J Heuristics (2006) 12: 375–392
DOI 10.1007/s10732-006-7234-9

Exact Max-SAT solvers for over-constrained problems

Josep Argelich · Felip Manyà

© Springer Science + Business Media, LLC 2006

Abstract We present a new generic problem solving approach for over-constrained problems based on Max-SAT. We first define a Boolean clausal form formalism, called *soft CNF formulas*, 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, 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 2-SAT, Max-CSP, graph coloring, pigeon hole and quasigroup completion) which provides experimental evidence that our approach is very competitive compared with the state-of-the-art approaches developed in the CSP and SAT communities.

Keywords Soft constraints · Max-SAT · Solvers

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

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*.

J. Argelich
Computer Science Department, Universitat de Lleida, Jaume II, 69, E-25001 Lleida, Spain
e-mail: jargelich@diei.udl.es

F. Manyà (✉)
Artificial Intelligence Research Institute, IIIA, CSIC, Campus UAB, 08193 Bellaterra, Spain
e-mail: felip@iiia.csic.es

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 consider that all the constraints are *crisp*; i.e., they can only be either completely satisfied or completely violated. We do not consider *fuzzy* constraints; i.e., constraints that allow intermediate degrees of satisfaction. Nevertheless, crisp constraints are classified either as *hard* (i.e., they must be satisfied by any solution) or as *soft* (i.e., they 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. We invite the reader to consult (Meseguer et al., 2003) 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 Boolean 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, 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 2-SAT, Max-CSP, graph coloring, pigeon hole and quasigroup completion) which provides experimental evidence that our approach is very competitive compared with the state-of-the-art approaches developed in the CSP and SAT communities.

Problem solving of over-constrained problems with Max-SAT local search algorithms has been investigated before in Jiang et al. (1995); Cha et al. (1997). 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 3.2 we describe a solver for soft CNF formulas with static variable selection heuristics. In Section 3.3 we describe a solver for soft CNF formulas with dynamic variable selection heuristics. In Section 4 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 block 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 about constraint violations in a more natural way. On the other hand, it allows us to get more propagation at certain nodes (this point is discussed in the next section), as well as to define variable selection heuristics that exploit the fact that a variable occurs in a hard or in a soft block.

3. Soft-SAT solvers

In this section 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.

3.1. A basic Soft-SAT solver

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. A basic Soft-SAT solver 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 (Loveland, 1978) to the literals that occur in unit clauses of hard blocks; i.e., given a literal $\neg p$ (p), it deletes all the clauses containing the literal $\neg p$ (p) and removes all the occurrences of the literal p ($\neg p$). 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. 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 the creation of two branches from the current branch: the left branch corresponds to instantiating p to false, and the right branch corresponds to instantiating 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 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.

3.2. Soft-SAT-S: A solver with static variable selection heuristics

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, like in Alsinet et al. (2003); Wallace and Freuder (1996), the initial upper bound is obtained with a GSAT-like (Selman et al., 1992) 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 a hard block. Local minima are avoided by occasionally performing a random walk.

In branch and bound Max-SAT solvers, 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, which is calculated taking into account the inconsistency counts of the variables not yet instantiated. Since we are dealing with blocks of clauses, the lower bound

computation method of Max-SAT solvers like (Alsinet et al., 2003; Wallace and Freuder, 1996) cannot be applied to Soft CNF formulas because some inconsistent soft blocks could be counted more than once.

For instances with CSP variables with domain size greater than two, we defined a 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.

Inference. When branching is done, algorithms for Max-SAT like (Alsinet et al., 2003; Borchers and Furman, 1999; Wallace and Freuder, 1996; Xing and Zhang, 2005) 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 (Davis et al., 1962) 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 in the description of the basic Soft-SAT solver, 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 for Boolean CNF formulas, and is a key feature of our approach.

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 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 have also 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. In contrast to the lazy data structures used in Chaff (Moskewicz et al., 2001), where a dynamic variable selection heuristic is used, we do not have to deal with 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:** We instantiate first the variables that appear Most Often (MO). Ties are broken using the lexicographical order.
- **MOH:** We instantiate first the variables that appear most often, but we take into account if the variable occurs in a 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 hard blocks. Ties are broken using the lexicographical order.
  We give a bigger score to variables in hard blocks because Soft-SAT-S applies the one-literal rule to unit clauses of hard blocks.
- **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 a 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, 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 n-ary branchings.

