ l_1 \vee \bar{l}_2,\ l_2 \vee \bar{l}_3,\ \cdots,\ l_k \vee \bar{l}_{k+1},\ l_{k+1} \vee \bar{l}_{k+2} \vee l_{k+3},\ \bar{l}_{k+1} \vee l_{k+2} \vee \bar{l}_{k+3},\ l_{k+1} \vee \bar{l}_{k+3} \vee \bar{l}_{k+5},\ \bar{l}_{k+1} \vee l_{k+3} \vee l_{k+5},\ l_{k+1} \vee \bar{l}_{k+4} \vee l_{k+5},\ \bar{l}_{k+1} \vee l_{k+4} \vee \bar{l}_{k+5}\} \cup \phi'$*, then* ϕ_1 *and* ϕ_2 *are equivalent.*

Proof We prove the soundness of the rule by induction on k. Applying Max-SAT resolution, using as premises the clauses in bold face, when $k = 1$,

$$\begin{aligned}
\phi_1 &= \{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3, \bar{l}_3 \vee l_4, \mathbf{\bar{l}_2 \vee l_5}, \mathbf{\bar{l}_5 \vee l_6},\ \bar{l}_4 \vee \bar{l}_6\} \cup \phi' \\
&= \{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3, \bar{l}_3 \vee l_4,\ \mathbf{\bar{l}_4 \vee \bar{l}_6}, \mathbf{\bar{l}_2 \vee l_6}, l_2 \vee l_5 \vee \bar{l}_6, l_2 \vee \bar{l}_5 \vee l_6\} \cup \phi' \\
&= \{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3, \mathbf{\bar{l}_3 \vee l_4},\ \mathbf{\bar{l}_2 \vee \bar{l}_4},\ l_2 \vee \bar{l}_4 \vee \bar{l}_6, \bar{l}_2 \vee l_4 \vee l_6, \bar{l}_2 \vee l_5 \vee \bar{l}_6, \\
&\quad , l_2 \vee \bar{l}_5 \vee l_6\} \cup \phi' \\
&= \{l_1,\ \bar{l}_1 \vee l_2,\ \mathbf{\bar{l}_2 \vee l_3}, \mathbf{\bar{l}_2 \vee \bar{l}_3},\ \bar{l}_2 \vee l_3 \vee \bar{l}_4, l_2 \vee \bar{l}_3 \vee l_4,\ l_2 \vee \bar{l}_4 \vee \bar{l}_6, \bar{l}_2 \vee l_4 \vee l_6, \\
&\quad , \bar{l}_2 \vee l_5 \vee \bar{l}_6, l_2 \vee \bar{l}_5 \vee l_6\} \cup \phi' \\
&= \{l_1,\ \mathbf{\bar{l}_1 \vee l_2},\ \mathbf{\bar{l}_2},\ \bar{l}_2 \vee l_3 \vee \bar{l}_4, l_2 \vee \bar{l}_3 \vee l_4,\ l_2 \vee \bar{l}_4 \vee \bar{l}_6, \bar{l}_2 \vee l_4 \vee l_6, \bar{l}_2 \vee l_5 \vee \bar{l}_6, \\
&\quad , l_2 \vee \bar{l}_5 \vee l_6\} \cup \phi'
\end{aligned}$$

$$\begin{aligned}
&= \{\mathbf{l_1},\ \mathbf{\bar{l}_1},\ l_1 \vee \bar{l}_2,\ \bar{l}_2 \vee l_3 \vee \bar{l}_4, l_2 \vee \bar{l}_3 \vee l_4,\ l_2 \vee \bar{l}_4 \vee \bar{l}_6, \bar{l}_2 \vee l_4 \vee l_6, \bar{l}_2 \vee l_5 \vee \bar{l}_6, \\
&\quad , l_2 \vee \bar{l}_5 \vee l_6\} \cup \phi' \\
&= \{\Box,\ l_1 \vee \bar{l}_2,\ \bar{l}_2 \vee l_3 \vee \bar{l}_4, l_2 \vee \bar{l}_3 \vee l_4,\ l_2 \vee \bar{l}_4 \vee \bar{l}_6, \bar{l}_2 \vee l_4 \vee l_6, \bar{l}_2 \vee l_5 \vee \bar{l}_6, \\
&\quad , l_2 \vee \bar{l}_5 \vee l_6\} \cup \phi' \\
&= \phi_2
\end{aligned}$$

Assume that Rule 5.8 is sound for $k = n$. Let us prove that it is sound for $k = n+1$. In that case:

$$\phi_1 = \{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3,\ \cdots,\ \bar{l}_{n+1} \vee l_{n+2},\ \bar{l}_{n+2} \vee l_{n+3}, \bar{l}_{n+3} \vee l_{n+4}, \bar{l}_{n+2} \vee l_{n+5}, \bar{l}_{n+5} \vee l_{n+6},$$

$$,\ \bar{l}_{n+4} \vee \bar{l}_{n+6}\} \cup \phi'$$

© CSIC © del autor o autores / Todos los derechos reservados

By applying Max-SAT resolution between l_1 and $\bar{l}_1 \vee l_2$, we get:

$$\phi_1 = \{l_1 \vee \bar{l}_2,\ l_2,\ \bar{l}_2 \vee l_3,\ \cdots,\ \bar{l}_{n+1} \vee l_{n+2},\ \bar{l}_{n+2} \vee l_{n+3}, \bar{l}_{n+3} \vee l_{n+4}, \bar{l}_{n+2} \vee l_{n+5}, \bar{l}_{n+5} \vee l_{n+6},$$
$$,\ \bar{l}_{n+4} \vee \bar{l}_{n+6}\} \cup \phi'$$

By applying the induction hypothesis, we get:

$$\phi_1 = \{l_1 \vee \bar{l}_2,\ \Box,\ l_2 \vee \bar{l}_3,\ \cdots,\ l_{n+1} \vee \bar{l}_{n+2},\ l_{n+2} \vee \bar{l}_{n+3} \vee l_{n+4},\ \bar{l}_{n+2} \vee l_{n+3} \vee \bar{l}_{n+4},$$
$$,\ l_{n+2} \vee \bar{l}_{n+4} \vee \bar{l}_{n+6},\ \bar{l}_{n+2} \vee l_{n+4} \vee l_{n+6},\ l_{n+2} \vee \bar{l}_{n+5} \vee l_{n+6},\ \bar{l}_{n+2} \vee l_{n+5} \vee \bar{l}_{n+6}\} \cup \phi'$$

which is ϕ_2 when $k = n + 1$. Therefore, ϕ_1 and ϕ_2 are equivalent and the rule is sound. ■

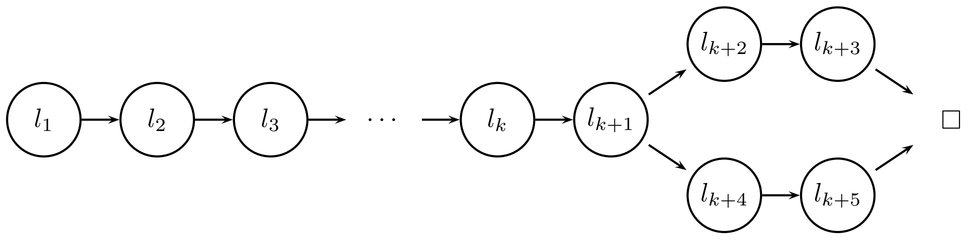


Figure 5.5: Rule 5.8 implication graph.

Rule 5.8 is an extension of Rule 5.6 with one implications more for each final implication line. Figure 5.5 shows the implication graph for Rule 5.8.

Rule 5.9 *If* ϕ_1=$\{l_1,\ l_2,\ l_3,\ \bar{l}_1 \vee \bar{l}_2 \vee \bar{l}_3\} \cup \phi'$ *and* ϕ_2=$\{\Box,\ l_1 \vee l_2,\ l_1 \vee l_3,\ \bar{l}_1 \vee l_2 \vee l_3\} \cup \phi'$*, then* ϕ_1 *and* ϕ_2 *are equivalent.*

Proof Applying Max-SAT resolution using as premises the clauses in bold face,

$$\begin{array}{rcl}
\phi_1 & = & \{l_1,\ \mathbf{l_2},\ l_3,\ \mathbf{\bar{l}_1} \vee \mathbf{\bar{l}_2} \vee \mathbf{\bar{l}_3}\} \cup \phi' \\
\phi_1 & = & \{\mathbf{l_1},\ l_3,\ \mathbf{\bar{l}_1} \vee \mathbf{\bar{l}_3},\ l_1 \vee l_2,\ \bar{l}_1 \vee l_2 \vee l_3\} \cup \phi' \\
\phi_1 & = & \{\mathbf{l_3},\ \mathbf{\bar{l}_3},\ l_1 \vee l_3,\ l_1 \vee l_2,\ \bar{l}_1 \vee l_2 \vee l_3\} \cup \phi' \\
\phi_1 & = & \{\Box,\ l_1 \vee l_3,\ l_1 \vee l_2,\ \bar{l}_1 \vee l_2 \vee l_3\} \cup \phi' \\
 & = & \phi_2
\end{array}$$

■

Rule 5.9 replaces three unit clauses and a ternary clause with an empty clause, two binary clauses and a ternary clause. Note that this rule is the first that has ternary clauses to be replaced.

Rule 5.10 *If* ϕ_1=$\{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3,\ \cdots,\ \bar{l}_{i-1} \vee l_i, l_{i+1},\ \bar{l}_{i+1} \vee l_{i+2},\ \bar{l}_{i+2} \vee l_{i+3},\ \cdots,\ \bar{l}_{j-1} \vee l_j, l_{j+1},\ \bar{l}_{j+1} \vee l_{j+2},\ \bar{l}_{j+2} \vee l_{j+3},\ \cdots,\ \bar{l}_{k-1} \vee l_k,\ \bar{l}_i \vee\ \bar{l}_j \vee\ \bar{l}_k\} \cup \phi'$ *and* ϕ_2=$\{\Box,\ l_1 \vee \bar{l}_2,\ l_2 \vee \bar{l}_3,\ \cdots,\ l_{i-1} \vee \bar{l}_i,\ l_{i+1} \vee \bar{l}_{i+2},\ l_{i+2} \vee \bar{l}_{i+3},\ \cdots,\ l_{j-1} \vee \bar{l}_j,\ l_{j+1} \vee \bar{l}_{j+2},\ l_{j+2} \vee \bar{l}_{j+3},\ \cdots,\ l_{k-1} \vee \bar{l}_k,\ l_i \vee l_j,\ l_i \vee l_k,\ \bar{l}_i \vee l_j \vee l_k\} \cup \phi'$*, then* ϕ_1 *and* ϕ_2 *are equivalent.*

Copia gratuita. Personal free copy http://libros.csic.es

Proof The soundness of this rule follows from the soundness of Rule 5.4 for the linear derivations, and follows from the soundness of Rule 5.9 for the conflict with the ternary clause $(\bar{l}_i \vee \bar{l}_j \vee \bar{l}_k)$ and the unit clauses derived by applying Rule 5.4 to the linear derivations. ■

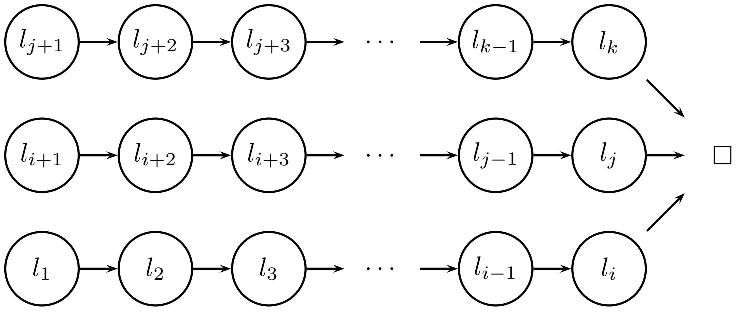


Figure 5.6: Rule 5.10 implication graph.

Rule 5.10 is a combination of linear derivation and Rule 5.9. Figure 5.6 shows the implication graph for Rule 5.10.

5.2.5 Hard learning

Our solvers incorporate a learning module that analyzes the conflicts detected with hard clauses. When a conflict is detected using unit propagation over hard clauses, it analyzes the conflicting clause detected using the 1-UIP learning schema [MMZ$^+$01] implemented in zChaff [ZMMM01], and learns a hard clause. The mission of the learned conflict clauses is to avoid visiting regions of the search space that cannot lead to an optimal solution, due to some violated hard clause.

Since any optimal solution of a Partial Max-SAT instance must satisfy all the hard clauses, the fact of adding redundant hard clauses does not affect the number of unsatisfied soft clauses. So, we can guarantee that the number of unsatisfied clauses is preserved by our clause learning module.

The new learned clause is added to the current list of unsatisfied hard clauses. Then, the conflict analysis module uses the information of the recently added unsatisfied hard clauses to backtrack to a previous decision level that solves the conflict, allowing the solver to perform non-chronological backtracking.

As stated in [MSLM08], existing results indicate that the 1-UIP clause learning procedure, and the associated non-chronological backtracking procedure, may end up doing more backtracking than the original clause learning of GRASP [ZMMM01]. However, zChaff creates significantly fewer clauses and is significantly more effective at backtracking. Figure 5.7 shows an example of the 1-UIP learning schema. The learned clause taking the 1-UIP cut is $\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4$.

© CSIC © del autor o autores / Todos los derechos reservados

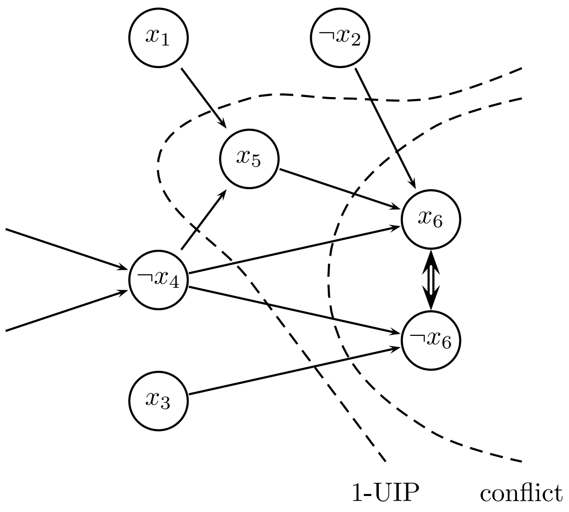


Figure 5.7: Implication graph.

There are several issues to take into account when we incorporate a hard learning module into a branch and bound Partial Max-SAT solver, like the construction of the implication graph. We apply unit propagation on hard clauses at each node of the search tree. When a hard conflict is reached, and only in this case, we build the implication graph from the conflict. We do not build the implication graph in each run of the unit propagation to save CPU time; e.g., in some Partial Max-SAT instances, the hard clauses are only *at-least-one* and *at-most-one* clauses (graph coloring, Max-CSP instances encoded as Partial Max-SAT instances,...), and the unit propagation process hardly ever reaches a conflict.

Another issue to take into account in the implementation of a branch and bound Partial Max-SAT solver with clause learning is the fast search of unit hard clauses. Unit hard clauses are required at each node to perform unit propagation. Having a separated list for hard clauses and unit hard clauses can help improve the performance of unit propagation. One list for all the unit clauses can slow down the process when we have a big ratio of soft clauses per hard clauses in the Partial Max-SAT instance.

As we will see in the experimental investigation, this learning schema produces significant performance improvements. We introduced first this learning schema in [AM06b]. It was, to the best of our knowledge, the first time that learning was incorporated into a branch and bound Partial Max-SAT solver.

Copia gratuita. Personal free copy http://libros.csic.es

5.2.6 Soft learning

The soft learning module analyzes the conflicts detected when at least one of the conflict clauses is soft. For the time being, our soft learning consist in applying Max-SAT resolution to two conflict clauses. These clauses are selected as follows: between all the conflict clauses, we choose the pairs of clauses $x \vee A$ and $\neg x \vee B$ that have the minimum number of literals and, finally, we choose the pair that has the minimum number of different literals among A and B. We give priority to resolve a hard clause and a soft clause.

In the case of Partial Max-SAT, Max-SAT resolution can be simplified when at least one of the premises is hard by applying the next rule, which is called absorption rule in [LH05a]:

$$\begin{array}{c} [D] \\ D \vee D' \\ \hline\hline [D] \end{array}$$

where D and D' are disjunctions of literals.

The calculus formed by the Max-SAT resolution rule and the absorption rule is complete for Partial Max-SAT. This follows from the fact that Max-SAT resolution is complete for Max-SAT and the absorption rule is sound. Another alternative is expressing this calculus by means of Rule 1, Rule 2, and Rule 3 in Figure 5.8. We follow this approach because it is easy to understand in our context.

