Solving Over-Constrained Problems with SAT Technology*

Josep Argelich[1] and Felip Manyà[2]

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

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

1 Introduction

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

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

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

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

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

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

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

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

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

2 Soft CNF Formulas

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5 Experimental Investigation

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

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

- Soft-SAT-S with heuristic MO and csp.
- Soft-SAT-D with heuristic 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 better than the lower bound of Soft-SAT because it is the sum of the number of unsatisfied clauses by the current partial assignment plus an underestimation of the number of clauses that will become unsatisfied if we extend the current partial assignment into a complete assignment, which is calculated taking into account the inconsistency counts of the variables not yet instantiated. WMax-SAT incorporates the following variable selection heuristic: It instantiates the variables taking into account the number of occurrences in decreasing order (MO).
- BF-improved: It is an improved version of Borchers and Furman's algorithm [3] described in [2]. It uses the popular dynamic variable selection heuristics MOMS (most often in minimum size clauses).
- PFC-MPRDAC: This is a highly optimized solver from the Constraint Programming community for solving binary Max-CSP problems [10].

The benchmarks we used in our experiments are:

- Random 2-SAT instances: We have generated random 2-SAT instances to which we have then assigned, randomly and uniformly, a label corresponding to a 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.
- Max-CSP instances: We used SAT-encoded random binary CSPs and solved the Max-CSP problem (the problem of finding an assignment to the variables that satisfies as many constraints as possible). We used Max-CSP instances because they have a natural representation using the formalism of soft CNF formulas. The instances were encoded using the support encoding[4] and generated with a generator of uniform random binary CSPs[5] —designed

[4] In the *support encoding* [9, 7], the idea 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 a hard block, and each set of clauses that encodes a CSP constraint was encoded as a different soft block.

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

and implemented by Frost, Bessière, Dechter and Regin— that implements the so-called model B [16]: 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.

– 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 in CP-2002, and randomly generated instances using the generator of Culberson [5]. We use 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).

All the benchmarks encoded as Soft CNF formulas were also encoded as Boolean weighted Max-SAT instances in order to compare our solvers with Boolean weighted Max-SAT solvers. The encoding is 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. Moreover, we associate a weight $w+1$, where w is the sum of weights of the clauses that encode soft blocks, with each clause belonging to a hard block. Any truth assignment where the sum of unsatisfied clauses is less than $w+1$ is a feasible solution. A solution of a soft CNF formula corresponds to a feasible solution of its weighted 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 unsatisfied by a truth assignment that satisfies all the hard blocks.

The Max-CSP instances and the graph coloring instances were also encoded as binary CSP using the format used by PFC-MPRDAC [10], 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 for a long time on this topic and, in contrast to the SAT community, it is a very active research subject.

5.1 Experiments with Random 2-SAT Instances

We compared Soft-SAT-S, Soft-SAT-D and WMax-SAT on random 2-SAT instances using heuristic MO.[6] Figure 1 shows the results for instances with 50 variables and with a number of clauses ranging from 200 to 430, and Figure 2 shows the results for instances with a number of variables ranging from 50 to 100

[6] We tried also to solve the instances with BF-improved, but the results were not competitive.

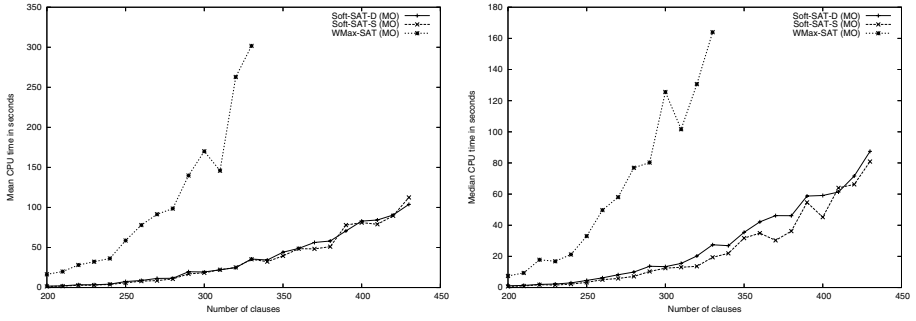

Fig. 1. Random 2-SAT instances with 50 variables and with a number of clauses ranging from 200 to 430