3.3. Soft-SAT-D: A solver with dynamic variable selection heuristics

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 (Moskewicz et al., 2001). 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) in the remaining clauses. 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 it is difficult to know the current size of a clause 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.

4. Experimental investigation

We next report the experimental investigation we conducted to evaluate the performance of our problem solving approach. We used a representative sample of instances (random 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 were performed on a 2GHz Pentium IV with 512 Mb of RAM under Linux.

4.1. 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).
- BF-improved: It is an improved version of the Borchers and Furman's (BF) algorithm (Borchers and Furman, 1999) for weighted Max-SAT described in Alsinet et al. (2003). It uses the popular dynamic variable selection heuristics MOMS (most often in minimum size clauses), and a lower bound of better quality than the original lower bound of BF.
- Toolbar (de Givry et al., 2003; Larrosa and Schiex, 2003): It is a weighted Max-SAT solver that uses Max-CSP techniques and encodings to solve Max-SAT problems. We used version 2.0.
- PFC-MPRDAC (Larrosa and Meseguer, 1999): This is a highly optimized solver from the Constraint Programming community for solving binary Max-CSP problems.
- Toolbar-CSP (de Givry et al., 2003; Larrosa and Schiex, 2003): Toolbar version 2.0 has an option in which it works as a weighted Max-CSP solver. It incorporates the best performing solution techniques for solving Max-CSP developed by the Constraint Programming community. 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.

We have not considered pseudo-Boolean and integer programming solvers. Nevertheless, it is shown in de Givry et al. (2003) that these solvers does not outperform Toolbar. It is also interesting to notice that we have considered most of the exact weighted Max-SAT solvers which are publicly available. In recent years, a considerable number of Max-SAT solvers have been developed, but they do not allow to associate weights with clauses (see, for example, (Alber et al., 1998; Li et al., 2005; Zhang et al., 2003)).

4.2. Benchmarks and encodings

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

Random 2-SAT instances. We generated random 2-SAT instances to which we 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. A constraint satisfaction problem (CSP) consists of a set of variables, each with a domain of values, and a set of constraints. Each constraint is defined over some subset of the original set of variables, and limits the combination of values that the variables in this subset can take. A binary CSP has only binary constraints. Given a binary CSP P, a solution of the binary Max-CSP problem for P is an assignment to the variables of P 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 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 (Smith and Dyer, 1996): 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 (Kasif, 1990; Gent, 2002). 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 a hard block, and each set of clauses that encodes a CSP constraint was encoded as a different soft block.

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, and randomly generated instances using the generator of Culberson (Culberson, 1995). 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 a 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.

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

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

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 a hard block of the 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.

Quasigroup completion instances. We considered unsatisfiable instances of the quasigroup (or Latin square) completion problem (QCP) that were generated as indicated in Gomes and Selman (1997). 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 colors preassigned form a 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 $2n$.

4.3. Weighted Max-SAT and Max-CSP encodings

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 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. 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 that can be unsatisfied by a truth assignment that satisfies all the hard blocks.

Example 2. Given the soft CNF formula of Example 1, we derive the following weighted Max-SAT instance:

$$
\begin{aligned}
&(v_1^1 \vee v_1^2; 16), (\neg v_1^1 \vee \neg v_1^2; 16), (v_2^1 \vee v_2^2; 16),\\
&(\neg v_2^1 \vee \neg v_2^2; 16), (v_3^1 \vee v_3^2; 16), (\neg v_3^1 \vee \neg v_3^2; 16),\\
&(s_1; 1), (\neg v_1^1 \vee \neg v_2^1 \vee \neg s_1; 2), (\neg v_1^2 \vee \neg v_2^2 \vee \neg s_1; 2),\\
&(s_2; 1), (\neg v_1^1 \vee \neg v_3^1 \vee \neg s_2; 2), (\neg v_1^2 \vee \neg v_3^2 \vee \neg s_2; 2),\\
&(s_3; 1), (\neg v_2^1 \vee \neg v_3^1 \vee \neg s_3; 2), (\neg v_2^2 \vee \neg v_3^2 \vee \neg s_3; 2).
\end{aligned}
$$