Rule 1

$$\begin{array}{l} x \vee a_1 \vee \cdots \vee a_s \\ \overline{x} \vee b_1 \vee \cdots \vee b_t \\ \hline\hline a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_t \\ x \vee a_1 \vee \cdots \vee a_s \vee \overline{b_1} \\ x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \overline{b_2} \\ \cdots \\ x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_{t-1} \vee \overline{b_t} \\ \overline{x} \vee b_1 \vee \cdots \vee b_t \vee \overline{a_1} \\ \overline{x} \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \overline{a_2} \\ \cdots \\ \overline{x} \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \cdots \vee a_{s-1} \vee \overline{a_s} \end{array}$$

Rule 2

$$\begin{array}{l} x \vee a_1 \vee \cdots \vee a_s \\ [\overline{x} \vee b_1 \vee \cdots \vee b_t] \\ \hline\hline [\overline{x} \vee b_1 \vee \cdots \vee b_t] \\ a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_t \\ x \vee a_1 \vee \cdots \vee a_s \vee \overline{b_1} \\ x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \overline{b_2} \\ \cdots \\ x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_{t-1} \vee \overline{b_t} \end{array}$$

Rule 3

$$\begin{array}{l} [x \vee a_1 \vee \cdots \vee a_s] \\ [\overline{x} \vee b_1 \vee \cdots \vee b_t] \\ \hline\hline [x \vee a_1 \vee \cdots \vee a_s] \\ [\overline{x} \vee b_1 \vee \cdots \vee b_t] \\ [a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_t] \end{array}$$

Figure 5.8: Resolution for Partial Max-SAT

Actually, our soft learning mechanism applies Rule 1 when both conflict

© CSIC © del autor o autores / Todos los derechos reservados

clauses are soft, and Rule 2 when one conflict clause is hard and the other is soft. When both conflict clauses are hard, it applies the 1-UIP learning schema [MMZ+01].

Example 5.4 *Let us consider a conflict between clauses $x_1 \vee x_2$ and $\neg x_1 \vee x_3$ in a Partial Max-SAT formula ϕ. Both clauses are soft and, if we apply Rule 1, we get:*

$$\phi - \{x_1 \vee x_2, \neg x_1 \vee x_3\} \cup \{x_2 \vee x_3, x_1 \vee x_2 \vee \neg x_3, \neg x_1 \vee x_3 \vee \neg x_2\}$$

Now, let us consider an initial case in which one of the clauses is hard and we have a conflict between $[x_1 \vee x_2]$ and $\neg x_1 \vee x_3$. With a hard clause and a soft clause we apply Rule 2 and get:

$$\phi - \{\neg x_1 \vee x_3\} \cup \{x_2 \vee x_3, \neg x_1 \vee x_3 \vee \neg x_2\}$$

In this example we can observe that, when there is a hard clause in the conflict, the process is simpler than when there are two soft clauses; and the length of the resulting formula applying Rule 2 is smaller than applying Rule 1.

5.2.7 Other learning techniques

When $LB \geq UB$, the solver cannot improve the best solution found so far and can prune the subtree below the current node. In this process, no hard clause is falsified and we cannot apply any hard learning schema. However, we can learn a hard clause taking the reason of all the soft conflicts that make $LB \geq UB$.

This learning schema consists in learning a clause c for every soft conflict detected in every failed branch, where c is a reason of the conflict. When the $LB \geq UB$, we learn one clause for every conflict. Then, we add as a hard clause the disjunction of all the clauses learned in the branch. A similar learning schema for soft conflicts was proposed in solver Clone [PD07]. In any case, the resulting clause is a violated hard clause, which can be used by the standard conflict analysis algorithm to perform non-chronological backtraking.

The main drawback of this approach is that the learned clauses are too big when the minimum number of unsatisfied clauses is not small.

5.3 Preprocessing techniques

There are some techniques which are not efficient if we apply them at each node of the search space. This can be due to the complexity of the technique, to the difficulty to maintain the data structures during the search process, or simply because it is not worth to apply the technique more than once. However, it can make sense to apply these techniques as a preprocessing.

We have designed and implemented a preprocessor that incorporates the following solving techniques:

Copia gratuita. Personal free copy http://libros.csic.es

- **Almost common clause rule:** It applies ACC rule to clauses of arbitrary length. Solvers like MaxSatz apply ACC at each node only to binary clauses.
- **Variable saturation:** It applies Max-SAT Resolution to saturate variables and simplify the search space.
- **Learning and restarts:** It adds to the initial formula learned clauses in several seach spaces.

These techniques consume too much CPU time to be computed at each node.

5.3.1 Almost common clause rule

The ACC rule is defined as follows:

ACC rule: *If* $\phi_1=\{l_1 \vee l_2 \vee \cdots \vee l_k,\ \bar{l}_1 \vee l_2 \vee \cdots \vee l_k\} \cup \phi'$ *and* $\phi_2=\{l_2 \vee \cdots \vee l_k\} \cup \phi'$, *then* ϕ_1 *and* ϕ_2 *are equivalent.*

The preprocess starts with the application of this rule to the whole formula. If it produces any change, we apply the rule again. Otherwise, we reach a state of stagnation and we stop the preprocess for this technique.

Example 5.5 *Let us consider a Partial Max-SAT instance* ϕ *with clauses* $[x_1 \vee x_2 \vee \neg x_3 \vee x_4]$, $\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4$ *and* $\neg x_2 \vee \neg x_3 \vee x_4$. *We start the ACC rule preprocess by applying the rule to* $[x_1 \vee x_2 \vee \neg x_3 \vee x_4]$ *and* $\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4$*:*

$$\phi_1 = \phi - \{\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4\} \cup \{x_2 \vee \neg x_3 \vee x_4\}$$

In a second step, we apply the ACC rule between $x_2 \vee \neg x_3 \vee x_4$ *and* $\neg x_2 \vee \neg x_3 \vee x_4$*:*

$$\phi_2 = \phi_1 - \{x_2 \vee \neg x_3 \vee x_4, \neg x_2 \vee \neg x_3 \vee x_4\} \cup \{\neg x_3 \vee x_4\}$$

Finally, we cannot apply the ACC rule and finish the preprocess. Note that, in the first step, the hard clause $[x_1 \vee x_2 \vee \neg x_3 \vee x_4]$ *is not removed from the formula.*

5.3.2 Variable saturation

The Max-SAT problem can be solved using the algorithm presented in [BLM06]. It is based on Max-SAT Resolution and variable saturation. The pseudocode is shown in Algorithm 5.2.

Given an initial Max-SAT instance ϕ, this algorithm obtains the minimum number of unsatisfied clauses in ϕ. The function `Saturation(`ϕ`,`x`)` computes a saturation of ϕ w.r.t. x using the Rule 1 of Figure 5.8. This inference rule is applied to multisets of clauses, and replaces the premises of the rule by its conclusions. We say that the rule *cuts* the variable x, and the tautologies concluded by the rule are removed from the resulting multiset. In Partial Max-SAT, the hard clauses remain and the clauses subsumed by the hard clause are removed.

© CSIC © del autor o autores / Todos los derechos reservados

Algorithm 5.2: `Max-SAT`(ϕ) : Resolution based Max-SAT algorithm

```
Output: The minimum number of clauses in the Max-SAT instance $\phi$
        that are falsified by an assignment
Function Max-SAT ($\phi$ : Max-SAT formula) : Natural
    $\phi_0 \leftarrow \phi$
    for $i \leftarrow 1$ to $n$ do
        $\phi \leftarrow$ Saturation($\phi_{i-1}$,$x_i$)
        $\langle \phi_i, \psi_i \rangle \leftarrow$ Partition($\phi$,$x_i$)
    return $| \phi_n |$
```

Definition 4 *A multiset of clauses ϕ is said to be* saturated w.r.t. x *if for every pair of clauses $C_1 = x \vee A$ and $C_2 = \neg x \vee B$ of ϕ, there is a literal ℓ such that ℓ is in A and $\bar{\ell}$ is in B. A multiset of clauses ϕ' is a* saturation of ϕ *w.r.t. x if ϕ' is saturated w.r.t. x and $\phi \vdash_x \phi'$; i.e., ϕ' can be obtained from ϕ applying Max-SAT resolution cutting x finitely many times.*

Trivially, by the previous definition, a multiset of clauses ϕ is saturated w.r.t. x if, and only if, every possible application of Max-SAT resolution cutting x only introduces clauses containing x (since tautologies get eliminated).

Lemma 5.1 *[BLM07] For every multiset of clauses ϕ and variable x, there exists a multiset ϕ' such that ϕ' is a saturation of ϕ w.r.t. x. Moreover, this multiset ϕ' can be computed by applying Max-SAT resolution to any pair of clauses $x \vee A$ and $\neg x \vee B$ with the restriction that $A \vee B$ is not a tautology, using any ordering of the literals, until we cannot apply Max-SAT resolution any longer.*

Function `Partition(`ϕ`,`x`)` of Algorithm 5.2 computes a partition of ϕ into the subset of clauses containing x (ψ_i) and the subset of clauses not containing x (ϕ_i). The order on the saturation of the variables can be freely chosen; i.e., the sequence $x_i, \dots, x_n$ can be any enumeration of the variables.

The main drawback of this algorithm is the computational cost of applying variable saturation. When we compute a saturation of the formula w.r.t x, we apply Max-SAT Resolution to clauses containing literals x and $\neg x$ until the subset of clauses ϕ' that contains variable x is saturated. Next, the subset of clauses ϕ' is removed from the formula, but new clauses not containing x have been generated during the saturation process. This makes the variable saturation process harder for the next step, because in general, bigger is the set of clauses containing the variable x and the number of literals per clause, larger is the process of computing the saturated subset of clauses ϕ' that contain x.

Nevertheless, we thought that it would make sense to saturate w.r.t. a limited number of variables as a preprocessing in order to simplify the formula. We select iteratively the variables to be saturated, depending on a parameter k, as follows: We build a graph whose nodes are the Boolean variables occurring in

Copia gratuita. Personal free copy http://libros.csic.es

the instance, and add an edge between two vertices if the variables of the vertices occur in the same clause. We select a variable whose vertex has minimal degree[2] if its degree is smaller than k. This process is repeated until no more variables can be selected.

Example 5.6 *Let us consider a Partial Max-SAT instance ϕ with the following clauses containing variable x_1:*

$$\{\neg x_1, x_1 \vee x_2, x_1 \vee x_3\}$$

To saturate ϕ w.r.t. x_1 we apply Max-SAT Resolution between clauses containing x_1.

$$\begin{aligned} \phi &= \phi' \cup \{\neg \mathbf{x_1}, \mathbf{x_1} \vee \mathbf{x_2}, x_1 \vee x_3\} \\ &= \phi' \cup \{x_2, \neg \mathbf{x_1} \vee \neg \mathbf{x_2}, \mathbf{x_1} \vee \mathbf{x_3}\} \\ &= \phi' \cup \{x_2, \neg x_2 \vee x_3, \neg x_1 \vee \neg x_2 \vee \neg x_3, x_1 \vee x_2 \vee x_3\} \end{aligned}$$

Now, we have a saturated multiset of clauses w.r.t. x_1. We can remove $\neg x_1 \vee \neg x_2 \vee \neg x_3$ and $x_1 \vee x_2 \vee x_3$ from the original formula and variable x_1 is also removed. The final formula after the saturation of ϕ w.r.t. x_1 is:

$$\phi = \phi' \cup \{x_2, \neg x_2 \vee x_3\}$$

As we will see in the experimentation, the application of variable saturation as a preprocessing can speed up the search considerably in some sets of instances.

5.3.3 Learning and restarts

This preprocessing technique is based on learning clauses during a limited period of time, for several runs of a solver, on different branches of the search tree. We start the preprocessing solving the instance with a Partial Max-SAT solver equipped with clause learning. When the given timeout for this run is reached, we restart the solver on a different branch by picking, as first branching variable, the best branching variable that has not been picked in previous runs. Once the preprocessing is finished after a given number of restarts, all the learned clauses are added to the original formula.

Starting the search on different branches in each run leads to explore branches of the search tree that could not be visited by the solver in a single regular run. The information of the conflicts reached in these branches is recorded, and used to avoid visiting branches that could lead to a conflict.

This preprocessing technique does not have a big impact in the performance of the solvers, but as we will see in the experimental investigation, it allows to solve more instances in some sets.

[2]For an undirected graph, the degree of a vertex is the number of edges adjacent to the vertex.

© CSIC © del autor o autores / Todos los derechos reservados

5.4 Partial Max-SAT solvers

In this section we describe two exact branch and bound Partial Max-SAT solvers, PMS and W-MaxSatz, that we have designed and implemented during our research on Partial Max-SAT solving. Since the techniques that they use are explained in Section 5.2, here we only present a short description.

5.4.1 PMS

We introduced PMS in SAT-2007 [AM07]. It is an implementation from scratch of our original idea of applying hard clause learning in Partial Max-SAT [AM06b]. PMS is an unweighted Partial Max-SAT solver and participated in the 2007 Max-SAT Evaluation in the categories of unweighted Max-SAT and unweighted Partial Max-SAT. This solver has been implemented in C++ and has the following features:

- Variable selection heuristic: It implements CS variable selection heuritic.
- Upper bound: It computes the initial upper bound using the P-GSAT algorithm.
- Lower bound: It computes the lower bound using UP with double queue (UP_{DQ}) without failed literal detection.
- Inference rules: It applies CUC at each node, and ACC as preprocessing.
- Hard learning: It implements hard clause learning using the 1-UIP schema.
- Soft learning: It has a module that allows the activation of soft clause learning.

PMS uses specific lists for every type of clauses. It maintains separated lists for hard clauses and soft clauses, and two additional lists for unit hard clauses and unit soft clauses. These lists help perform operations when we are only interested either in hard clauses or in soft clauses. PMS allows the activation of each technique, independently, using flags.

5.4.2 W-MaxSatz

W-MaxSatz has been build on top of the Max-SAT solver MaxSatz [LMP06, LMP07]. It is an adaptation of MaxSatz to solve weighted and Partial Max-SAT problems. W-MaxSatz has several modifications on the data structures to implement efficiently the weighted and Partial Max-SAT versions of all the techniques used in MaxSatz. Moreover, we have improved the solver by adding more techniques used only on Partial Max-SAT solvers. The first version of this solver was introduced in the 2007 Max-SAT Evaluation; it participated in all the categories. This solver has been implemented in C and has the following features:

Copia gratuita. Personal free copy http://libros.csic.es

– Variable selection heuristic: It uses the MaxSatz dynamic variable selection heuristic.

– Upper bound: It computes the initial upper bound using the repository of local search algorithms UBCSAT.

– Lower bound: It computes the lower bound using UP with double queue (UP_{DQ}), and with failed literal detection.

– Inference rules: It implements Rule 5.4 (generalization of Rule 5.2 and Rule 5.3), Rule 5.6 (generalization of Rule 5.5), Rule 5.7, Rule 5.8, and Rule 5.10 (generalization of Rule 5.9).

– Hard learning: It implements hard clause learning using the 1-UIP schema.

W-MaxSatz converts soft clauses into hard clauses when the weight of a soft clause is greater than or equal to the upper bound. This technique is similar to the one used in MiniMaxsat [HLO07]. MiniMaxsat has a special weight, called *top*, that is associated to hard clauses. When a soft clause has a weight greater than or equal to the top, it is treated as a hard clause.

The data structures used in MaxSatz are not suitable for adding permanent clauses in the database. Due to the static design of the MaxSatz data structures, the easy way to allow the addition of permanent clauses without loosing the efficiency of its data structures is to reserve more memory statically. This memory can be easily tuned and adapted to the size of the input instances. In further version of W-MaxSatz we plan to solve this problem with a dynamic memory allocation to adapt the size of the instance to the memory used by the solver.

5.5 Experimental investigation

We now report the experimental investigation we conducted to evaluate the techniques and the performance of our problem solving approach. First, we compare the different learning techniques of the PMS solver and the last improvements added to W-MaxSatz. We then compare our solvers with the best performing state-of-the-art Partial Max-SAT solvers (ChaffBS, ChaffLS, Clone, LB-SAT, MiniMaxsat, SAT4Jmaxsat, SR(w) and Toolbar). Finally, we report the results obtained by applying our preprocessor to several Partial Max-SAT solvers.

All the experiments of this section were performed on a cluster with the following specifications:

– Number of hosts: 80 bi-processor

– Operating System: Rocks Cluster 4.0.0 (Linux 2.6.9)

– Processor: AMD Opteron(tm) Processor 248, 2.2 GHz

– Memory: 1 GB

– Cache: 1 MB

© CSIC © del autor o autores / Todos los derechos reservados

5.5.1 Experiments with PMS

We next report the experimental investigation we conducted to compare the different techniques of our solver PMS. The versions of PMS used for the experimentation are the following:

– **PMS:** It is the version described in Section 5.4.1 without soft learning.

– **PMS+SL:** It is PMS with soft learning.

– **PMS-HL:** It is PMS without hard learning.

We used four sets of benchmarks:

– Random Partial Max-2-SAT instances with 100 variables and a number of clauses ranging from 1000 to 3000, and Partial Max-3-SAT instances with 100 variables and a number of clauses ranging from 200 to 700. These are typical random 2-SAT/3-SAT instances, generated using the generator `mwff` developed by Bart Selman, in which 100 clauses are declared, at random, as hard and the rest are declared as soft.

– Random 2-SoftSAT instances generated with the algorithm described in [HSvdW06]. These instances are harder than random Partial Max-2-SAT instances. We solved instances with 150 variables and 150 hard clauses varying the density from 5 to 15. By density we mean the ratio of number of clauses to number of variables.

– Benchmarks from the SAT-2002 Competition [3]. We used benchmarks from the SAT-2002 Competition because they are not so hard as the benchmarks from subsequent competitions. These are satisfiable instances to which we solved the Max-One problem (i.e., compute the maximum number of variables that can be assigned to true by a satisfying assignment).