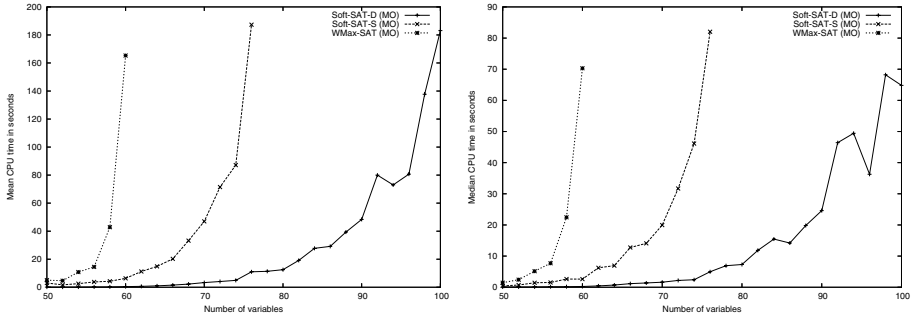

Fig. 2. Random 2-SAT instances with a number of variables ranging from 50 to 100 and with 300 clauses

and with 300 clauses. In both figures we give mean and median time, and each data point corresponds to the time needed to solve a set of 100 instances. In Figure 1 we observe that, in general, Soft-SAT-D is slightly better that Soft-SAT-S, and in Figure 2 we observe that Soft-SAT-D is superior than Soft-SAT-S. WMax-SAT is much worse: Even the fact of having a better lower bound in WMax-SAT does not compensate the extra propagation achieved by exploiting the fact that Soft-SAT knows which clauses encode hard constraints.

5.2 Experiments with Max-CSP Instances

In this section we describe a number of experiments we performed on random binary CSP. For instances with CSP variables with domain size greater than 2, we defined a lower bound 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

Table 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
$\langle 10, 15, 45/45, 190/225 \rangle$	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

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.

In Table 1 we compare Soft-SAT-S without underestimation 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 heuristic used is csp. Table 2 shows the number of backtracks instead of the CPU time for the same instances. In both cases we

Table 2. Comparison of Soft-SAT-S without underestimation and Soft-SAT-S with underestimation on Max-CSP instances. Mean and median number of backtracks

	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 3. Comparison of Soft-SAT-S, PFC-MPRDAC and WMax-SAT on Max-CSP instances. Time in seconds

	Soft-SAT-S		PFC-MPRDAC		WMax-SAT	
$\langle n, d, p_1, p_2 \rangle$	mean	median	mean	median	mean	median
$\langle 10, 8, 45/45, 48/64 \rangle$	0.33	0.32	0.19	0.19	11.95	11.41
$\langle 12, 6, 66/66, 27/36 \rangle$	0.48	0.47	0.23	0.23	45.50	45.48
$\langle 14, 5, 91/91, 18/25 \rangle$	0.82	0.77	0.35	0.35	188.75	193.30
$\langle 16, 4, 120/120, 12/16 \rangle$	0.37	0.32	0.22	0.22	275.45	276.25
$\langle 18, 3, 153/153, 6/9 \rangle$	1.00	0.93	0.31	0.31	68.04	62.71
$\langle 15, 6, 60/105, 27/36 \rangle$	0.25	0.24	0.20	0.20	777.94	538.63
$\langle 18, 5, 80/153, 18/25 \rangle$	0.67	0.58	0.33	0.30	5382.68	3364.17
$\langle 20, 5, 70/190, 18/25 \rangle$	0.54	0.44	0.33	0.31	4701.25	2714.86

observe that the fact of adding a lower bound of better quality leads to dramatic performance improvements. In the rest of the paper, all the results reported take into account the above described underestimation.

In the second experiment we compared Soft-SAT-S with heuristic csp,[7] PFC-MPRDAC and WMax-SAT on Max-CSP instances. The results obtained are shown in Table 3. We observe that solver PFC-MPRDAC (which is specialized on solving Max-CSP instances) is about 2 times faster than Soft-SAT-S, but the Weighted Max-SAT approach is much worse. We do not display results with BF-improved because they are worse than the results of WMax-SAT. Even when our solver is not the best, it is quite competitive.