The Max-CSP instances and the graph coloring instances were also encoded as binary CSP 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

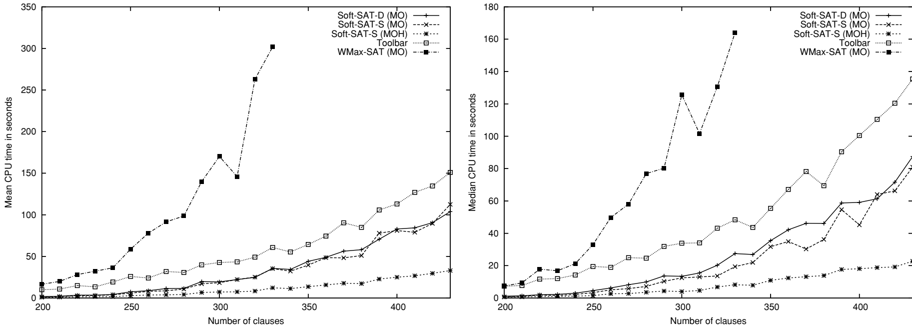

Fig. 1 Random 2-SAT instances with 50 variables, with a number of clauses ranging from 200 to 430, where 20 clauses are in the hard block and the rest of clauses are randomly distributed among 100 soft blocks. Mean time (left plot) and median time (right plot) in seconds

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.

4.4. Experiments with random 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 2-SAT instances.[2]

Figure 1 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 2 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 3 shows the results for instances with 60 variables and 300 clauses, where the number of clauses in hard blocks ranges from 10 to 50 and the rest of clauses are randomly distributed among 50 soft blocks. In all the 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 and Figure 3, we observe that the best performing solver is Soft-SAT-S with heuristic MOH, while in Figure 2 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 hard blocks and in soft blocks.

4.5. Experiments with Max-CSP instances

In this section we describe a number of experiments we performed on random binary CSP.

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 variable selection heuristic used is csp. Table 2 shows the number of backtracks instead of the CPU time for the same

[2] We tried also to solve the instances with BF-improved, but the results were not competitive. In Figure 3 we do not display the results for WMax-SAT because they were not competitive too.

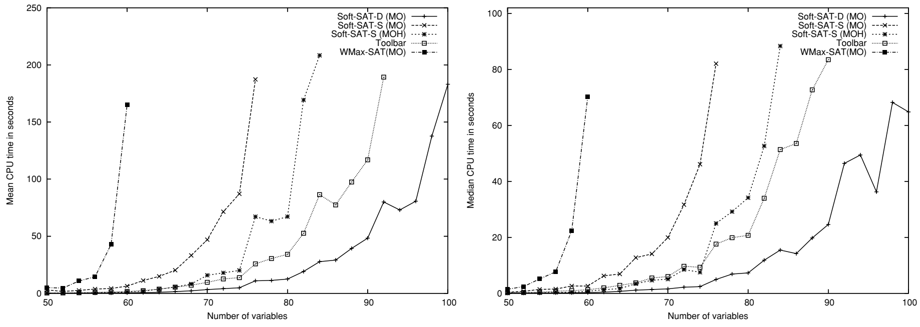

Fig. 2 Random 2-SAT instances with a number of variables ranging from 50 to 100 and with 300 clauses, where 50 clauses are in the hard block and the rest of clauses are randomly distributed among 50 soft blocks. Mean time (left plot) and median time (right plot) in seconds

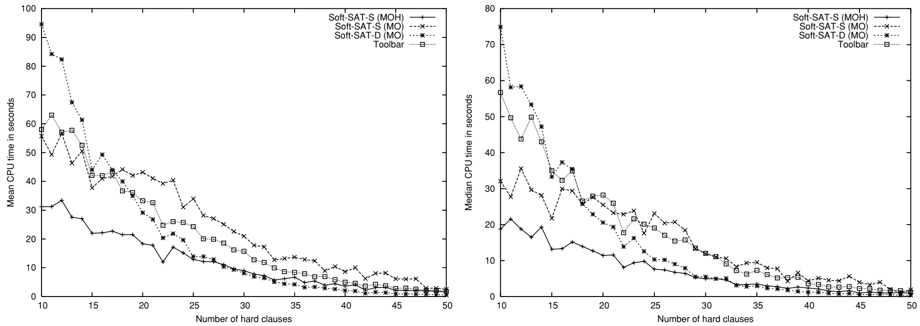

Fig. 3 Random 2-SAT instances with 60 variables and 300 clauses, where the number of clauses in hard blocks ranges from 10 to 50 and the rest of clauses are randomly distributed among 50 soft blocks. Mean time (left plot) and median time (right plot) in seconds