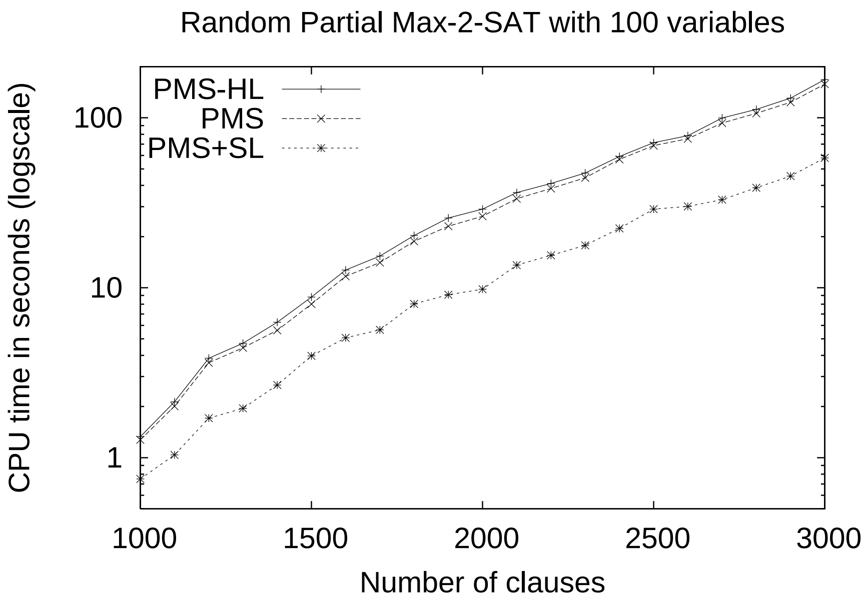
– Random and structured Partial Max-SAT instances from the 2007 Max-SAT Evaluation. There are 722 instances which are divided into 15 sets.

Experiments with random Partial Max-2-SAT and Max-3-SAT instances

The results of solving random Partial Max-2-SAT and random Partial Max-3-SAT instances are shown in Figure 5.9 and Figure 5.10, respectively. We solved 100 instances for each data point. The upper plots display the mean time needed to solve an instance with PMS, PMS-HL and PMS+SL. The lower plots display the mean number of nodes traversed by our solvers. We observe that the best performing solver for Partial Max-2-SAT is PMS+SL while the best performing solvers for Partial Max-3-SAT are PMS and PMS-HL. In this example, it is particularly interesting to observe the performance improvements achieved on Partial Max-2-SAT by incorporating our learning schema of soft clauses.

[3]http://www.satlib.org/Benchmarks/SAT/New/Competition-02/sat-2002-beta.tgz

Copia gratuita. Personal free copy http://libros.csic.es



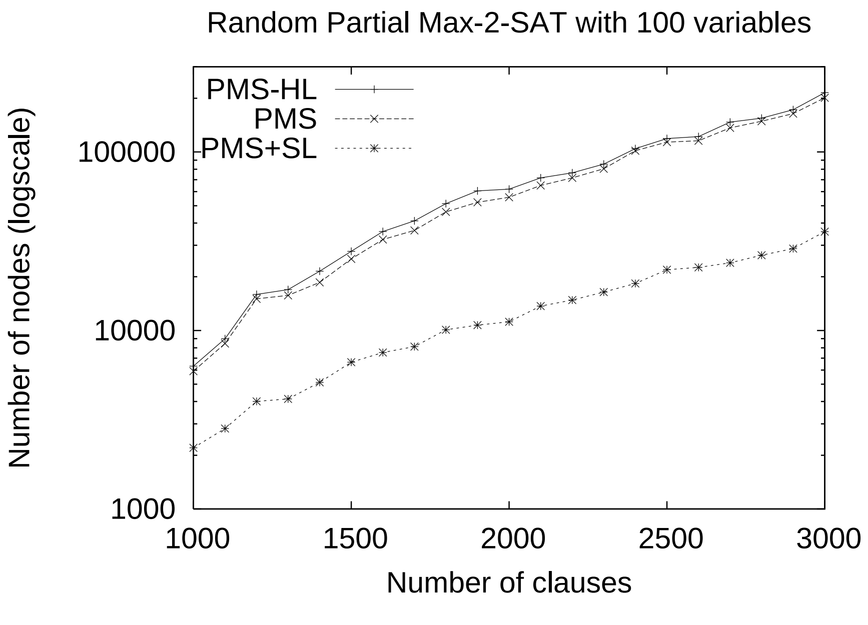
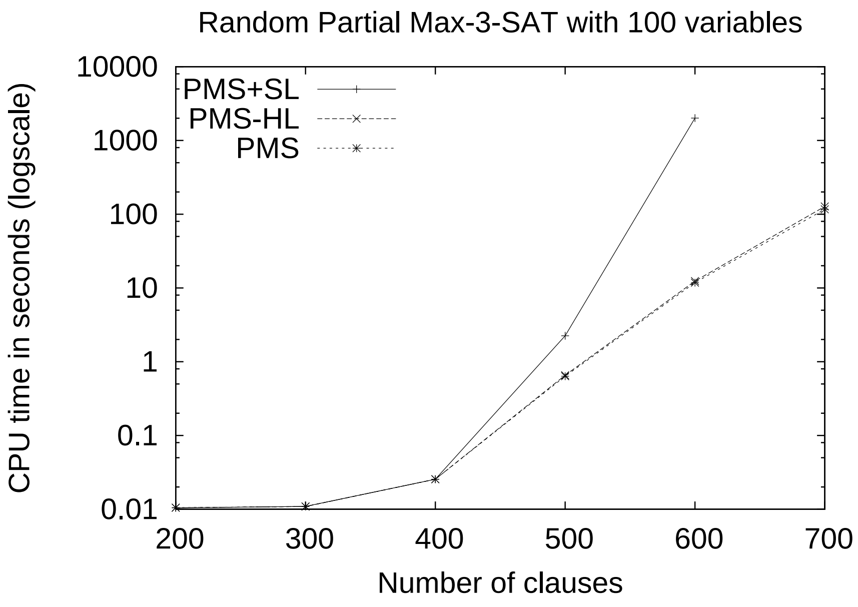


Figure 5.9: Comparison of PMS, PMS-HL and PMS+SL with random Partial Max-2-SAT instances. Mean CPU time in seconds (upper plot) and mean number of nodes (lower plot).

© CSIC © del autor o autores / Todos los derechos reservados



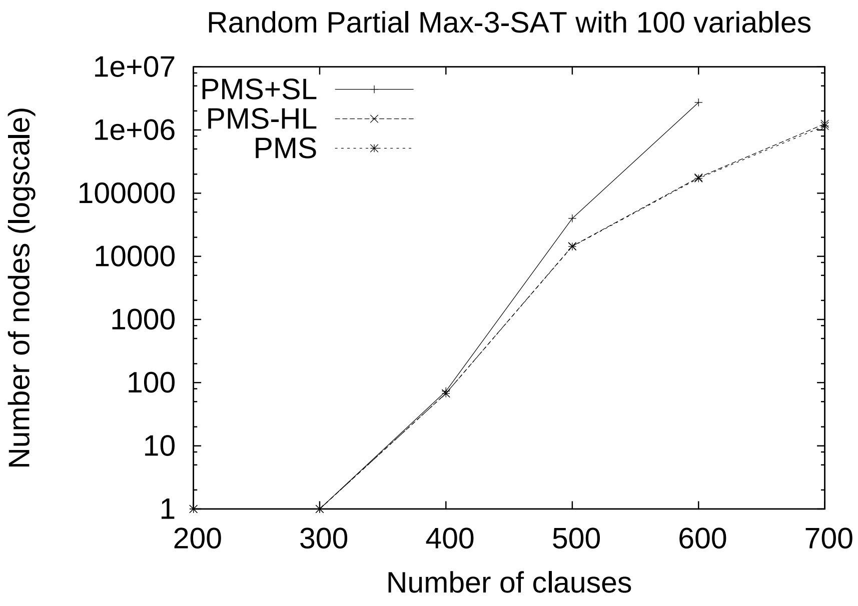


Figure 5.10: Comparison of PMS, PMS-HL and PMS+SL with random Partial Max-3-SAT instances. Mean CPU time in seconds (upper plot) and mean number of nodes (lower plot).

Copia gratuita. Personal free copy http://libros.csic.es

Experiments with random 2-SoftSAT

The results of solving random 2-SoftSAT instances are shown in Figure 5.11. We solved 100 instances for each data point. The upper plot displays the mean CPU time needed to solve an instance with PMS, PMS-HL and PMS+SL. The lower plot displays the mean number of nodes traversed by our solvers. We observe that the best performing solver is PMS+SL, and that when we apply soft learning we get important gains both in time and in number of nodes. The gains in number of nodes are superior to the gains in time due to the overhead of applying learning.

Experiments with SAT-2002 Competition benchmarks

The results of solving the benchmarks from the SAT-2002 Competition, using a timeout of 3600 seconds, are shown in Table 5.1 and Table 5.2. The first column of Table 5.1 shows the name of the set of instances, the second column shows the number of instances in the set, the rest of columns show the median time (among the instances solved within the timeout) needed to solve an instance, and the number of instances solved (in brackets). Table 5.2 is like Table 5.1 but shows number of nodes instead of time for PMS, PMS-HL and PMS+SL. We observe that the best performing solver is PMS and then PMS+SL.

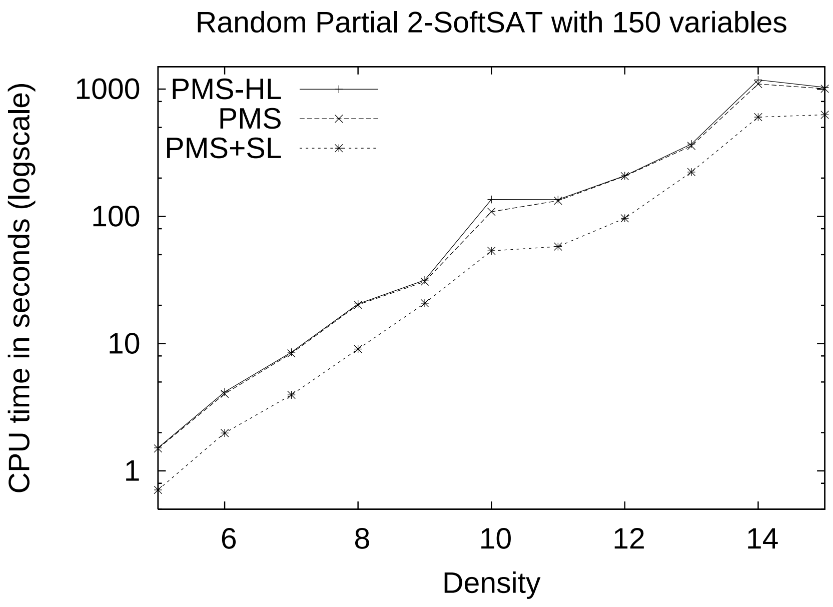
Instance set	#	PMS-HL	PMS	PMS+SL
3-coloring	30	818.62(19)	**241.74(30)**	53.54(20)
AIM	12	91.98(10)	0.41(12)	**0.37(12)**
CNT	6	86.05(1)	155.26(2)	**137.96(2)**
DP	11	594.71(3)	**598.21(4)**	638.60(4)
EZFACT	10	2739.14(1)	**214.10(10)**	69.26(8)
MED	4	4.25(1)	**4.10(1)**	6.72(1)

Table 5.1: Benchmarks from the SAT-2002 Competition solving the Max-One problem. Time in seconds.

Instance set	#	PMS-HL	PMS	PMS+SL
3-coloring	30	2714206(19)	**425872(30)**	93769(20)
AIM	12	3414797(10)	**1842(12)**	1929(12)
CNT	6	650615(1)	137756(2)	**119154(2)**
DP	11	99533(3)	**140412(4)**	178220(4)
EZFACT	10	5774450(1)	**395677(10)**	119923(8)
MED	4	32278(1)	28881(1)	**16236(1)**

Table 5.2: Benchmarks from the SAT-2002 Competition solving the Max-One problem. Number of nodes.

© CSIC © del autor o autores / Todos los derechos reservados



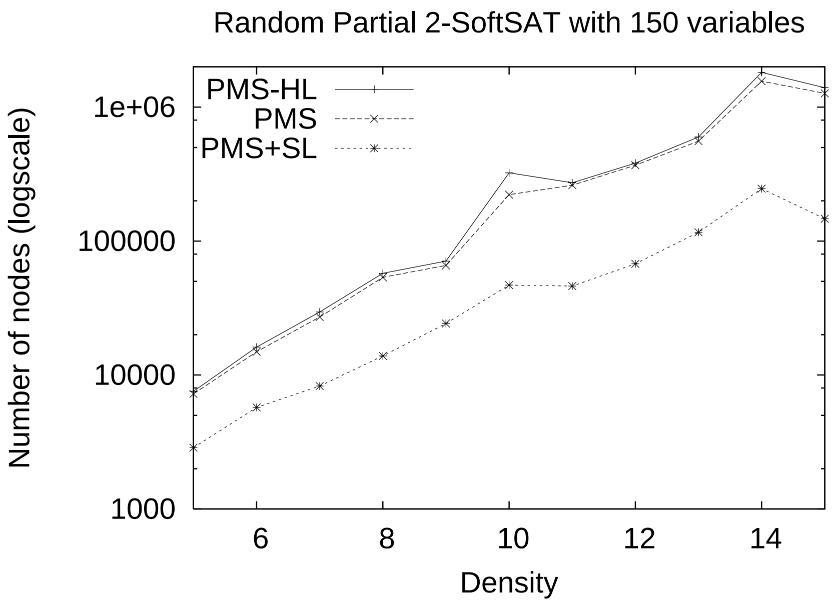


Figure 5.11: Comparison of PMS, PMS-HL and PMS+SL with random 2-SoftSAT instances. Mean CPU time in seconds (upper plot) and mean number of nodes (lower plot).

Copia gratuita. Personal free copy http://libros.csic.es

Figure 5.12 displays the number of instances x from the SAT-2002 Competition that can be solved in y seconds. We observe that the best solver is PMS followed by PMS+SL.

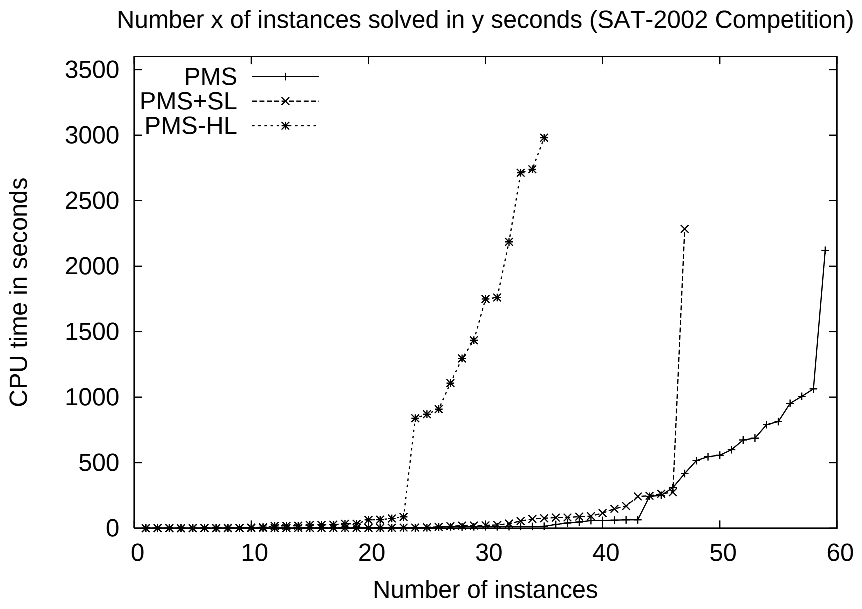


Figure 5.12: Number of instances x that can be solved in y seconds. Instances from the SAT-2002 Competition.

Experiments with 2007 Max-SAT Evaluation benchmarks

The results of solving the benchmarks from the 2007 Max-SAT Evaluation, using a timeout of 3600 seconds, are shown in Table 5.3 and Table 5.4. The best performing solvers is PMS. With these benchmarks, we observe the importance of clause learning. Solvers with learning, PMS and PMS+SL, are able to solve more instances in the following sets: Partial Max-2-SAT (random), Max-One (structured), Pseudo (primes dimacs) and Pseudo (routing). The solver without learning, PMS-HL, has not a good performance with these benchmark.

Figure 5.13 displays the number of instances x from the 2007 Max-SAT Evaluation that can be solved in y seconds. In this case, we observe that the best solvers are PMS followed by PMS+SL.

5.5.2 Experiments with W-MaxSatz

We now report the experimental investigation we conducted to compare the first version of W-MaxSatz with the improvements incorporated into the final version.

© CSIC © del autor o autores / Todos los derechos reservados

Instance set	#	PMS-HL	PMS	PMS+SL
Partial Max-2-SAT (random)	90	224.27(45)	220.26(44)	**360.50(49)**
Partial Max-3-SAT (random)	60	90.04(59)	**80.82(59)**	88.53(43)
Max-Clique (random)	96	69.36(96)	**68.18(96)**	166.86(52)
Max-Clique (structured)	62	**160.83(27)**	171.13(27)	135.45(11)
Max-One (3-SAT)	80	**2.33(80)**	4.23(80)	97.11(41)
Max-One (structured)	60	77.47(13)	**176.70(37)**	178.33(33)
Pseudo (garden)	7	**0.54(5)**	0.55(5)	50.73(5)
Pseudo (logic synthesis)	17	2.54(1)	2.54(1)	**2.50(1)**
Pseudo (primes dimacs)	148	190.37(80)	**124.09(88)**	2.19(74)
Pseudo (routing)	15	0.00(0)	**25.97(5)**	38.87(5)
Weighted CSP (dense loose)	20	2.01(20)	2.02(20)	**1.15(20)**
Weighted CSP (dense tight)	20	2.25(20)	2.24(20)	**1.79(20)**
Weighted CSP (sparse loose)	20	1.40(20)	1.41(20)	**1.02(20)**
Weighted CSP (sparse tight)	20	2.19(20)	2.19(20)	**1.79(20)**
Weighted CSP (w-queens)	7	15.07(7)	**12.94(7)**	2.25(6)

Table 5.3: Benchmarks from the 2007 Max-SAT Evaluation. Time in seconds.

