# Solving Satisfiability Problems with Preferences ⋆

Emanuele Di Rosa, Enrico Giunchiglia, and Marco Maratea

DIST, Università di Genova, Viale Causa, 13 – 16145 Genova, Italy
{emanuele,enrico,marco}@dist.unige.it

**Abstract.** Propositional satisfiability (SAT) is a success story in Computer Science and Artificial Intelligence: SAT solvers are currently used to solve problems in many different application domains, including planning and formal verification. The main reason for this success is that modern SAT solvers can successfully deal with problems having millions of variables. All these solvers are based on the Davis-Logemann-Loveland procedure (DLL). In its original version, DLL is a decision procedure, but it can be very easily modified in order to return one or all assignments satisfying the input set of clauses, assuming at least one exists. However, in many cases it is not enough to compute assignments satisfying all the input clauses: Indeed, the returned assignments have also to be "optimal" in some sense, e.g., they have to satisfy as many other constraints –expressed as preferences– as possible.
In this paper we start with qualitative preferences on literals, defined as a partially ordered set (poset) of literals. Such a poset induces a poset on total assignments and leads to the definition of optimal model for a formula $\psi$ as a minimal element of the poset on the models of $\psi$. We show $(i)$ how DLL can be extended in order to return *one* or *all* optimal models of $\psi$ (once converted in clauses and assuming $\psi$ is satisfiable), and $(ii)$ how the same procedures can be used to compute optimal models wrt a qualitative preference on formulas and/or wrt a quantitative preference on literals or formulas. We implemented our ideas and we tested the resulting system on a variety of very challenging structured benchmarks. The results indicate that our implementation has comparable performances with other state-of-the-art systems, tailored for the specific problems we consider.

## 1 Introduction

Propositional satisfiability (SAT) is a success story in Computer Science and Artificial Intelligence: SAT solvers are currently used to solve problems in many different application domains, including planning [3], formal verification [4], and many others, such as RNA folding, hand-writing recognition, graph isomorphism and sudoku problems. The main reason for this success is that modern SAT solvers can successfully deal with problems having millions of variables [5, 6].[1] All these solvers are based on the Davis-Logemann-Loveland procedure (DLL) [7]. The original version of DLL is a decision procedure —given a finite set of clauses $\varphi$, DLL returns whether $\varphi$ is satisfiable or not)— but DLL can be easily modified in order to return one or all assignments satisfying $\varphi$, assuming at least one exists. However, in many cases, it is not enough to compute

⋆ This work extends the results presented in [1, 2].

[1] http://www.satcompetition.org/

one or more satisfying assignments: Indeed, the returned assignments have also to be "optimal" in some sense, e.g., they have to satisfy as many other constraints –expressed as preferences– as possible. For example, in standard MIN-ONE (resp. MIN-ONE$_\subseteq$), given a satisfiable instance, the goal is to find a satisfying assignment in which the set of variables assigned to true is of minimal cardinality (resp. subset-minimal). In standard MAX-SAT (resp. MAX-SAT$_\subseteq$), the goal is to find an assignment satisfying as many clauses as possible, i.e., such that the set of satisfied clauses is of maximal cardinality (resp. subset-maximal). In the partial version of MIN-ONE/MIN-ONE$_\subseteq$ (resp. MAX-SAT/MAX-SAT$_\subseteq$) the optimization is performed on a subset of the variables (resp. clauses) of the instance.

In this paper we start considering the simple model in which preferences are expressed as a partially ordered set (poset) of literals as in, e.g., [8, 9]. Such a poset induces a poset on total assignments and leads to the definition of optimal model for a formula $\psi$ as a minimal element of the poset on the models of $\psi$. Given a qualitative preference on literals and a finite set of clauses $\varphi$, we show how DLL can be easily modified in order to return an optimal model of $\varphi$, assuming $\varphi$ is satisfiable. The simple idea for computing *one* optimal model, is to force DLL branching heuristic in order to follow the partial order on the literals. The idea of computing "optimal" (according to some given definition) models by modifying the heuristic in order to follow the expressed preferences on literals has been already proposed in [10] for SAT and in [11] for acyclic CP-nets [12]. There are however some important differences in the underlying formalism used in [10, 11] for expressing preferences —and thus on the procedures based on these formalisms— wrt ours:[2]