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 paper, all the results reported take into account the above described underestimation.

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 one-literal rule to unit clauses that appear in hard blocks. We generated sets 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 shows the experimental results obtained; we give mean time (left plot) and median time (right plot). We see that applying this inference technique that exploits the fact of knowing whether a variable belongs to a 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 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 Max-SAT approach is much worse. We do not display results with BF-improved and Toolbar because they are worse than the results of

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
⟨10, 15, 45/45, 190/225⟩	**12.25**	**10.31**	605.03	547.52
⟨12, 13, 60/66, 130/169⟩	**17.94**	**16.21**	2256.91	2010.26
⟨13, 8, 78/78, 50/64⟩	**12.51**	**11.28**	1028.91	973.41
⟨15, 10, 50/105, 75/100⟩	**1.63**	**1.39**	77.54	57.97
⟨17, 5, 110/136, 18/25⟩	**3.35**	**2.83**	394.82	343.61
⟨18, 5, 80/153, 18/25⟩	**0.86**	**0.76**	53.64	46.52
⟨20, 5, 90/190, 18/25⟩	**3.06**	**2.42**	406.53	378.00
⟨22, 6, 70/231, 28/36⟩	**7.10**	**4.13**	910.97	493.15
⟨23, 4, 150/253, 12/16⟩	**16.25**	**13.59**	4615.67	3797.04
⟨25, 3, 160/300, 7/9⟩	**2.66**	**2.09**	142.80	112.51

Table 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

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

WMax-SAT. Even when our solver is not the best, the differences with Weighted Max-SAT are substantial.

In the fourth experiment, whose results are shown in Table 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 contrast to the recent theoretical work by Hwang and Mitchel (Hwang and Mitchell, 2005), where they identify a family of CSP instances that have search trees

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

	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

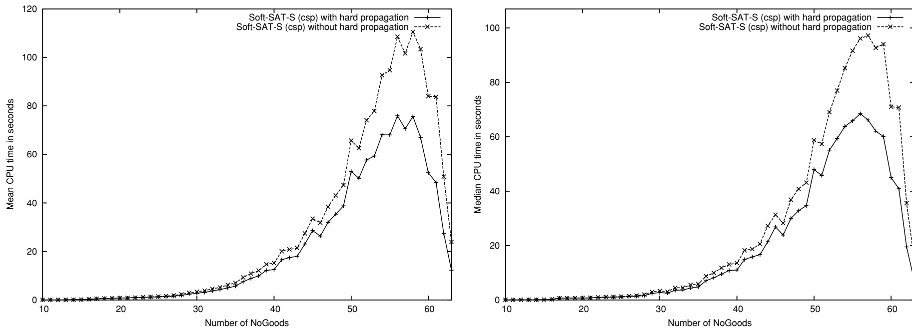

Fig. 4 Comparison of Soft-SAT-S with a version of Soft-SAT-S in which the one-literal rule is not applied to unit clauses that appear in hard blocks. Mean time (left plot) and median time (right plot) in seconds

of exponential size for an n-ary branching and of polynomial size for a binary branching, we provide evidence that random binary CSP's are solved faster with a solver with an n-ary branching. On the one hand, the instances considered are different. On the other hand, SAT resolution cannot be applied in Max-SAT.