Instance set	#	PMS-HL	PMS	PMS+SL
Partial Max-2-SAT (random)	90	325541(45)	323845(44)	**98163(49)**
Partial Max-3-SAT (random)	60	745911(59)	**667329(59)**	404503(43)
Max-Clique (random)	96	**1691970(96)**	**1691970(96)**	130786(52)
Max-Clique (structured)	62	**2461683(27)**	**2461683(27)**	12114(11)
Max-One (3-SAT)	80	**34006(80)**	38422(80)	106593(41)
Max-One (structured)	60	484758(13)	**257574(37)**	134716(33)
Pseudo (garden)	7	**7096(5)**	**7096(5)**	546033(5)
Pseudo (logic synthesis)	17	170(1)	170(1)	170(1)
Pseudo (primes dimacs)	148	1571426(80)	**253132(88)**	7160(74)
Pseudo (routing)	15	0(0)	**84513(5)**	190193(5)
Weighted CSP (dense loose)	20	33834(20)	33834(20)	**17588(20)**
Weighted CSP (dense tight)	20	22380(20)	22380(20)	**17385(20)**
Weighted CSP (sparse loose)	20	19984(20)	19984(20)	**13613(20)**
Weighted CSP (sparse tight)	20	26052(20)	26052(20)	**20254(20)**
Weighted CSP (w-queens)	7	211489(7)	**163525(7)**	40218(6)

Table 5.4: Benchmarks from the 2007 Max-SAT Evaluation. Number of nodes.

The two versions of the solver used for the experimentation are the following:

- **W-MaxSatz:** It is the version described in Section 5.4.2.
- **W-MaxSatz0:** It is the first working version of the solver. W-MaxSatz uses the same techniques as MaxSatz adapted to deal with weights. Al-

Copia gratuita. Personal free copy http://libros.csic.es

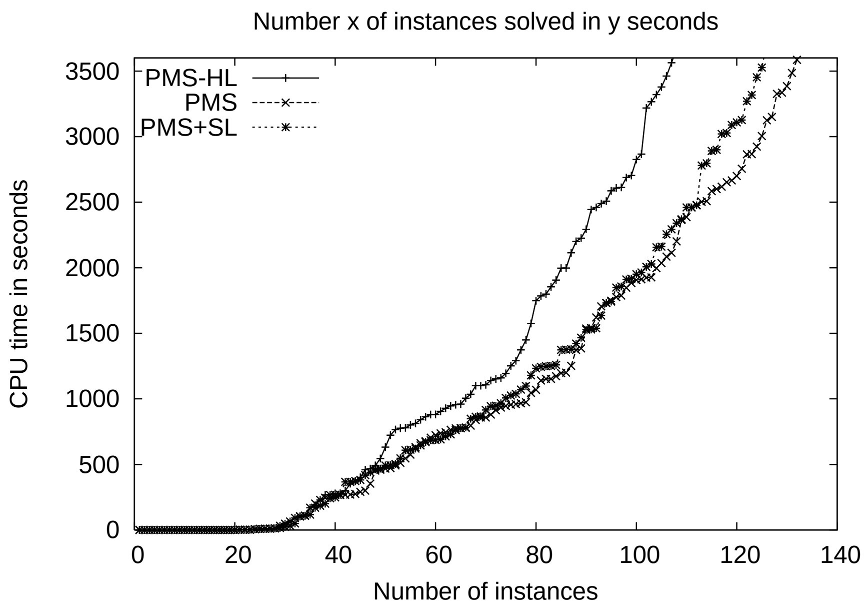


Figure 5.13: Number of instances x that can be solved in y seconds. Instances from the 2007 Max-SAT Evaluation.

though it can deal with Partial Max-SAT instances, it has not the notion of hard and soft clauses. This version of the solver does not use UBCSAT to compute the initial upper bound; it does not propagate hard unit clauses; it does not implement Rule 5.7, Rule 5.8 and Rule 5.10; and it does not incorporate the hard learning module. W-MaxSatz0 was introduced in the 2007 Max-SAT Evaluation and it got very good results, especially on solving random instances in all the categories.

The benchmarks used in the experimentation for W-MaxSatz are random instances to compare the performance of the new inference rules, and the benchmarks used in the categories of weighted and unweighted Partial Max-SAT in the 2007 Max-SAT Evaluation:

- Random and structured Partial Max-SAT instances from the 2007 Max-SAT Evaluation. There are 722 instances which are divided into 15 sets.
- Random and structured weighted Partial Max-SAT instances from the 2007 Max-SAT Evaluation. There are 746 instances which are divided into 11 sets.
- Random weighted Partial Max-2-SAT instances with 150 variables and a number of clauses ranging from 1000 to 5000, These are typical random

© CSIC © del autor o autores / Todos los derechos reservados

2-SAT/3-SAT instances in which 150 clauses are declared, at random, as hard and the rest are declared as soft.

Experiments with Partial Max-SAT instances

The results for Partial Max-SAT instances are shown in Table 5.5. We display the number of instances in each set and, for each solver and each set of instances, we display the mean time in seconds needed to solve an instance and the number of solved instances (in brackets). We set a timeout of 30 minutes. We can see that the best performing solver is W-MaxSatz in almost all the tested instances.

Instance set	#	W-MaxSatz0	W-MaxSatz
Partial Max-2-SAT (random)	90	40.41(90)	**13.28(90)**
Partial Max-3-SAT (random)	60	59.00(60)	**34.65(60)**
Max-Clique (random)	96	49.34(80)	**43.57(80)**
Max-Clique (structured)	62	153.30(22)	**187.08(23)**
Max-One (3-SAT)	80	199.15(77)	**166.13(80)**
Max-One (structured)	60	385.89(54)	**178.38(58)**
Pseudo (garden)	7	2.16(4)	**1.74(4)**
Pseudo (logic synthesis)	17	0.00(0)	0.00(0)
Pseudo (primes dimacs)	148	129.96(85)	**78.29(94)**
Pseudo (routing)	15	143.94(5)	**106.79(5)**
Weighted CSP (dense loose)	20	7.18(20)	**5.50(20)**
Weighted CSP (dense tight)	20	10.52(20)	**9.73(20)**
Weighted CSP (sparse loose)	20	25.55(20)	**16.35(20)**
Weighted CSP (sparse tight)	20	26.03(20)	**24.00(20)**
Weighted CSP (w-queens)	7	85.09(6)	**72.28(6)**

Table 5.5: Results for Partial Max-SAT instances. Mean time in seconds.

Experiments with weighted Partial Max-SAT instances

The results for weighted Partial Max-SAT instances are shown in Table 5.6. We display the number of instances in each set and, for each solver and each set of instances, we display the mean time in seconds to solve an instance and the number of solved instances (in brackets). We set a timeout of 30 minutes. As in the Partial Max-SAT benchmarks, the best performing solver is W-MaxSatz in almost all the tested instances. These results provide empirical evidence that the new solving techniques that we have incorporated into W-MaxSatz produce substantial improvements on a representative sample of instances.

Experiments with random weighted Partial Max-2-SAT instances

In this section we use a special version of W-MaxSatz with the difference that it does not have the new inference rules. We used this version of W-MaxSatz in

Copia gratuita. Personal free copy http://libros.csic.es

Instance set	#	W-MaxSatz0	W-MaxSatz
Weighted Partial Max-2-SAT (random)	90	196.30(88)	**56.63(89)**
Weighted Partial Max-3-SAT (random)	60	91.80(60)	**46.18(60)**
Auctions (paths)	88	243.97(70)	**233.77(71)**
Auctions (regions)	84	6.69(84)	**5.25(84)**
Auctions (scheduling)	84	103.84(82)	**89.76(84)**
Pseudo (factor)	186	**0.43(186)**	11.03(186)
Pseudo (miplib)	16	**1.49(4)**	1.95(4)
Quasigroup Completion	25	37.53(11)	**199.68(15)**
Weighted CSP (planning)	71	101.49(59)	**13.83(71)**
Weighted CSP (spot5 dir)	21	17.35(2)	**14.88(2)**
Weighted CSP (spot5 log)	21	**640.86(4)**	18.98(2)

Table 5.6: Results for weighted Partial Max-SAT instances. Mean time in seconds.

order to asses the performance of the new inference rules we have introduced in this chapter.

Figure 5.14 show the scalability of W-MaxSatz with and without the new inference rules, on random weighted Partial Max-2-SAT instances with 150 variables, 150 hard clauses and soft clauses ranging from 850 to 4850. One hundred instances are solved at a point to compute the displayed CPU mean time. A weighted Partial Max-SAT instance is generated as a weighted instance with hard clauses having weight equal to the addition of its soft clause weights (randomly generated between 1 and 10). The number of hard clauses in an instance corresponds to its number of variables.

We observe that W-MaxSatz with the new inference rules scales better than W-MaxSatz without the new inference rules.

5.5.3 2007 Max-SAT Evaluation

The solvers used in the experimental investigation, and described in Section 3.6, are the following ones:

- **ChaffBS & ChaffLS** (Zhaohui Fu and Sharad Malik)
- **Clone** (Knot Pipatsrisawat, Mark Chavira, Arthur Choi and Adnan Darwiche)
- **LB-SAT** (Han Lin and Kaile Su)
- **MiniMaxsat** (Federico Heras, Javier Larrosa and Albert Oliveras)
- **SAT4Jmaxsat** (Daniel Le Berre)
- **PMS** (Josep Argelich and Felip Manyà)

© CSIC © del autor o autores / Todos los derechos reservados

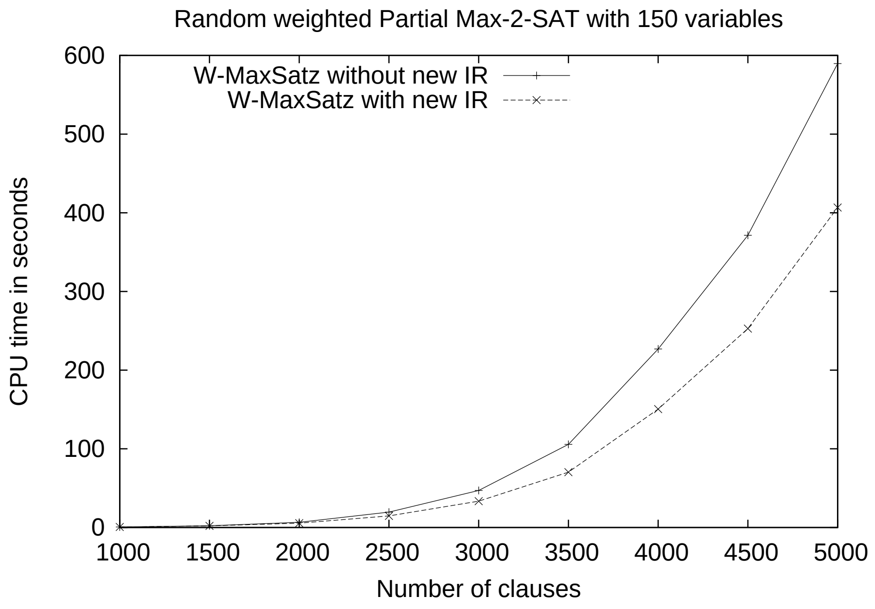


Figure 5.14: Scalability of W-MaxSatz with and without the new inference rules, on random weighted Partial Max-2-SAT instances with 150 variables, 150 hard clauses and number of soft clauses ranging from 850 to 4850. The total clause number ranges from 1000 to 5000.

- **SR(w)** (Miquel Ramírez and Héctor Geffner)
- **ToolBar** (Simon de Givry, Federico Heras, Javier Larrosa and Thomas Schiex)
- **W-MaxSatz** (Josep Argelich, Chu Min Li and Felip Manyà)

Most of the benchmarks used in the evaluation were contributed by F. Heras, J. Larrosa, S. de Givry and T. Schiex [HLdGS07]. All the benchmarks are available online at the 2007 Max-SAT Evaluation web site[4]. Next, we report the results of the 2007 Max-SAT Evaluation in the categories of unweighted and weighted Partial Max-SAT.

Partial Max-SAT category

Table 5.7 show the experimental results of the Partial Max-SAT category of the Max-SAT 2007 Evaluation. The instances are divided into 15 sets. We display the number of instances in each set (#Ins.), then for each solver and for

[4]http://www.maxsat.udl.es/07/

Copia gratuita. Personal free copy http://libros.csic.es

each set of instances, we display the mean time (in seconds) needed to solve an instance of the set within a time limit of 30 minutes and the number of solved instances (in brackets). We can see that MiniMaxsat is the best performing solver on Max-Clique, Max-One and Weighted CSP, and W-MaxSatz is the best on random Partial Max-2-SAT and Partial Max-3-SAT. PMS has a good general performance.

Figure 5.15 globally compares the performance of the solvers in the Partial Max-SAT category of the 2007 Max-SAT Evaluation instances in Table 5.7. Each point (x, y) in a curve shows the number x of instances that the corresponding solver is able to solve within y seconds. In other words, each of the x instances is solved within y seconds (the total run time for these x instances may be larger than y seconds), y being limited to 30 minutes to solve an instance. The solver that solves more instances is MiniMaxsat followed by W-MaxSatz and PMS.

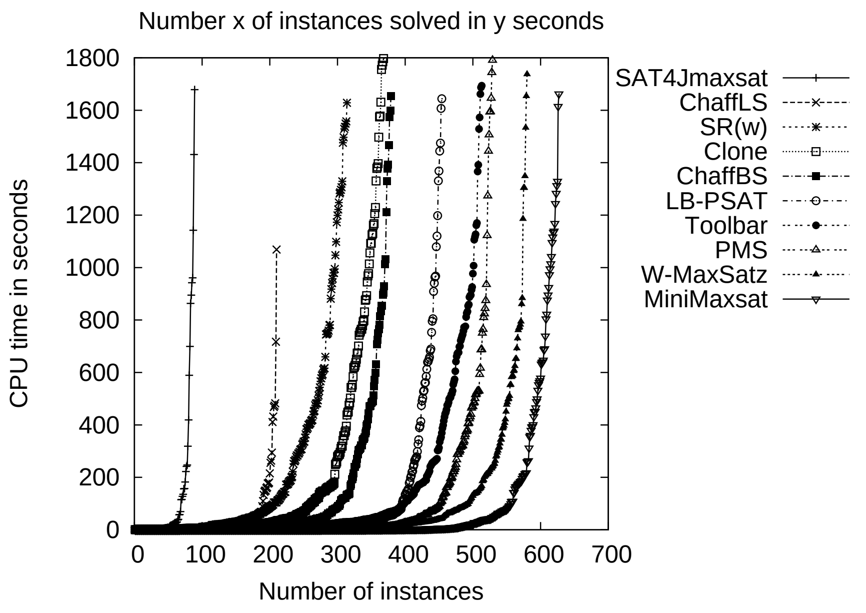


Figure 5.15: Number x of instances that can be solved in y seconds. Partial Max-SAT category.

Figure 5.16 and Figure 5.17 show the scalability of the three fastest solvers[5] in the Partial Max-SAT category of the 2007 Max-SAT Evaluation on random Partial Max-2-SAT instances with 150 variables, 150 hard clauses and soft clauses ranging from 850 to 4850, and on random Partial Max-3-SAT instances with 100 variables, 100 hard clauses and soft clauses ranging from 200 to 700. One hundred instances are solved at each point to compute the displayed CPU mean

[5]We pick the first and second best solvers for Partial Max-2-SAT and Partial Max-3-SAT.