In the third experiment, whose results are shown in Table 4, we solve 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.

5.3 Experiments with Graph Coloring Instances

The last benchmark we used was graph coloring. In the first experiment we considered 7 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, Soft-SAT-D with heuristic MO, and PFC-MPRDAC.[8] The results obtained are shown in Table 5: the first column displays the parameters

[7] We used this solver of soft CNF formulas because is the best performing one for Max-CSP instances.

[8] We do not give results with Weighted Max-SAT because is not competitive with the solvers used.

Table 4. Comparison of Soft-SAT-D with heuristic MO-csp and Soft-SAT-D with heuristic MO on Max-CSP instances. Time in seconds

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

	Soft-SAT-S		Soft-SAT-D-MO-csp		Soft-SAT-D-MO		PFC-MPRDAC	
$\langle n, k, c \rangle$	mean	median	mean	median	mean	median	mean	median
$\langle 15, 15, 8 \rangle$	**103.62**	**10.47**	318.75	26.73	743.64	20.12	132.73	21.29
$\langle 15, 15, 10 \rangle$	**102.69**	**0.05**	261.64	0.06	653.74	0.06	139.89	0.15
$\langle 16, 14, 6 \rangle$	**197.13**	**49.00**	986.61	224.60	1350.28	342.58	234.14	78.63
$\langle 16, 14, 8 \rangle$	**164.81**	**19.38**	391.85	29.45	611.52	42.81	207.56	26.26
$\langle 16, 16, 6 \rangle$	**208.48**	**129.61**	950.25	545.06	1503.82	1089.34	250.16	180.93
$\langle 16, 16, 8 \rangle$	**91.87**	**23.33**	224.92	37.11	287.99	46.66	147.50	37.01
$\langle 18, 10, 5 \rangle$	**72.17**	**32.84**	314.09	144.96	435.76	212.44	73.74	42.64

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 solvers used.

We repeated the previous experiments but using a representative sample of individual instances from the graph coloring symposium celebrated in CP-2002. The results obtained are shown in Table 6: 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 solvers used.

We observe that, in both cases, our approach is superior to the constraint programming approach. For all the sets of randomly generated instances, Soft-SAT-S outperforms PFC-MPRDAC, while for the individual instances some times is better Soft-SAT-S and sometimes Soft-SAT-D. We also observe in both experiments that the n-ary branchings analyzed lead to better performance profiles than the binary branching.

We have also solved some graph coloring instances with the pseudo-Boolean solver PBS v2.1 [1]. Our preliminary results indicate that, at least for the in-

Table 6. Comparison between Soft-SAT-S with heuristic csp, Soft-SAT-D with heuristic MO-csp, Soft-SAT-D with heuristic MO, and PFC-MPRDAC on individual graph coloring instances. Time in seconds

$\langle$Instance, $k, c\rangle$	vc	Soft-SAT-S	Soft-SAT-D-MO-csp	Soft-SAT-D-MO	PFC-MPRDAC
$\langle$myciel5.col, 6, 3$\rangle$	16	**11.04**	46.39	50.47	12.11
$\langle$myciel5.col, 6, 4$\rangle$	4	**78.50**	226.59	344.79	96.41
$\langle$myciel5.col, 6, 5$\rangle$	1	3177.85	**31.87**	73.73	44.34
$\langle$GEOM30a.col, 6, 3$\rangle$	11	**9.31**	27.22	36.11	14.33
$\langle$GEOM30a.col, 6, 4$\rangle$	4	4.48	**2.35**	7.29	22.89
$\langle$GEOM30a.col, 6, 5$\rangle$	1	0.49	**0.15**	0.22	0.18
$\langle$GEOM40.col, 6, 2$\rangle$	22	**3.89**	20.58	21.01	4.42
$\langle$GEOM40.col, 6, 3$\rangle$	7	**10.83**	30.63	65.97	770.46
$\langle$GEOM40.col, 6, 4$\rangle$	3	95.18	**14.67**	69.55	>7200.00
$\langle$GEOM40.col, 6, 5$\rangle$	1	1.58	**0.51**	1.89	1574.44
$\langle$queen5_5.col, 5, 3$\rangle$	29	57.60	167.60	205.37	**27.27**
$\langle$queen5_5.col, 5, 4$\rangle$	12	**37.50**	124.24	148.27	73.67

stances tested, our Soft-SAT solvers outperform PBS. We plan to perform a comprehensive comparison with more pseudo-Boolean solvers in an extended version of this paper.