4.6. Experiments with graph coloring instances

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

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
⟨10, 8, 45/45, 48/64⟩	**0.69**	**0.68**	1.83	1.76
⟨12, 6, 66/66, 27/36⟩	**1.20**	**1.11**	2.92	2.66
⟨14, 5, 91/91, 18/25⟩	**2.55**	**2.33**	6.82	6.27
⟨16, 4, 120/120, 12/16⟩	**2.69**	**2.54**	5.44	5.11
⟨18, 3, 153/153, 6/9⟩	**0.69**	**0.65**	1.40	1.28
⟨15, 6, 60/105, 27/36⟩	**1.16**	**1.01**	2.21	1.92
⟨18, 5, 80/153, 18/25⟩	**2.97**	**2.48**	5.98	4.38
⟨20, 5, 70/190, 18/25⟩	**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, Toolbar-CSP and PFC-MPRDAC on randomly generated graph coloring instances. Time in seconds

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

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 5: 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 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 in both experiments that Soft-SAT is very competitive with respect to Toolbar-CSP and superior to PFC-MPRDAC.

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

Table 6 Comparison between Soft-SAT-S, Soft-SAT-D, Toolbar-CSP and PFC-MPRDAC on individual graph coloring instances. Time in seconds

⟨Instance, k, c⟩	vc	Soft-SAT-S	Soft-SAT-D	Toolbar-CSP	PFC-MPRDAC
⟨myciel5.col, 6, 3⟩	16	11.04	46.39	**0.66**	12.11
⟨myciel5.col, 6, 4⟩	4	78.50	226.59	**6.28**	96.41
⟨myciel5.col, 6, 5⟩	1	3178	31.87	**26.02**	44.34
⟨GEOM30a.col, 6, 3⟩	11	9.31	27.22	**0.87**	14.33
⟨GEOM30a.col, 6, 4⟩	4	4.48	**2.35**	2.42	22.89
⟨GEOM30a.col, 6, 5⟩	1	0.49	**0.15**	0.17	0.18
⟨GEOM40.col, 6, 2⟩	22	3.89	20.58	**0.08**	4.42
⟨GEOM40.col, 6, 3⟩	7	**10.83**	30.63	25.20	770
⟨GEOM40.col, 6, 4⟩	3	95.18	**14.67**	1981	>7200.00
⟨GEOM40.col, 6, 5⟩	1	1.58	**0.51**	1186	1574
⟨queen5_5.col, 5, 3⟩	29	57.60	168	**9.22**	27.27
⟨queen5_5.col, 5, 4⟩	12	37.50	124	**13.18**	73.67

Table 7 Comparison between Soft-SAT-S with heuristic csp, Soft-SAT-D with heuristic MO-csp, Toolbar, WMax-SAT and BF-improved on pigeon hole instances. Time in seconds

N	Soft-SAT-S	Soft-SAT-D	Toolbar	WMax-SAT	BF-improved
7	**0.07**	0.10	0.43	0.08	0.11
8	**0.19**	0.28	4.05	0.54	0.76
9	**1.13**	1.55	43	5.32	7.49
10	**11**	12	521	75	86
11	133	**103**	6741	705	1068
12	1784	**990**	>7200	>7200	>7200

4.7. 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) and weighted Max-SAT solvers (Toolbar, WMax-SAT and BF-improved). The results obtained are shown in Table 7. 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 Soft-SAT solvers.

4.8. 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 8. We observe that the best performing solver is Soft-SAT-D and then Soft-SAT-S. The weighted Max-SAT solvers scale worse than the Soft-SAT solvers.

Table 8 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

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

5. Concluding remarks

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 way, and to design Max-SAT-like solvers that traverse the search space of all possible truth assignments in a more efficient way. In particular, we have provided experimental evidence that exploiting the fact of knowing whether variables and clauses appear in hard blocks or 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 a hard block or a soft block.
– Inference methods: we get an extra level of propagation by applying the one-literal rule to unit clauses that appear in hard blocks.

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

We have shown that our approach is much better than reducing over-constrained problems to weighted Max-SAT. On the one hand, Soft-SAT solvers have more powerful inference techniques and variable selection heuristics than weighted Max-SAT solvers. On the other hand, Soft-SAT solvers have novel data structures, branchings and lower bounds that could be incorporated into existing weighted Max-SAT solvers. 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 two. We believe that the techniques introduced here can contribute to develop weighted Max-SAT solvers with a better performance profile on more realistic instances.

We have shown 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 the results of this paper open an interesting research avenue in the SAT community.

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 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.

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.