© CSIC © del autor o autores / Todos los derechos reservados

Instance set	#Ins.	ChaffBS	ChaffLS	Clone	LB-PSAT	MiniMaxsat	PMS	SAT4Jmaxsat	SR(w)	Toolbar	W-MaxSatz
Partial Max-2-SAT	90	0.00(0)	0.00(0)	8.19(1)	305.92(59)	221.56(83)	220.26(44)	0.00(0)	0.00(0)	149.86(89)	**13.28(90)**
Partial Max-3-SAT	60	40.24(24)	22.44(22)	251.62(19)	52.42(59)	156.46(58)	80.82(59)	4.57(20)	327.61(16)	172.31(47)	**34.65(60)**
Max-Clique (random)	96	146.24(54)	0.00(0)	189.65(79)	9.89(96)	**2.38(96)**	68.18(96)	0.00(0)	225.37(55)	11.38(96)	43.57(80)
Max-Clique (structured)	62	282.82(19)	54.44(9)	308.72(16)	128.33(32)	**85.26(36)**	171.13(27)	13.16(1)	19.35(9)	202.67(33)	187.08(23)
Max-One (3-SAT)	80	402.14(23)	11.67(41)	420.66(54)	62.17(76)	**1.29(80)**	4.23(80)	1013.93(5)	273.87(70)	102.34(80)	166.13(80)
Max-One (structured)	60	52.97(57)	81.21(2)	258.19(32)	2.29(2)	**31.03(60)**	176.70(37)	412.66(3)	443.59(22)	221.30(44)	178.38(58)
Pseudo (garden)	7	1.33(5)	0.78(5)	2.59(5)	**0.46(5)**	7.13(5)	0.55(5)	1.42(3)	2.54(5)	1.81(4)	1.74(4)
Pseudo (logic synthesis)	17	39.41(2)	**32.16(4)**	0.00(0)	865.73(3)	216.28(2)	2.54(1)	0.00(0)	0.00(0)	0.00(0)	0.00(0)
Pseudo (primes dimacs)	148	72.92(99)	41.24(46)	89.71(99)	82.67(35)	**88.14(107)**	124.09(88)	82.10(45)	67.03(77)	68.70(60)	78.29(94)
Pseudo (routing)	15	**180.32(15)**	0.21(14)	19.08(5)	0.00(0)	93.88(14)	25.97(5)	0.00(0)	0.00(0)	0.00(0)	106.79(5)
Weighted CSP (dense loose)	20	324.93(14)	143.86(6)	831.09(1)	1.15(20)	**0.65(20)**	2.02(20)	0.00(0)	588.36(1)	336.70(15)	5.50(20)
Weighted CSP (dense tight)	20	65.82(20)	106.80(18)	25.89(20)	2.87(20)	**0.68(20)**	2.24(20)	0.00(0)	199.93(18)	461.83(20)	9.73(20)
Weighted CSP (sparse loose)	20	19.16(20)	41.79(19)	122.27(13)	1.86(20)	**0.35(20)**	1.41(20)	222.86(10)	264.08(16)	4.18(10)	16.35(20)
Weighted CSP (sparse tight)	20	28.86(20)	16.23(19)	29.57(20)	7.14(20)	**0.84(20)**	2.19(20)	0.00(0)	219.98(19)	20.36(10)	24.00(20)
Weighted CSP (wqueens)	7	13.94(7)	18.93(5)	80.48(4)	**5.25(7)**	0.49(6)	12.94(7)	11.16(2)	45.17(6)	12.73(5)	72.28(6)

Table 5.7: Results in Partial Max-SAT category. Mean time in seconds.

Copia gratuita. Personal free copy http://libros.csic.es

time. A Partial Max-SAT instance is generated as a weighted instance with hard clauses having weight equal to the number of its soft clauses. The number of hard clauses in an instance corresponds to its number of variables. The soft clauses have weight 1.

In Figure 5.16 we observe that W-MaxSatz is up to one order of magnitude faster than Toolbar, and in Figure 5.17 we observe that W-MaxSatz is almost 3 times faster than LB-PSAT in the hardest region of the plot.

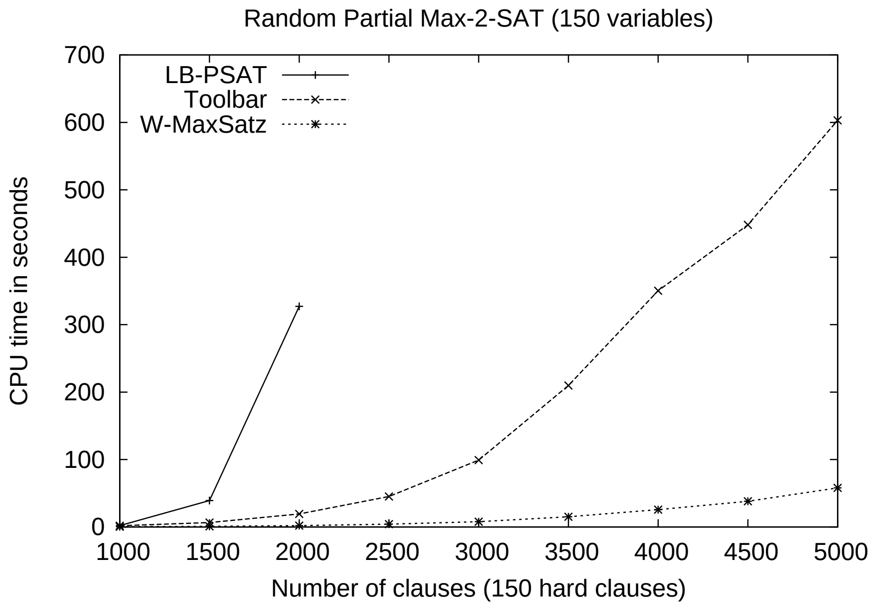


Figure 5.16: Scalability of the three fastest solvers in Partial Max-SAT category on random Partial Max-2-SAT instances with 150 variables, 150 hard clauses and number of soft clauses ranging from 850 to 4850. The total clause number ranges from 1000 to 5000.

Weighted Partial Max-SAT category

The solvers used in this experimentation are the ones that can handle weighted Partial Max-SAT instances. The difference with Partial Max-SAT instances is that we have a weight associated to each soft clauses. The solvers that can solve these instances are Clone, MiniMaxsat, SAT4Jmaxsat, SR(w), Toolbar and W-MaxSatz.

Table 5.8 show the experimental results of the weighted Partial Max-SAT category of the Max-SAT 2007 Evaluation. The instances are divided into 11 sets. We display the number of instances in each set (#Ins.), then for each solver and for each set of instances, we display the mean time in seconds to solve an

© CSIC © del autor o autores / Todos los derechos reservados

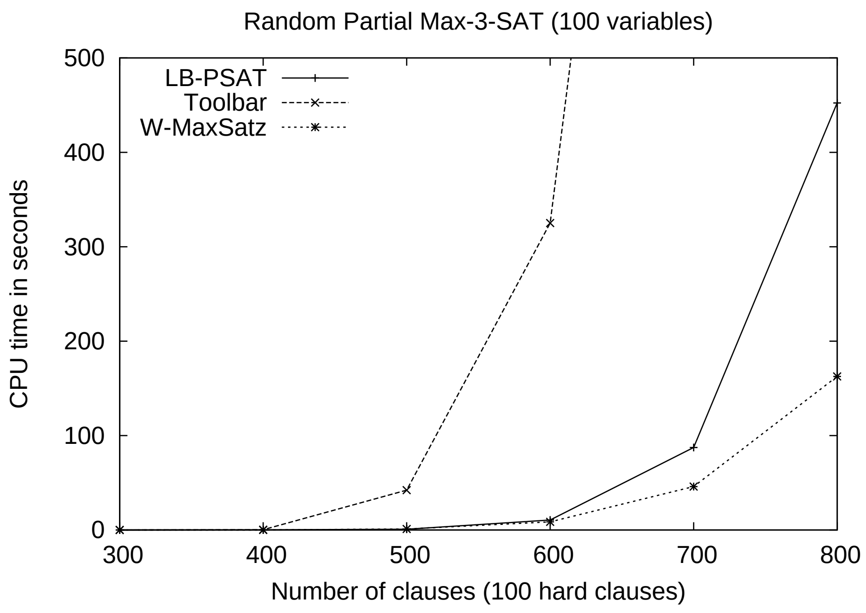


Figure 5.17: Scalability of the three fastest solvers in Partial Max-SAT category on random Partial Max-3-SAT instances with 100 variables, 100 hard clauses and number of soft clauses ranging from 200 to 700. The total clause number ranges from 300 to 800.

instance of the set within a time limit of 30 minutes and the number of solved instances (in brackets). We can see that W-MaxSatz is the best performing solver on random weighted Partial Max-2-SAT and Max-3-SAT instances; SR(w) is the best for Pseudo (miplib), WCSP (spot5 dir) and WCSP (spot5 log), and MiniMaxsat is the best for the remaining instances.

Figure 5.18 globally compares the performance of the solvers in the weighted Partial Max-SAT category of the 2007 Max-SAT Evaluation instances in Table 5.8. Each point $(x,\, y)$ in a curve shows the number x of instances that the corresponding solver is able to solve within y seconds. The solver that solves more instances is MiniMaxsat closely followed by W-MaxSatz.

Figure 5.19 and Figure 5.20 show the scalability of the three fastest solvers[6] in the weighted Partial Max-SAT category of the 2007 Max-SAT Evaluation on random weighted Partial Max-2-SAT instances with 150 variables, 150 hard clauses and soft clauses ranging from 850 to 3850, and on random weighted Partial Max-3-SAT instances with 100 variables, 100 hard clauses and soft clauses ranging from 200 to 700. One hundred instances are solved at each point to

[6]We pick the first and second best solvers for weighted Partial Max-2-SAT and weighted Partial Max-3-SAT.

Copia gratuita. Personal free copy http://libros.csic.es

Instance set	#	Clone	Minimaxsat	SAT4Jmaxsat	SR(w)	Toolbar	W-MaxSatz
Weighted Partial Max-2-SAT	90	0.00(0)	246.27(81)	0.00(0)	0.00(0)	213.23(88)	**56.63(89)**
Weighted Partial Max-3-SAT	60	136.27(21)	186.63(58)	6.41(20)	275.44(17)	188.74(47)	**46.18(60)**
Auctions (paths)	88	50.78(88)	**31.55(88)**	0.00(0)	163.45(77)	48.68(88)	233.77(71)
Auctions (regions)	84	30.50(84)	**1.60(84)**	0.00(0)	130.19(82)	6.44(84)	5.25(84)
Auctions (scheduling)	84	228.15(74)	**46.21(84)**	0.00(0)	231.83(55)	74.10(82)	89.76(84)
Pseudo (factor)	186	9.84(186)	**1.17(186)**	598.29(55)	0.00(0)	246.39(12)	11.03(186)
Pseudo (miplib)	16	132.41(5)	41.66(5)	6.74(3)	**244.84(6)**	2.92(4)	1.95(4)
Quasigroup Completion	25	0.00(0)	**25.00(20)**	377.01(14)	652.49(5)	191.07(12)	199.68(15)
Weighted CSP (planning)	71	261.14(62)	**9.97(71)**	73.22(16)	365.47(52)	22.81(52)	13.83(71)
Weighted CSP (spot5 dir)	21	9.31(6)	3.83(3)	0.56(1)	**2.91(6)**	128.04(5)	14.88(2)
Weighted CSP (spot5 log)	21	7.27(5)	9.18(4)	0.54(1)	**14.90(6)**	111.41(4)	18.98(2)

Table 5.8: Results in weighted Partial Max-SAT category. Mean time in seconds.

© CSIC © del autor o autores / Todos los derechos reservados

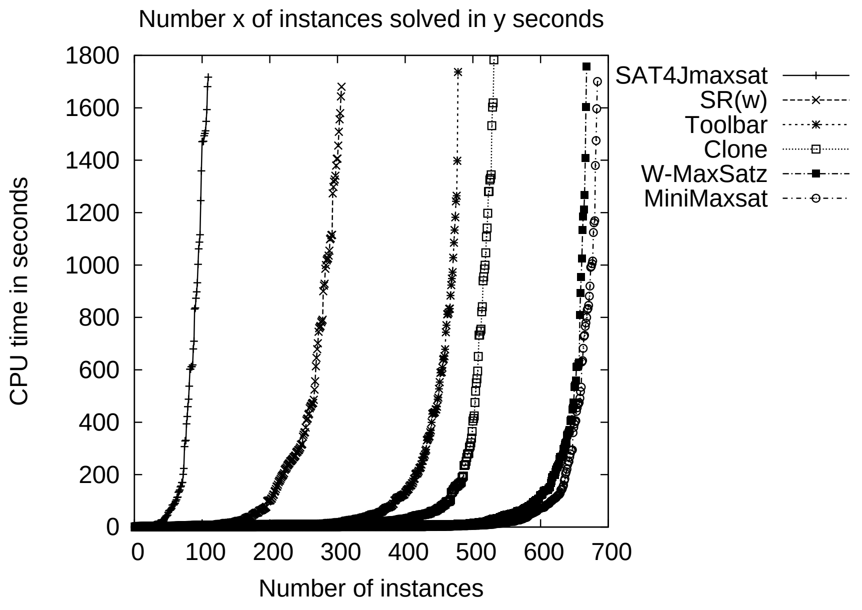


Figure 5.18: Number x of instances that can be solved in y seconds. Weighted Partial Max-SAT category.

compute the displayed CPU mean time. A weighted Partial Max-SAT instance is generated as a weighted instance with hard clauses having weight equal to the addition of its soft clause weights (randomly generated between 1 and 10). The number of hard clauses in an instance corresponds to its number of variables.

In Figure 5.19 we observe that W-MaxSatz is 3 times faster than Toolbar, and in Figure 5.20 W-MaxSatz is about 6 times faster than MiniMaxsat in the hardest region of the plot.

5.5.4 Experiments with preprocessing

Preprocessing with variable saturation

To assess the impact of the preprocessor with variable saturation on the performance of branch and bound Partial Max-SAT solvers, we solved instances[7] of the 2007 Max-SAT Evaluation (with a timeout of 30 minutes as in the evaluation) on five of the most successful and representative state-of-the-art solvers: MiniMaxsat, W-MaxSatz, SR(w), Clone and ChaffBS. The versions of the solvers are the same as in Section 5.5.3.

[7]We solved only the instances in which the preprocessor detected variables that could be saturated with a value of $k = 6$.

Copia gratuita. Personal free copy http://libros.csic.es

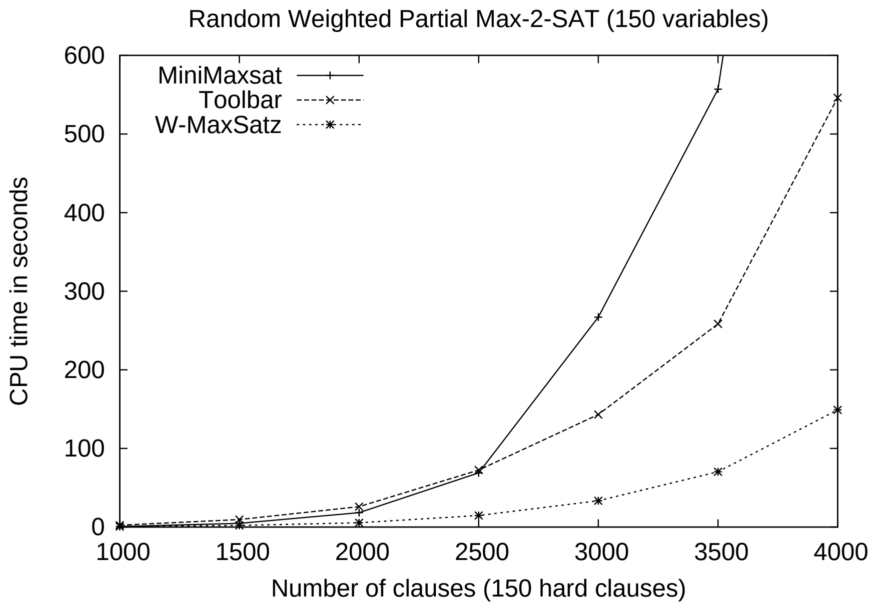


Figure 5.19: Scalability of the three fastest solvers in weighted Partial Max-SAT category on random weighted Partial Max-2-SAT instances with 150 variables, 150 hard clauses and number of soft clauses ranging from 850 to 3850. The total clause number ranges from 1000 to 4000.

We select iteratively the variables to be saturated, depending on a parameter k, as follows: We build a graph whose nodes are the Boolean variables occurring in the instance, and add an edge between two vertices if the variables of the vertices occur in the same clause. We select a variable whose vertex has minimal degree if its degree is smaller than k. We executed the preprocessor with $k = 6, 10, 14$.

Tables 5.9 and 5.10 show the experimental results for W-MaxSatz. The instances are divided into sets. The first column is the name of the set, the second column shows the number of instances in each set, the third column shows the results for the solver without preprocessing, and the rest of columns show the results with preprocessing for $k = 6, 10, 14$. We display the mean time (in seconds) of the solved instances, as well as the number of solved instances (in brackets). We observe that W-MaxSatz with preprocessing solves more instances in 5 sets, and reduces considerably the CPU time in most of the other sets. The best improvements are achieved for Max-Clique (random), where the preprocessing allows to solve 8 additional instances, and for Auctions (paths), where the preprocessing allows to solve 9 additional instances.

Tables 5.11 and 5.12 show the results for MiniMaxsat. In this case, the gains are not so significant as for W-MaxSatz, although the preprocessing allows to

© CSIC © del autor o autores / Todos los derechos reservados

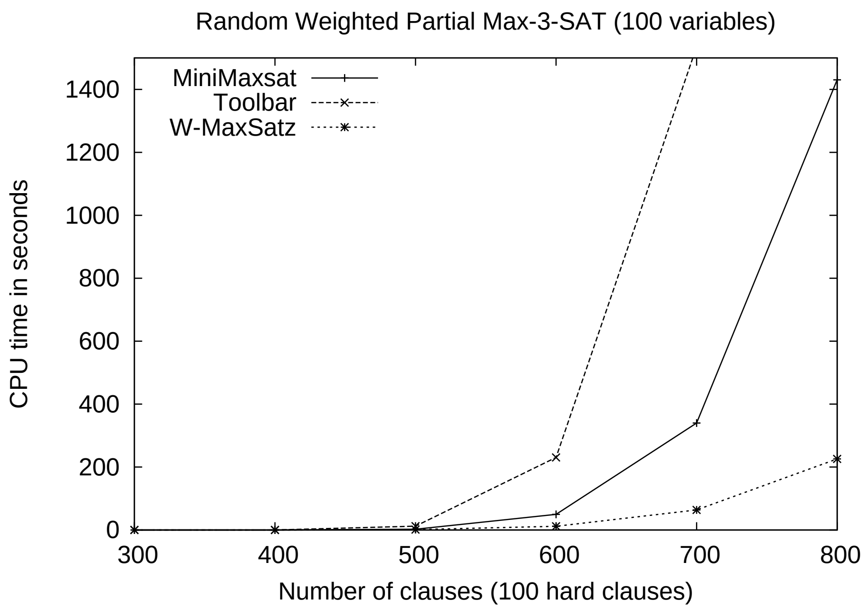


Figure 5.20: Scalability of the three fastest solvers in weighted Partial Max-SAT category on random weighed Partial Max-3-SAT instances with 100 variables, 100 hard clauses and number of soft clauses ranging from 200 to 700. The total clause number ranges from 300 to 800.