6 Concluding Remarks

We have presented a new generic problem solving approach for over-constrained problems based on Max-SAT algorithms 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 way, and to traverse the search space of all possible truth assignments in a more efficient way; the extra level of propagation achieved in our solvers is a key factor of the good performance profiles obtained. Our experiments indicate that our approach is better than reducing over-constrained problems to weighted Max-SAT problems. Interestingly, for graph coloring instances, we have shown that the problem solving approach based on Soft CNF formulas also outperforms the approach based on Max-CSP instances. Taking into account the amount of efforts devoted in the constraint programming community on investigating methods of solving over-constrained problems, we believe that the results of this paper open an interesting research avenue.

An important point of our experimental investigation is the good results we obtained due to the lower bound we have implemented, as well as to the fact of using n-ary branching instead of binary branching. It is worth to consider these two points when designing weighted Max-SAT solvers that have to solve more realistic instances. One problem of state-of-the-art Max-SAT solvers is that they are biased to solve randomly generated 2-SAT and 3-SAT instances. They are rarely evaluated with more structured instances and with instances that encode

CSP variables with domain size greater than 2. The results reported here can provide some hints to improve Max-SAT solvers on more realistic instances.

Another important point of our experimental investigation is the good results we obtained with Soft-SAT-S. The extremely efficient data structures that we have implemented are the key 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.

As future work, 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", and to incorporate more advanced variable selection heuristics.

It is worth mentioning that we have 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 have found refer to local search algorithms, and do not incorporate the notion of block of clauses.

Finally, we would like to point out that we believe that it is worth exploring how the SAT technology developed for decision problems can be applied to solve optimization problems. This paper has tried to make a step forward in that direction.

References

1. F. Aloul, A. Ramani, I. Markov, and K. Sakallah. PBS: A backtrack search pseudo-Boolean solver. In *Symposium on the Theory and Applications of Satisfiability Testing, SAT-2002*, 2002.
2. T. Alsinet, F. Manyà, and J. 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.
3. B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2:299–306, 1999.
4. B. Cha, K. Iwama, Y. Kambayashi, and S. 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.
5. J. Culberson. Graph coloring page: The flat graph generator. See http://web.cs.ualberta.ca/˜ joe/Coloring/Generators/flat.html, 1995.
6. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.
7. I. P. Gent. Arc consistency in SAT. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI), Lyon, France*, pages 121–125, 2002.
8. Y. Jiang, H. Kautz, and B. 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.
9. S. Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, 45:275–286, 1990.
10. J. Larrosa. *Algorithms and Heuristics for Total and Partial Constraint Satisfaction*. PhD thesis, FIB, Universitat Politècnica de Catalunya, Barcelona, 1998.

11. D. W. Loveland. *Automated Theorem Proving. A Logical Basis*, volume 6 of *Fundamental Studies in Computer Science*. North-Holland, 1978.
12. P. Meseguer, N. Bouhmala, T. Bouzoubaa, M. Irgens, and M. Sánchez. Current approaches for solving over-constrained problems. *Constraints*, 8(1):9–39, 2003.
13. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *39th Design Automation Conference*, 2001.
14. B. Selman, H. Levesque, and D. 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.
15. H. Shen and H. Zhang. Study of lower bound functions for max-2-sat. In *Proceedings of AAAI-2004*, pages 185–190, 2004.
16. B. Smith and M. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:155–181, 1996.
17. R. Wallace and E. 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. 1996.
18. Z. Xing and W. Zhang. Efficient strategies for (weighted) maximum satisfiability. In *Proceedings of CP-2004*, pages 690–705, 2004.