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", 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.

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

Alber, J., J. Gramm, and R. Niedermeier. (1998). "Faster Exact Algorithms for Hard Problems: A Parameterized Point of View." In *25th Conf. on Current Trends in Theory and Practice of Informatics*, LNCS, Springer-Verlag, pp. 168–185.

Alsinet, T., F. Manyà, and J. Planes. (2003). "Improved Branch and Bound Algorithms for Max-SAT." In *Proceedings of the 6th International Conference on the Theory and Applications of Satisfiability Testing*.

Borchers, B. and J. Furman. (1999). "A Two-Phase Exact Algorithm for MAX-SAT and Weighted MAX-SAT Problems." *Journal of Combinatorial Optimization* 2, 299–306.

Cha, B., K. Iwama, Y. Kambayashi, and S. Miyazaki. (1997). "Local Search Algorithms for Partial MAXSAT." In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97, Providence/RI, USA*, AAAI Press, pp. 263–268.

Culberson, J. (1995). "Graph Coloring Page: The Flat Graph Generator." See http://web.cs.ualberta.ca/˜joe/Coloring/Generators/flat.html.

Davis, M., G. Logemann, and D. Loveland. (1962). "A Machine Program for Theorem-Proving." *Communications of the ACM* 5, 394–397.

de Givry, S., J. Larrosa, P. Meseguer, and T. Schiex. (2003). "Solving Max-SAT as Weighted CSP." In *9th International Conference on Principles and Practice of Constraint Programming, CP-2003, Kinsale, Ireland*, Springer LNCS 2833, pp. 363–376.

Gent, I. P. (2002). "Arc Consistency in SAT." In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI), Lyon, France*, pp. 121–125.

Gomes, C. P. and B. Selman. (1997). "Problem Structure in the Presence of Perturbations." In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97, Providence/RI, USA*, AAAI Press, pp. 221–226.

Hwang, J. and D. G. Mitchell. (2005). "2-way vs. d-way Branching for CSP." In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, CP-2005, Sitges, Spain*, Springer LNCS 3709, pp. 343–357.

Jiang, Y., H. Kautz, and B. Selman. (1995). "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.*

Kasif, S. (1990). "On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks." *Artificial Intelligence* 45, 275–286.

Larrosa, J. and P. Meseguer. (1999). "Partition-based Lower Bound for Max-CSP." In *5th International Conference on Principles and Practice of Constraint Programming, CP'99, Alexandria, USA*, Springer LNCS 1713, pp. 303–315.

Larrosa, J. and T. Schiex. (2003). "In the Quest of the Best form of Local Consistency for Weighted CSP." In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'03, Acapulco, México*, pp. 239–244.

Li, C. M., F. Manyà, and J. Planes. (2005). "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*, Springer LNCS 3709, pp. 403–414.

Loveland, D. W. (1978). *Automated Theorem Proving. A Logical Basis*, volume 6 of *Fundamental Studies in Computer Science*. North-Holland.

Meseguer, P., N. Bouhmala, T. Bouzoubaa, M. Irgens, and M. Sánchez. (2003). "Current Approaches for Solving Over-Constrained Problems." *Constraints* 8(1), 9–39.

Moskewicz, M., C. Madigan, Y. Zhao, L. Zhang, and S. Malik. (2001). "Chaff: Engineering an Efficient Sat Solver." In *39th Design Automation Conference*.

Selman, B., H. Levesque, and D. Mitchell. (1992). "A New Method for Solving Hard Satisfiability Problems." In *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI'92, San Jose/CA, USA*, AAAI Press, pp. 440–446.

Smith, B. and M. Dyer. (1996). "Locating the Phase Transition in Binary Constraint Satisfaction Problems." *Artificial Intelligence* 81, 155–181.

Wallace, R. and E. Freuder. (1996). "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, pp. 587–615.

Xing, Z. and W. Zhang. (2005). "MaxSolver: An Efficient Exact Algorithm for (Weighted) Maximum Satisfiability." *Artificial Intelligence* 164(1-2), 47–80.

Zhang, H., H. Shen, and F. Manya. (2003). "Exact Algorithms for MAX-SAT." In *4th Int. Workshop on First order Theorem Proving*, June.