Instance set	#	W-MaxSatz	$k = 6$	$k = 10$	$k = 14$
Max-Clique (random)	96	43.57(80)	69.30(83)	61.04(85)	**53.85(88)**
Max-Clique (struc.)	62	187.08(23)	183.30(24)	178.03(24)	**171.13(25)**
Max-One (3-SAT)	50	261.95(50)	122.08(50)	**62.06(50)**	328.07(48)
Max-One (structured)	60	**178.38(58)**	234.56(56)	223.76(42)	6.57(1)
W-CSP (dense loose)	20	5.50(20)	5.26(20)	**3.39(20)**	8.31(20)
W-CSP (dense tight)	20	9.73(20)	9.76(20)	**7.83(20)**	12.95(20)
W-CSP (sparse loose)	20	16.35(20)	9.18(20)	**4.77(20)**	36.51(19)
W-CSP (sparse tight)	20	24.00(20)	21.70(20)	**18.07(20)**	84.81(20)
W-CSP (wqueens)	7	72.28(6)	72.19(6)	**72.17(6)**	72.18(6)

Table 5.9: Partial Max-SAT benchmarks with variable saturation as preprocessing for W-MaxSatz.

solve 1 additional instance for Max-Clique (structured) and for Weighted CSP (spot5 dir).

Tables 5.13 and 5.14 show the experimental results for SR(w). In this case, we solve an additional instance for 3 sets (Weighted CSP (dense loose), Weighted

Copia gratuita. Personal free copy http://libros.csic.es

Instance set	#	W-MaxSatz	$k = 6$	$k = 10$	$k = 14$
Auctions (paths)	88	233.77(71)	**178.50(80)**	127.72(77)	266.47(63)
Auctions (regions)	84	**5.25(84)**	5.30(84)	5.52(84)	5.62(84)
Auctions (schedu.)	84	89.76(84)	89.62(84)	89.66(84)	**89.61(84)**
Pseudo (factor)	186	**11.03(186)**	11.64(186)	226.88(186)	924.37(2)
Pseudo (miplib)	16	1.95(4)	**0.96(4)**	190.93(4)	2.34(1)
Quasigroup Com.	25	199.68(15)	**199.36(15)**	199.46(15)	199.52(15)
W-CSP (planning)	71	**13.83(71)**	21.97(71)	63.65(70)	233.29(42)
W-CSP (spot5 dir)	21	14.88(2)	6.59(5)	**57.96(6)**	13.27(5)
W-CSP (spot5 log)	21	18.98(2)	91.03(3)	2.55(4)	**1.46(4)**

Table 5.10: Weighted Partial Max-SAT benchmarks with variable saturation as preprocessing for W-MaxSatz.

Instance set	#	MiniMaxsat	$k = 6$	$k = 10$	$k = 14$
Max-Clique (random)	96	**2.41(96)**	2.44(96)	2.67(96)	4.38(96)
Max-Clique (struc.)	62	85.22(36)	82.15(37)	**67.94(37)**	66.43(36)
Max-One (3-SAT)	50	**0.37(50)**	0.40(50)	0.43(50)	8.87(50)
Max-One (structured)	60	**31.35(60)**	20.57(54)	65.88(42)	0.78(1)
W-CSP (dense loose)	20	**0.65(20)**	0.71(20)	0.87(20)	5.11(20)
W-CSP (dense tight)	20	**0.69(20)**	0.70(20)	0.70(20)	2.87(20)
W-CSP (sparse loose)	20	**0.35(20)**	0.36(20)	0.57(20)	21.20(20)
W-CSP (sparse tight)	20	**0.85(20)**	0.87(20)	0.94(20)	27.05(20)
W-CSP (wqueens)	7	55.47(7)	55.28(7)	**54.56(7)**	179.13(7)

Table 5.11: Partial Max-SAT benchmarks with variable saturation as preprocessing for MiniMaxsat.

Instance set	#	MiniMaxsat	$k = 6$	$k = 10$	$k = 14$
Auctions (paths)	88	29.82(88)	**19.44(88)**	13.52(84)	78.21(75)
Auctions (regions)	84	1.63(84)	**1.55(84)**	**1.55(84)**	1.56(84)
Auctions (schedu.)	84	**46.14(84)**	46.24(84)	46.28(84)	46.16(84)
Pseudo (factor)	186	**1.16(186)**	1.79(186)	5.53(186)	905.51(183)
Pseudo (miplib)	16	**41.35(5)**	84.90(5)	398.55(5)	1.43(1)
Quasigroup Com.	25	25.00(20)	26.71(20)	25.28(20)	**24.65(20)**
W-CSP (planning)	71	**9.97(71)**	10.11(71)	22.12(71)	235.45(47)
W-CSP (spot5 dir)	21	2.63(3)	11.82(3)	8.18(4)	**6.99(4)**
W-CSP (spot5 log)	21	**9.07(4)**	5.69(2)	152.16(3)	323.82(4)

Table 5.12: Weighted Partial Max-SAT benchmarks with variable saturation as preprocessing for MiniMaxsat.

© CSIC © del autor o autores / Todos los derechos reservados

CSP (w-queens) and Auctions (scheduling)), and 185 additional instances for Pseudo (factor). The latter is the best improvement achieved with our preprocessor.

Instance set	#	SR(w)	$k = 6$	$k = 10$	$k = 14$
Max-Clique (ran.)	96	244.85(55)	219.40(55)	224.65(55)	**218.38(55)**
Max-Clique (struc.)	62	21.18(9)	**17.56(9)**	22.67(8)	20.17(8)
Max-One (3-SAT)	50	386.23(41)	**338.69(41)**	718.76(22)	758.61(1)
Max-One (struc.)	60	**471.72(22)**	449.33(19)	618.92(18)	1078.54(1)
W-CSP (den. loose)	20	697.74(1)	633.31(1)	**1162.49(2)**	0.00(0)
W-CSP (den. tight)	20	209.22(18)	**199.18(18)**	202.71(18)	350.83(15)
W-CSP (spa. loose)	20	296.48(16)	**272.89(16)**	408.06(15)	853.86(7)
W-CSP (spa. tight)	20	235.98(19)	**216.19(19)**	230.31(19)	563.63(12)
W-CSP (wqueens)	7	54.00(6)	230.25(7)	**228.06(7)**	258.10(7)

Table 5.13: Partial Max-SAT benchmarks with variable saturation as preprocessing for SR(w).

Instance set	#	SR(w)	$k = 6$	$k = 10$	$k = 14$
Auctions (paths)	88	**173.42(77)**	161.15(76)	169.32(72)	353.90(66)
Auctions (regions)	84	146.54(82)	136.45(82)	126.93(82)	**119.52(82)**
Auctions (schedu.)	84	276.91(56)	240.61(56)	**270.71(57)**	239.26(56)
Pseudo (factor)	186	0.00(0)	2.86(37)	**520.50(185)**	1091.88(1)
Pseudo (miplib)	16	**2.62(5)**	3.04(4)	216.89(4)	4.12(1)
Quasigroup Com.	25	715.58(5)	**572.40(5)**	675.34(5)	674.26(5)
W-CSP (planning)	71	**379.57(57)**	371.42(53)	286.88(46)	285.58(25)
W-CSP (spot dir)	21	2.95(6)	**1.90(6)**	9.27(4)	61.92(3)
W-CSP (spot log)	21	14.56(6)	**11.53(6)**	25.30(5)	10.83(4)

Table 5.14: Weighted Partial Max-SAT benchmarks with variable saturation as preprocessing for SR(w).

Tables 5.15 and 5.16 show the experimental results for Clone. In this case, we solve an additional instance for 2 sets (Max-Clique (structured) and Max-One (3-SAT)), 3 additional instances in Max-One (structured), 9 additional instances in Weighted CSP (dense loose), 4 additional instances in Weighted CSP (sparse loose), and 2 additional instances in Weighted CSP (planning). As we can see, using the preprocessor with this solver, we are able to solve more instances in 5 of 9 sets for Partial Max-SAT benchmarks.

Table 5.17 shows the experimental results for ChaffBS. In this case, we solve an additional instance in Max-One (3-SAT), and 2 additional instances in Weighted CSP (dense loose).

We can observe that for high values of k, and in some sets, the performance

Copia gratuita. Personal free copy http://libros.csic.es

Instance set	#	Clone	$k = 6$	$k = 10$	$k = 14$
Max-Clique (ran.)	96	**185.77(78)**	141.85(75)	143.05(75)	165.74(76)
Max-Clique (struc.)	62	295.56(16)	395.55(17)	**382.63(17)**	426.71(17)
Max-One (3-SAT)	50	794.98(22)	**860.28(23)**	820.86(16)	892.24(2)
Max-One (struc.)	60	170.54(31)	**278.67(34)**	516.66(20)	873.49(1)
W-CSP (den. loose)	20	817.39(1)	1515.55(1)	**407.70(10)**	957.94(2)
W-CSP (den. tight)	20	**24.19(20)**	27.05(20)	33.96(20)	144.94(20)
W-CSP (spa. loose)	20	103.00(13)	155.97(13)	**172.92(17)**	391.15(10)
W-CSP (spa. tight)	20	29.76(20)	**27.99(20)**	35.96(20)	552.75(15)
W-CSP (wqueens)	7	68.76(4)	104.99(4)	52.93(4)	**39.27(4)**

Table 5.15: Partial Max-SAT benchmarks with variable saturation as preprocessing for Clone.

Instance set	#	Clone	$k = 6$	$k = 10$	$k = 14$
Auctions (paths)	88	**53.15(88)**	56.00(88)	101.75(84)	287.21(71)
Auctions (regions)	84	32.37(84)	**31.21(84)**	31.38(84)	33.07(84)
Auctions (schedu.)	84	231.75(74)	**227.56(74)**	229.95(74)	229.21(74)
Pseudo (factor)	186	**9.27(186)**	9.91(186)	65.11(186)	0.00(0)
Pseudo (miplib)	16	127.69(5)	**126.97(5)**	190.09(4)	22.85(1)
Quasigroup Com.	25	0.00(0)	0.00(0)	0.00(0)	0.00(0)
W-CSP (planning)	71	327.37(64)	293.55(63)	**357.51(66)**	314.27(33)
W-CSP (spot dir)	21	9.37(6)	**5.06(6)**	115.84(6)	270.39(5)
W-CSP (spot log)	21	**28.80(6)**	75.53(6)	207.84(6)	261.96(5)

Table 5.16: Weighted Partial Max-SAT benchmarks with variable saturation as preprocessing for Clone.

Instance set	#	ChaffBS	$k = 6$	$k = 10$	$k = 14$
Max-Clique (ran.)	96	151.91(54)	147.72(54)	149.39(54)	**146.96(54)**
Max-Clique (struc.)	62	**214.21(18)**	253.15(18)	296.48(17)	299.66(16)
Max-One (3-SAT)	50	550.42(5)	**688.97(6)**	481.18(3)	578.34(1)
Max-One (struc.)	60	**54.40(57)**	27.29(56)	28.40(40)	0.00(0)
W-CSP (den. loose)	20	336.87(14)	**605.54(16)**	597.94(11)	437.78(8)
W-CSP (den. tight)	20	64.99(20)	**62.96(20)**	63.95(20)	222.22(20)
W-CSP (spa. loose)	20	**18.30(20)**	26.73(20)	31.61(20)	276.64(18)
W-CSP (spa. tight)	20	**27.38(20)**	28.46(20)	32.77(20)	346.08(16)
W-CSP (wqueens)	7	13.57(7)	14.11(7)	**13.43(7)**	14.48(7)

Table 5.17: Partial Max-SAT benchmarks with variable saturation as preprocessing for ChaffBS.

© CSIC © del autor o autores / Todos los derechos reservados

of this preprocessing technique is not so good. This can be due to that the size of the clauses of the resulting formula are too big or to the CPU time needed for preprocessing. With low values of k, the loose of performance is not so big.

Preprocessing with restarts and learning

To assess the impact of the preprocessor with restarts and learning on the performance of branch and bound Partial Max-SAT solvers, we solved instances of the 2007 Max-SAT Evaluation (with a timeout of 30 minutes as in the evaluation) on our solvers: PMS and W-MaxSatz. The versions of the solvers are the same as in Section 5.5.3. We executed the preprocessor with 1 run of 1 second ($r = 1, t = 1$), 10 runs of 1 second ($r = 10, t = 1$) and 5 runs of 2 seconds ($r = 5, t = 2$).

Tables 5.18 and 5.19 show the experimental results for PMS and W-MaxSatz, respectively. The instances are divided into sets. The first column is the name of the set, the second column shows the number of instances in each set, the third column shows the results for the solver without preprocessing, and the rest of columns show the results with preprocessing for $r = 1$ and $t = 1$, $r = 10$ and $t = 1$, and $r = 5$ and $t = 2$. We display the mean time (in seconds) of the solved instances, as well as the number of solved instances (in brackets). We observe that PMS with this preprocessing solves 2 additional instance in random Partial Max-2-SAT and one more instance in Pseudo (primes cnf), and W-MaxSatz with preprocessing is able to solve one more instance in Pseudo (primes cnf), and it is almost two times faster solving Pseudo (routing) instances.

Instance set	#	PMS	$r = 1, t = 1$	$r = 10, t = 1$	$r = 5, t = 2$
Partial Max-2-SAT	90	220.26(44)	214.63(45)	**238.48(46)**	224.29(45)
Partial Max-3-SAT	60	**80.82(59)**	81.44(59)	86.98(59)	86.63(59)
Max-Clique (ran.)	96	**68.18(96)**	69.09(96)	77.96(96)	76.85(96)
Max-Clique (struc.)	62	171.13(27)	**170.76(27)**	176.61(27)	175.62(27)
Max-One (3-SAT)	80	**4.23(80)**	6.45(80)	21.39(80)	19.39(80)
Max-One (struc.)	60	176.70(37)	120.51(36)	152.67(37)	**152.43(37)**
Pseudo (garden)	7	**0.55(5)**	0.77(5)	3.31(4)	3.24(4)
Pseudo (logic syn.)	17	**2.54(1)**	3.59(1)	12.85(1)	12.74(1)
Pseudo (primes cnf)	148	124.09(88)	106.94(87)	111.84(87)	**120.31(89)**
Pseudo (routing)	15	25.97(5)	24.94(5)	29.46(5)	**23.43(5)**
W-CSP (den. loose)	20	**2.02(20)**	2.73(20)	10.13(20)	8.12(20)
W-CSP (den. tight)	20	**2.24(20)**	3.20(20)	12.12(20)	10.68(20)
W-CSP (spa. loose)	20	**1.41(20)**	1.95(20)	6.42(20)	5.43(20)
W-CSP (spa. tight)	20	**2.19(20)**	3.15(20)	11.71(20)	9.86(20)
W-CSP (wqueens)	7	**12.94(7)**	61.45(7)	93.60(7)	173.76(7)

Table 5.18: Partial Max-SAT benchmarks with restarts and learning as preprocessing for PMS.

Copia gratuita. Personal free copy http://libros.csic.es

Instance set	#	W-MaxSatz	$r = 1, t = 1$	$r = 10, t = 1$	$r = 5, t = 2$
Partial Max-2-SAT	90	**13.28(90)**	14.25(90)	23.12(90)	23.08(90)
Partial Max-3-SAT	60	34.65(60)	**32.40(60)**	38.53(60)	37.45(60)
Max-Clique (ran.)	96	**43.57(80)**	44.60(80)	53.57(80)	51.89(80)
Max-Clique (struc.)	62	187.08(23)	**177.22(23)**	191.38(23)	192.71(23)
Max-One (3-SAT)	80	**166.13(80)**	170.76(80)	195.80(80)	185.81(80)
Max-One (struc.)	60	**178.38(58)**	134.79(57)	165.62(57)	175.68(57)
Pseudo (garden)	7	**1.74(4)**	1.76(4)	2.48(3)	2.39(3)
Pseudo (logic syn.)	17	0.00(0)	0.00(0)	0.00(0)	0.00(0)
Pseudo (primes cnf)	148	78.29(94)	**76.84(95)**	80.03(94)	83.23(95)
Pseudo (routing)	15	106.79(5)	70.17(5)	55.22(5)	**54.97(5)**
W-CSP (den. loose)	20	**5.50(20)**	6.26(20)	13.59(20)	11.51(20)
W-CSP (den. tight)	20	**9.73(20)**	10.71(20)	19.61(20)	17.89(20)
W-CSP (spa. loose)	20	16.35(20)	**15.05(20)**	19.51(20)	18.58(20)
W-CSP (spa. tight)	20	**24.00(20)**	24.97(20)	33.53(20)	31.48(20)
W-CSP (wqueens)	7	**72.28(6)**	93.48(6)	26.97(5)	22.48(5)

Table 5.19: Partial Max-SAT benchmarks with restarts and learning as preprocessing for W-MaxSatz.

5.6 Summary

We have presented the two Partial Max-SAT solvers we have designed and implemented, and provided empirical evidence that they are competitive. These solvers exploit the fact of knowing which clauses are declared to be hard and which clauses are declared to be soft, and incorporate conflict clause learning.

We have also show the advantages of using Partial Max-SAT solvers over weighted Max-SAT solvers when solving problems with hard and soft constraints. On the one hand, we can exploit the learning of modern SAT solvers in the Max-SAT context. As we have seen in the experimental investigation, learning hard clauses produces significant performance improvements on a variety of instances. On the other hand, hard clauses allow to apply a more efficient inference, as well as to compute lower bounds of better quality: (i) the Max-SAT resolution rule is simpler when at least one of the premises is hard; (ii) unit propagation can be enforced on unit hard clauses (while unit propagation on soft clauses is unsound); (iii) a branch of the proof tree can be pruned as soon as a hard clause is violated; and (iv) further inconsistencies can be detected in lower bound UP due to the fact that hard clauses used to derive one contradiction can be used again to derive additional contradictions.

We have introduced new inference rules that improve the performance of Partial Max-SAT solvers. The rules transform the formula into an equivalent formula with a larger number of empty clauses.

We have defined, to the best of our knowledge, the first hard learning schema for branch and bound Partial Max-SAT solvers. The experimental investigation

© CSIC © del autor o autores / Todos los derechos reservados

we conducted shows that the best performing solvers of the 2007 Max-SAT Evaluation incorporate this learning technique.

Another contribution is that we have defined, to the best of our knowledge, the first learning schema for soft clauses, and shown that it accelerates the search for an optimal solution on some instances. As we can see in the experiments, it has been particularly useful when solving Partial Max-2-SAT instances. When we look at number of nodes instead of time, we observe that learning soft clauses is superior to learning just hard clauses in a number of instances. We believe that it is worth to design and implement more efficient procedures for learning soft clauses.

It is worth to notice that we have also discussed how Max-SAT resolution can be simplified in the context of Partial Max-SAT, obtaining a complete resolution-style calculus for Partial Max-SAT which is simpler than the calculus for Max-SAT.

Copia gratuita. Personal free copy http://libros.csic.es

Chapter 6

Conclusions

In this disseration, we have focused on two Max-SAT formalisms, Soft-SAT and Partial Max-SAT, for solving over-constrained problems in which some clauses are hard and some clauses are soft. We have argued that Soft-SAT and Partial Max-SAT formalisms are well suited for representing hard and soft constraints, and we have adapted and introduced new techniques to accelerate the search of Soft-SAT and Partial Max-SAT solvers.

The main contributions of this research on those aspects can be summarized as follows:

- We have defined the Soft-SAT formalism, which allows to encode over-constrained problems in a natural and compact way. Soft-SAT encodes constraints as blocks of clauses without needing to introduce auxiliary variables, and declares each block either as hard or soft. Soft-SAT solvers use the notion of blocks to get more propagation at certain nodes and identify clauses of violated blocks that are not relevant for further checks.

- We have introduced new techniques for solving Soft-SAT problems that take into account the structure behind the Boolean encoding, such as branching techniques and underestimation techniques. These techniques have been incorporated into the Soft-SAT solvers we have designed and implemented, and we have provided empirical evidence that, on some types of instances, these techniques have better performance than the techniques that do not take into account the structure of the encoding.

- We have conducted an empirical comparison of our Soft-SAT solvers with other solvers for over-constrainted problems. We can conclude that our approach is much better than reducing over-constrained problems to weighted Partial Max-SAT for some classes of problems, and that the Soft-SAT solvers we have designed and implemented are competitive compared with the current state-of-the-art solvers developed in the Constraint Programming community.

Copia gratuita. Personal free copy http://libros.csic.es

- We have extended, to Partial Max-SAT, existing solving techniques for SAT and Max-SAT. The most important SAT technique incorporated into our Partial Max-SAT solvers is clause learning derived from the analysis of conflicts detected with hard clauses, which is incorporated in the best Partial Max-SAT solvers of the 2007 Max-SAT Evaluation. The most important Max-SAT techniques incorporated into our solvers are the computation methods of good quality lower bounds, which include the computation of underestimations using unit propagation enhanced with failed literal detection, and the application of sound inference rules. Such techniques allow to use hard clauses more than once to increase the lower bound. We have also defined new inference rules for Partial Max-SAT.

- We have developed new techniques for Partial Max-SAT such as variable selection heuristics that take into account the size of the clause in which the variable appears and if the clause is hard or soft, learning from soft conflicts using Max-SAT resolution, and preprocessing techniques. The first preprocessing technique is based on variable saturation, and it helps to reduce the search space by removing variables from the initial formula. The second preprocessing technique is based on restarts and learning, and adds to the initial formula a set of learned clauses.

- We have provided empirical evidence that the techniques described in this thesis help improve the performance of the solvers we have designed and implemented for Partial Max-SAT. The results obtained also show that our solvers are competitive with the current state-of-the-art Partial Max-SAT solvers, specially W-MaxSatz with random benchmarks.

- We have conducted an empirical evaluational of the new preprocessing techniques we have described. Preprocessing with variable saturation and preprocessing with learning and restarts can improve the performance of the solvers in some sets of instances.

There are many extensions to the current work, but we consider that the feasible points to be exploited in the near future are:

- We plan to extend the language of soft CNF formulas to capture fuzzy constraints, to define alternative notions of "the solution that best respects the constraints of the problem", to incorporate more advanced variable selection heuristics, and to investigate how the techniques developed for dealing with soft constraints in the Constraint Programming community could be adapted to our framework.

- New techniques developed for Partial Max-SAT in the recent years can be adapted to the Soft-SAT formalism. We believe that the performance of our Soft-SAT solvers can be improved by incorporating hard and soft learning, and lower bound computation methods based on unit propagation and failed literal detection, and by applying inference rules.

© CSIC © del autor o autores / Todos los derechos reservados

- We plan to incorporate into PMS additional Max-SAT inference rules used in W-MaxSatz, as well as to define new learning schemas for soft clauses.
- Preprocessing with variable saturation can be improved by incorporating failed literal detection after the preprocessing. The detection of failed literals can help eliminate more variables from the initial formula.
- Study the impact that different encodings of a same problem have in Partial Max-SAT solvers, and identify properties of encodings that can help improve the performance of solvers. A first step in this direction is our recent work at SAT-2008 [ACLM08].

Copia gratuita. Personal free copy http://libros.csic.es

Index

Copia gratuita. Personal free copy http://libros.csic.es

Bibliography

[ACLM08] Josep Argelich, Alba Cabiscol, Inês Lynce, and Felip Manyà. Modelling max-csp as partial max-sat. In *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing, SAT-2008, Guangzhou, P. R. China*, pages 1–15. Springer LNCS 4996, 2008.

[AM06a] Josep Argelich and Felip Manyà. Exact Max-SAT solvers for over-constrained problems. *Journal of Heuristics*, 12(4–5):375–392, 2006.

[AM06b] Josep Argelich and Felip Manyà. Learning hard constraints in Max-SAT. In *Proceedings of the Workshop on Constraint Solving and Constraint Logic Porgramming, CSCLP-2006, Lisbon, Portugal*, pages 1–12, 2006.

[AM07] Josep Argelich and Felip Manyà. Partial max-sat solvers with clause learning. In *Proceedings of 10th International Conference on Theory and Applications of Satisfiability Testing, SAT-2007, Lisbon, Portugal*, pages 28–40. Springer LNCS 4501, 2007.

[AMP03] Teresa Alsinet, Felip Manyà, and Jordi Planes. Improved branch and bound algorithms for Max-SAT. In *Proceedings of the 6th International Conference on the Theory and Applications of Satisfiability Testing*, 2003.

[AMP04] Teresa Alsinet, Felip Manyà, and Jordi Planes. A Max-SAT solver with lazy data structures. In *Proceedings of the 9th Ibero-American Conference on Artificial Intelligence, IBERAMIA 2004, Puebla, México*, pages 334–342. Springer LNCS 3315, 2004.

[AMP05] Teresa Alsinet, Felip Manyà, and Jordi Planes. Improved exact solver for weighted Max-SAT. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing, SAT-2005, St. Andrews, Scotland*, pages 371–377. Springer LNCS 3569, 2005.

Copia gratuita. Personal free copy http://libros.csic.es

[AMP08] Teresa Alsinet, Felip Manyà, and Jordi Planes. An efficient solver for Weighted Max-SAT. *Journal of Global Optimization*, 41:61–73, 2008.

[Anj05] Miguel F. Anjos. Semidefinite optimization approaches for satisfiability and maximu-satisfiability problems. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:1–47, 2005.

[ASM06] Fadi Aloul, Karem Sakallah, and Igor Markov. Efficient symmetry breaking for boolean satisfiability. *IEEE Transactions on Computers*, 55(2):549–558, 2006.

[Ber] Daniel Le Berre. Sat4j, a satisfiability library for java. http://www.sat4j.org.

[BF99] Brian Borchers and Judith Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2:299–306, 1999.

[BGS99] Laure Brisoux, Eric Gregoire, and Lakhdar Sais. Improving backtrack search for sat by means of redundancy. In *Foundations of Intelligent Systems, 11th International Symposium, (ISMIS-99)*, pages 301–309, 1999.

[BK02] Armin Biere and Wolfgang Kunz. Sat and atpg: Boolean engines for formal hardware verification. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD-2002, San Jose, California, USA*, pages 782–785. ACM, 2002.

[BLM06] María Bonet, Jordi Levy, and Felip Manyà. A complete calculus for Max-SAT. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing, SAT-2006, Seattle, USA*, pages 240–251. Springer LNCS 4121, 2006.

[BLM07] María Bonet, Jordi Levy, and Felip Manyà. Resolution for Max-SAT. *Artificial Intelligence*, 171(8-9):606–618, 2007.

[BM00] Ramón Béjar and Felip Manyà. Solving the round robin problem using propositional logic. In *Proceedings of the 17th National Conference on Artificial Intelligence, AAAI-2000, Austin/TX, USA*, pages 262–266, 2000.

[BR99] Nikhil Bansal and Venkatesh Raman. Upper bounds for MaxSat: Further improved. In *Proceedings of the 10th International Symposium on Algorithms and Computation, ISAAC'99, Chennai, India*, pages 247–260. Springer LNCS 1741, 1999.

© CSIC © del autor o autores / Todos los derechos reservados

[BS94] Belaid Benhamou and Lakhdar Sais. Tractability through symmetries in propositional calculus. *Journal of Automatic Reasoning*, 12(1):89–102, 1994.

[BS97] Roberto J. Bayardo and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97, Providence/RI, USA*, pages 203–208. AAAI Press, 1997.

[CA93] James M. Crawford and Larry D. Auton. Experimental results on the crossover point in satisfiability problems. In *Proceedings of the 11th National Conference on Artificial Intelligence, AAAI'93, Washington, D.C., USA*, pages 21–27. AAAI Press, 1993.

[CA96] James M. Crawford and Larry D. Auton. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, 81:31–57, 1996.

[CdGS07] Martin C. Cooper, Simon de Givry, and Thomas Schiex. Optimal soft arc consistency. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, 2007*, pages 68–73, 2007.

[CGLR96] James M. Crawford, Matthew L. Ginsberg, Eugene Luck, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Proceedings ot the 5th International Conference on Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, 1996.

[CIKM97] Byungki Cha, Kazuo Iwama, Yahiko Kambayashi, and Shuichi Miyazaki. Local search algorithms for partial MAXSAT. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97, Providence/RI, USA*, pages 263–268. AAAI Press, 1997.

[CS00] Philippe Chatalic and Laurent Simon. Zres: The old davis-putnam procedure meets zbdd. In David McAllester, editor, *17th International Conference on Automated Deduction (CADE'17)*, number 1831 in LNCS, pages 449–454, 2000.

[Cul] Joseph Culberson. Graph coloring page: The flat graph generator. http://web.cs.ualberta.ca/˜joe/Coloring/Generators/flat.html.

[DABC93] Olivier Dubois, Pascal André, Yacine Boufkhad, and Jaques Carlier. Can a very simple algorithm be efficient for solving sat problem? In *Proc. of the DIMACS Challenge II Workshop*, 1993.

[Dar] Adnan Darwiche. c2d compiler. http://reasoning.cs.ucla.edu/c2d/.

Copia gratuita. Personal free copy http://libros.csic.es

[DD01] Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'01, Seattle/WA, USA*, pages 248–253, 2001.

[DDDL07] Sylvain Darras, Gilles Dequen, Laure Devendeville, and Chu Min Li. On inconsistent clause-subsets for Max-SAT solving. In *Proceedings of 13th International Conference on Principles and Practice of Constraint Programming, CP-2007, Providence, USA*, pages 225–240. Springer LNCS 4741, 2007.

[dGHZL05] Simon de Givry, Federico Heras, Matthias Zytnicki, and Javier Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted csps. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-2005, Edinburgh, Scotland*, pages 84–89. Morgan Kaufmann, 2005.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[ES] Niklas Eén and Niklas Sörensson. Minisat. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/.

[ES03] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing, SAT-2003, Santa Margherita Ligure, Italy*, pages 502–518. Springer LNCS 2919, 2003.

[eSSMS99] Luís Guerra e Silva, Luis Miguel Silveira, and João P. Marques-Silva. Algorithms for solving boolean satisfiability in combinational circuits. In *Proceedings of Design, Automation and Test in Europe, DATE'99, Munich, Germany*, pages 526–530, 1999.

[FM06] Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing, SAT-2006, Seattle, USA*, pages 252–265. Springer LNCS 4121, 2006.

[Fre95] Jon William Freeman. *Improvements to Propositional Satisfiability Search Algorithms.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1995.

[Gel02] A. Van Gelder. Generalizations of watched literals for backtracking search. In *Proceedings of the 7th International Symposium on Artificial Intelligence and Mathematics, Ft. Lauderdale, FL*, 2002.

© CSIC © del autor o autores / Todos los derechos reservados

[Gen02] Ian P. Gent. Arc consistency in SAT. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI), Lyon, France*, pages 121–125, 2002.

[GN01] Evgueni Goldberg and Yakov Novikov. BerkMin: A fast and robust SAT solver. In *Proceedings of Design, Automation and Test in Europe, DATE-2002, Paris, France*, pages 142–149. IEEE Computer Society, 2001.

[GS97] Carla P. Gomes and Bart Selman. Problem structure in the presence of perturbations. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97, Providence/RI, USA*, pages 221–226. AAAI Press, 1997.

[GvHL06] Carla P. Gomes, Willem Jan van Hoeve, and Lucian Leahu. The power of semidefinite programming relaxations for max-sat. In *Proceedings of the Conference on Integration of Artificial Intelligence and Operation Research, CPAIOR-2006, Cork, Ireland*, pages 104–118. Springer LNCS 3990, 2006.

[GW93] Ian Gent and Toby Walsh. Towards an understanding of hill-climbing procedures for sat. In *Proceedings of National Conference on Artificial Intelligence (AAAI-93)*, pages 28–33, 1993.

[GW94a] Michel X. Goemans and David P. Williamson. .879-approximation algorithms for max cut and max 2sat. In *Proceedings of the 26th ACM Symposium on the Theory of Computing*, pages 422–431, 1994.

[GW94b] Michel X. Goemans and David P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal of Discrete Mathematics*, 7(4):656–666, 1994.

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[Hås97] Johan Håstad. Some optimal inapproximability results. In *Proceedings of the 28th ACM Symposium on the Theory of Computing*, pages 1–10, 1997.

[HDvMvZ04] Marijn Heule, Mark Dufour, Hans van Maaren, and Joris van Zwieten. March_eq: Implementing efficiency and additional reasoning into a lookahead sat-solver. *Journal on Satisfiability, Boolean Modeling and Computation*, pages 25–30, 2004.

[HL06] Federico Heras and Javier Larrosa. New inference rules for efficient Max-SAT solving. In *Proceedings of the National Conference on*

Copia gratuita. Personal free copy http://libros.csic.es

Artificial Intelligence, AAAI-2006, Boston/MA, USA, pages 68–73, 2006.

[HLdGS07] Federico Heras, Javier Larrosa, Simon de Givry, and Thomas Schiex. 2006 and 2007 max-sat evaluations: Contributed instances. *submitted to JSAT, Special issue on SAT 2007 competitions and evaluations*, 2007.

[HLO07] Federico Heras, Javier Larrosa, and Albert Oliveras. Minimaxsat: A new weighted Max-SAT solver. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing, SAT-2007, Lisbon, Portugal*, pages 41–55. Springer LNCS 4501, 2007.

[Hoo99] Holger H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proceedings of the 16th National Conference on Artificial Intelligence, AAAI'99*, pages 661–666. AAAI Press, 1999.

[HS04] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search. Foundations and Applications.* Morgan Kaufmann, 2004.

[HSvdW06] V.E.P. Heinink, M.J. Seckington, and F.S.D. van der Werf. Experiments on random 2-SoftSAT. Technical report, Delft University of Technology, 2006.

[HTH02] Frank Hutter, Dave Tompkins, and Holger Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In *Proceedings of CP-02*, volume 2470 of *LNCS*, pages 233–248. Springer, 2002.

[HV95] J. N. Hooker and V. Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, 15:359–383, 1995.

[JKS95] Yuejun Jiang, Henry Kautz, and Bart Selman. Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT. In *Proceedings of the 1st International Workshop on Artificial Intelligence and Operations Research*, 1995.

[Joh74] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Comput. and Sys. Sci.*, 9:256–278, 1974.

[JW90] Robert G. Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[Kas90] Simon Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, 45(3):275–286, 1990.

© CSIC © del autor o autores / Todos los derechos reservados

[Kau06] Henry A. Kautz. Deconstructing planning as satisfiability. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAAI-2006, Boston/MA, USA*, 2006.

[Kri85] Balakrishnan Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22(3):253–275, 1985.

[KS96] Henry A. Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'96, Portland/OR, USA*, pages 1194–1201, 1996.

[KSHK07] Daher Kaiss, Marcelo Skaba, Ziyad Hanna, and Zurab Khasidashvili. Industrial strength sat-based alignability algorithm for hardware equivalence verification. In *Proceedings of the 7th International Conference on Formal Methods in Computer-Aided Design, FMCAD-2007, Austin/TX, USA*, pages 20–26. IEEE Computer Society, 2007.

[KSTW05] Philip Kilby, John K. Slaney, Sylvie Thibaux, and Toby Walsh. Backbones and backdoors in satisfiability. In *Proceedings of the Twentieth National Conference in Artificial Intelligence (AAAI-05)*, pages 1368–1373. AAAI Press, 2005.

[KZ97] Howard J. Karloff and Uri Zwick. A 7/8-approximation algorithm for max 3sat? In *Proceedings of the 38th Annual IEEE Symposium on Fundations of Computer Science, FOCS'97*, pages 406–415, 1997.

[LA97a] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'97, Nagoya, Japan*, pages 366–371, 1997.

[LA97b] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of the 3rd International Conference on Principles of Constraint Programming, CP'97, Linz, Austria*, pages 341–355. Springer LNCS 1330, 1997.

[LH05a] Javier Larrosa and Federico Heras. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-2005, Edinburgh, Scotland*, pages 193–198. Morgan Kaufmann, 2005.

[LH05b] Chu Min Li and Wen Qi Huang. Diversification and determinism in local search for satisfiability. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing, SAT-2005, St. Andrews, Scotland*, pages 158–172. Springer LNCS 3569, 2005.

Copia gratuita. Personal free copy http://libros.csic.es

[LHdG08] Javier Larrosa, Federico Heras, and Simon de Givry. A logical approach to efficient max-sat solving. *Artificial Intelligence*, 172(2–3):204–233, 2008.

[Li03] Chu Min Li. Equivalent literal propagation in the DLL procedure. *Discrete Applied Mathematics*, 130:251–276, 2003.

[LM99] Javier Larrosa and Pedro Meseguer. Partition-based lower bound for Max-CSP. In *5th International Conference on Principles and Practice of Constraint Programming, CP'99, Alexandria, USA*, pages 303–315. Springer LNCS 1713, 1999.

[LM09] Chu Min Li and F. Manyà. Max-sat, hard and soft constraints. In Armin Biere, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, pages 613–631. IOS Press, 2009.

[LMP05] Chu Min Li, Felip Manyà, and Jordi Planes. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, CP-2005, Sitges, Spain*, pages 403–414. Springer LNCS 3709, 2005.

[LMP06] Chu Min Li, Felip Manyà, and Jordi Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAAI-2006, Boston/MA, USA*, pages 86–91, 2006.

[LMP07] Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for max-sat. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.

[LMS99] Javier Larrosa, Pedro Meseguer, and Thomas Schiex. Maintaining reversible dac for max-csp. *Artificial Intelligence*, 107(1):149–163, 1999.

[LMS01] Inês Lynce and Joao P. Marques-Silva. Integrating simplification techniques in sat algorithms. In *IEEE Symposium on Logic in Computer Science*, 2001. Short paper session.

[LMS02] Inês Lynce and Joao P. Marques-Silva. Efficient data structures for backtrack search SAT solvers. In *Fifth International Symposium on the Theory and Applications of Satisfiability Testing, SAT-2002, Cincinnati, USA*, pages 308–315, 2002.

[LMS05] Inês Lynce and João P. Marques-Silva. Efficient data structures for backtrack search sat solvers. *Annals of Mathematics and Artificial Intelligence*, 43(1):137–152, 2005.

© CSIC © del autor o autores / Todos los derechos reservados

[LMS06a] Inês Lynce and João Marques-Silva. Efficient haplotype inference with boolean satisfiability. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAAI-2006, Boston/MA, USA*, 2006.

[LMS06b] Inês Lynce and João Marques-Silva. Sat in bioinformatics: Making the case with haplotype inference. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing, SAT-2006, Seattle, USA*, pages 136–141. Springer LNCS 4121, 2006.

[LS07] Han Lin and Kaile Su. Exploiting inference rules to compute lower bounds for max-sat solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI-2007, Hyderabad, India*, pages 2334–2339, 2007.

[MBB$^+$03] Pedro Meseguer, Noureddine Bouhmala, Taoufik Bouzoubaa, Morten Irgens, and Martí Sánchez. Current approaches for solving over-constrained problems. *Constraints*, 8(1):9–39, 2003.

[MIK96] Shuichi Miyazaki, Kazuo Iwama, and Yahiko Kambayashi. Database queries as combinatorial optimization problems. In *CODAS*, pages 477–483, 1996.

[Mit05] David Mitchell. A sat solver primer. *European Association for Theoretical Computer Science (EATCS) Bulletin*, 85:112–133, 2005.

[MMZ$^+$01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 39th Design Automation Conference, DAC'01*, pages 530–535, 2001.

[MRS06] Pedro Meseguer, Francesca Rossi, and Thomas Schiex. Soft constraints. In Francesca Rossi, Peter Van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, chapter 9. Elsevier, 2006.

[MS99] Joao P. Marques-Silva. The impact of branching heuristics in propositional satisfiability algorithms. In Pedro Barahona and José Júlio Alferes, editors, *Proceedings of the 9th Portuguese Conference on Artificial Intelligence: Progress in Artificial Intelligence (EPIA-99)*, volume 1695 of *LNCS*, pages 62–74, 1999.

[MSG97] Bertrand Mazure, Lakhdar Saïs, and Éric Grégoire. Tabu search for SAT. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97, Providence/RI, USA*, pages 281–285. AAAI Press, 1997.

Copia gratuita. Personal free copy http://libros.csic.es

[MSK97] David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97, Providence/RI, USA*, pages 321–326. AAAI Press, 1997.

[MSLM08] João P. Marques-Silva, Inês Lynce, and Sharad Malik. Cdcl solvers. In Armin Biere, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*. IOS Press, 2008.

[MSP08] João P. Marques-Silva and Jordi Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Proceedings of Design, Automation and Test in Europe, DATE-2008*, 2008.

[MSS99] João P. Marques-Silva and Karem A. Sakallah. Graps: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[Nad02] A. Nadel. *Backtrack Search Algorithms for Propositional Logic Satisfiability: Review and innovations*. PhD thesis, Hebrew University of Jerusalem, 2002.

[NR00] Rolf Niedermeier and Peter Rossmanith. New upper bounds for maximum satisfiability. *Journal of Algorithms*, 36:63–88, 2000.

[PD07] Knot Pipatsrisawat and Adnan Darwiche. Clone: Solving weighted max-sat in a reduced search space. In *Proceedings of the 20th Australian Conference on Artificial Intelligence, AI-2007, Gold Coast, Australia*, pages 223–233. Springer LNCS 4830, 2007.

[Pre93] Daniele Pretolani. Efficiency and stability of hypergraph SAT algorithms. In *Proceedings of the DIMACS Challenge II Workshop*, 1993.

[RG07] Miquel Ramírez and Hector Geffner. Structural relaxations by variable renaming and their compilation for solving mincostsat. In *Proceedings of 13th International Conference on Principles and Practice of Constraint Programming, CP-2007, Providence, USA*, pages 605–619. Springer LNCS 4741, 2007.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, 1965.

[Rya04] Lawrence Ryan. Efficient algorithms for clause learning SAT solvers. Master's thesis, Simon Fraser University, 2004.

[Sch89] Uwe Schöning. *Logic for Computer Scientists*, volume 8 of *Progress in Computer Science and Applied Logic*. Birkhäuser, 1989.

© CSIC © del autor o autores / Todos los derechos reservados

[SD96] Barbara M. Smith and Martin E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):155–181, 1996.

[SHR01] Thomas Stützle, Holger Hoos, and Andrea Roli. A review of the literature on local search algorithms for MAX-SAT. Technical report, AIDA-01-02, FG Intellektik, FB Informatik, TU Darmstadt, Germany, 2001.

[SHS03] Kevin Smyth, Holger H. Hoos, and Thomas Stützle. Iterated robust tabu search for max-sat. In *Proceedings of the 16th Conference of the Canadian Society for Computational Studies of Intelligence, AI-2003, Halifax, Canada*, pages 129–144. Springer LNCS 2671, 2003.

[SK93] Bart Selman and Henry A. Kautz. Domain-independent extensions of GSAT: Solving large structured satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'93, Chambery, France*, pages 290–295, 1993.

[SKC94] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Intelligence, AAAI'94, Seattle/WA, USA*, pages 337–343. AAAI Press, 1994.

[SLM92] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI'92, San Jose/CA, USA*, pages 440–446. AAAI Press, 1992.

[SZ04] Haiou Shen and Hantao Zhang. Study of lower bound functions for max-2-sat. In *Proceedings of the 19th National Conference on Artificial Intelligence, 16th Conference on Innovative Applications of Artificial Intelligence, San Jose, California, USA*, pages 185–190. AAAI Press / The MIT Press, 2004.

[TH05] Dave A. D. Tompkins and Holger H. Hoos. Ubcsat: An implementation and experimentation environment for sls algorithms for sat and max-sat. In *7th International Conference on Theory and Applications of Satisfiability Testing, SAT-2004, Vancouver, BC, Canada*, pages 306–320. Springer LNCS 3542, 2005.

[Urq87] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, 1987.

[VB01] Miroslav N. Velev and Randal E. Bryant. Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors. In *Proceedings of the 38th Design Automation Conference, DAC-2001, Las Vegas/NV, USA*, pages 226–231, 2001.

Copia gratuita. Personal free copy http://libros.csic.es

[WF96] Richard J. Wallace and Eugene Freuder. Comparative studies of constraint satisfaction and Davis-Putnam algorithms for maximum satisfiability problems. In D. Johnson and M. Trick, editors, *Cliques, Coloring and Satisfiability*, volume 26, pages 587–615. American Mathematical Society, 1996.

[WvM98] Joost P. Warners and Hans van Maaren. A two-phase algorithm for solving a class of hard satisfiability problems. *Operations Research Letters*, 23:81–88, 1998.

[XZ04] Zhao Xing and Weixiong Zhang. Efficient strategies for (weighted) maximum satisfiability. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming, CP-2004, Toronto, Canada*, pages 690–705. Springer LNCS 3258, 2004.

[XZ05] Zhao Xing and Weixiong Zhang. An efficient exact algorithm for (weighted) maximum satisfiability. *Artificial Intelligence*, 164(2):47–80, 2005.

[Yan94] Mihalis Yannakakis. On the approximation of maximum satisfiability. *Journal of Algorithms*, 17:475–502, 1994.

[Zha97] Hantao Zhang. SATO: An efficient propositional prover. In *Conference on Automated Deduction (CADE-97)*, pages 272–275, 1997.

[Zha03] Lintao Zhang. *Searching for truth: techniques for satisfiability of boolean formulas.* PhD thesis, Department of Electrical Engineering. Princeton University., June 2003.

[ZLS04] Hantao Zhang, Dapeng Li, and Haiou Shen. A sat based scheduler for tournament schedules. In *7th International Conference on Theory and Applications of Satisfiability Testing, SAT-2004, Vancouver, BC, Canada.* Springer LNCS 3542, 2004.

[ZM88] R. Zabih and D. A. McAllester. A rearrangement search strategy for determining propositional satisfiability. In *In Proceedings of the National Conference on Artificial Intelligence (AAAI-88)*, pages 155–160, 1988.

[ZM02] L. Zhang and S. Malik. The quest for efficient Boolean satisfiability solvers. In *18th International Conference on Automated Deduction, CADE-18, Copenhagen, Denmark*, pages 295–313. Springer, LNCS 2392, 2002.

[ZMMM01] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *International Conference on Computer Aided Design, ICCAD-2001, San Jose/CA, USA*, pages 279–285, 2001.

© CSIC © del autor o autores / Todos los derechos reservados

[ZS96] Hantao Zhang and Mark E. Stickel. An efficient algorithm for unit propagation. In *Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics, AI-MATH'96*, Fort Lauderdale (Florida USA), 1996.

[ZSM03] Hantao Zhang, Haiou Shen, and Felip Manya. Exact algorithms for MAX-SAT. *Electronic Notes in Theoretical Computer Science*, 86(1), 2003.

Copia gratuita. Personal free copy http://libros.csic.es

Monografies de l'Institut d'Investigació en Intel·ligència Artificial

Num. 1	J. Puyol, *MILORD II: A Language for Knowledge–Based Systems*, (1995).
Num. 2	J. Levy, *The Calculus of Refinements, a Formal Specification Model Based on Inclusions*, (1995).
Num. 3	Ll. Vila, *On Temporal Representation and Reasoning in Knowledge–Based Systems*, (1995).
Num. 4	M. Domingo, *An Expert System Architecture for Identification in Biology*, (1995).
Num. 5	E. Armengol, *A Framework for Integrating Learning and Problem Solving*, (1998).
Num. 6	J. Ll. Arcos, *The Noos Representation Language*, (1998).
Num. 7	J. Larrosa, *Algorithms and Heuristics for Total and Partial Constraint Satisfaction* , (1998).
Num. 8	P. Noriega, *Agent Mediated Auctions: The Fishmarket Metaphor*, (1999).
Num. 9	F. Manyà, *Proof Procedures for Multiple-Valued Propositional Logics*, (1999).
Num. 10	W. M. Schorlemmer, *On Specifying and Reasoning with Special Relations*, (1999).
Num. 11	M. López-Sánchez, *Approaches to Map Generation by means of Collaborative Autonomous Robots*, (2000).
Num. 12	D. Robertson, *Pragmatics in the Synthesis of Logic Programs*, (2000).
Num. 13	P. Faratin, *Automated Service Negotiation between Autonomous Computational Agents*, (2003).
Num. 14	J. A. Rodríguez, *On the Design and Construction of Agent-mediated Electronic Institutions*, (2003).
Num. 15	T. Alsinet, *Logic Programming with Fuzzy Unification and Imprecise Constants: Possibilistic Semantics and Automated Deduction*, (2003).
Num. 16	A. Zapico, *On Axiomatic Foundations for Qualitative Decision Theory A Posibilistic Approach*, (2003).
Num. 17	A. Valls, *ClusDM: A multiple criteria decision method for heterogeneous data sets*, (2003).
Num. 18	D. Busquets, *A Multiagent Approach to Qualitative Navigation in Robotics*, (2003).
Num. 19	M. Esteva, *Electronic Institutions: from specification to development*, (2003).
Num. 20	J. Sabater, *Trust and reputation for agent societies*, (2003).

Copia gratuita. Personal free copy   http://libros.csic.es

Num. 21	J. Cerquides, *Improving Algorithms for Learning Bayesian Network Classifiers*, (2005).
Num. 22	M. Villaret, *On Some Variants of Second-Order Unification*, (2005).
Num. 23	M. Gómez, *Open, Reusable and Configurable Multi-Agent Systems: A Knowledge Modelling Approach*, (2005).
Num. 24	S. Ramchurn, *Multi-Agent Negotiation Using Trust and Persuasion*, (2005).
Num. 25	S. Ontañón, *Ensemble Case-Based Learning for Multi-Agent Systems*, (2006).
Num. 26	M. Sánchez, *Contributions to Search and Inference Algorithms for CSP and Weighted CSP*, (2006).
Num. 27	C. Noguera, *Algebraic Study of Axiomatic Extensions of Triangular Norm Based Fuzzy Logics*, (2007).
Num. 28	E. Marchioni, *Functional Definability Issues in Logics Based on Triangular Norms*, (2007).
Num. 29	M. Grachten, *Expressivity-Aware Tempo Transformations of Music Performances Using Case Based Reasoning*, (2007).
Num. 30	I. Brito, *Distributed Constraint Satisfaction*, (2007).
Num. 31	E. Altamirano, *On Non-clausal Horn-like Satisfiability Problems*, (2007).
Num. 32	A. Giovannucci, *Computationally Manageable Combinatorial Auctions for Supply Chain Automation*, (2008).
Num. 33	R. Ros, *Action Selection in Cooperative Robot Soccer using Case-Based Reasoning*, (2008).
Num. 34	A. García Cerdaña, *On some Implication-free Fragments of Substructural and Fuzzy Logics*, (2008).
Num. 35	A. García Camino, *Normative Regulation of Open Multi-agent Systems*, (2008).
Num. 36	A. Robles, *Enabling Intelligent Organizations: An Electronic Institutions Approach for Building Agent Oriented Information Systems*, (2008).
Num. 37	I. Drummond, *Imprecise Classification Based on Fuzzy Logic and Possibility Theory*, (2008).
Num. 38	J. Planes, *Design and Implementation of Exact MAX-SAT Solvers*, (2008).
Num. 39	A. Bogdanovych, *Virtual Institutions*, (2008).
Num. 40	J. Nin, *Contributions to Record Linkage for Disclosure Risk Assessment*, (2009).
Num. 41	J. Argelich, *Max-SAT Formalisms with Hard and Soft Constraints*, (2009).

© CSIC © del autor o autores / Todos los derechos reservados