1. In the language: Both [10] and [11] allow for expressing preferences on literals, but in these approaches it is not possible to rank the preferences according to a partial order. For instance, these approaches allow for directly expressing a preference in which literals $l_1$ and $l_2$ are assigned to true, but do not allow for expressing that having $l_1$ assigned to true is preferred to having $l_2$ assigned to true. Further, in [10, 11] the set of preferences has to be consistent, while we do not make this assumption.
2. In the semantics: Even considering the case in which preferences are expressed as a consistent set $S$ of literals, the order on models induced by $S$ in [10, 11] is different from ours. For example, given a language with two variables $x_1$ and $x_2$, and assuming that our only preference is to have $x_1$ assigned to true, given an assignment $\mu_1$ (resp. $\mu_2$) assigning both $x_1$ and $x_2$ to true (resp. false), $\mu_1$ is preferred to $\mu_2$ according to our semantics (see Sec. 2), while this is not the case for the semantics in [10, 11] (see Def. 7 in [10] and Sec. 2.3 in [11]).

We then extend our procedure in order to find more, and possibly all, optimal models. As in [10], the idea is to add to the input formula a constraint imposing that the new models have not to follow $\mu$ in the partial order: Thus, assuming we have already generated a non empty set of optimal models and we are interested in more, differently from the procedures in [11, 13] for CP-nets, our algorithm for generating a new optimal

[2] In the case of [12], we consider the simple case in which variables are Boolean and preferences are not conditional.

model $\mu$ never requires a dominance test to see if there exists another model which is preferred to $\mu$. Finally, we show how the same procedures can be extended to compute optimal models wrt a qualitative preference on formulas and a quantitative preference on literals or formulas. Indeed, this is a trivial consequence of the fact that all these concepts (qualitative/quantitative preference on literals/formulas and also their mixing) can be reduced to the basic framework of qualitative preference on literals.

We implemented our ideas in MINISAT [14], and we called $n$OPTSAT the resulting system.[3] In order to comparatively test our system, we focused our experimental analysis on MAX-SAT/MAX-SAT$_\subseteq$ and MIN-ONE/MIN-ONE$_\subseteq$ problems. Indeed, this is a very challenging —if not the most challenging— test bench for our implementation:

1. for MIN-ONE/MAX-SAT, a wide variety of recently developed, customized systems are available, e.g., those in the last Pseudo-Boolean (PB) and Max-SAT Evaluations;[4] and
2. in these problems, the number of preferences is very high (equal to the number of clauses in standard MAX-SAT/MAX-SAT$_\subseteq$ and to the number of variables in standard MIN-ONE/MIN-ONE$_\subseteq$): As we have already shown in [15], in the context of planning as satisfiability, the more preferences we have, the more the performances of our system are negatively affected.

Despite the above, our analysis shows that $n$OPTSAT has comparable performances with respect to other state-of-the-art systems on MIN-ONE/MAX-SAT problems. In the case of MIN-ONE$_\subseteq$/MAX-SAT$_\subseteq$ problems, we consider the only other implementation available for standard MAX-SAT$_\subseteq$, and here again we show that our system compares well. However, we remark that our results goes far beyond the MIN-ONE/MIN-ONE$_\subseteq$ and MAX-SAT/MAX-SAT$_\subseteq$ cases, which

1. are the simplest cases of preferences on literals and on formulas respectively, since they have an empty partial order on preferences; and
2. are the most difficult problems for our system, given the very high number of preferences they have.

Indeed, we allow for preferences to be partially ordered and in many applications the number of preferences is relatively low, as, e.g., in planning with soft goals [16, 15].

Summing up, given a set of clauses $\varphi$, the main contributions of the paper are:

1. We show how it is possible to easily extend DLL in order to compute one optimal model of $\varphi$ wrt a qualitative preference on literals: We allow for inconsistent set of preferences and for a partial order on the preferred literals.
2. We extend the procedure in order to compute and return more than one, and possibly all, optimal models of $\varphi$ wrt a given qualitative preference: Our procedure does not require any dominance test.
3. We show how qualitative preferences on formulas and quantitative preferences on literals or formulas can be reduced to the basic framework of qualitative preferences on literals: This allows to use our procedures also in these extended settings, and also for solving problems with mixed qualitative and quantitative preferences.

[3] Available at `http://www.star.dist.unige.it/~emanuele/nOPTSAT/`.

[4] See `http://www.cril.univ-artois.fr/PB09/` and `http://www.maxsat.udl.cat/09/`, respectively.

4. We implemented these ideas on top of MINISAT and we comparatively tested our system on a variety of structured MIN-ONE/MIN-ONE$_\subseteq$ and MAX-SAT/MAX-SAT$_\subseteq$ problems against various state-of-the-art systems, tailored for such problems: Despite the generality of our procedure, the results indicate that our system has comparable performances wrt the others.

The paper is structured as follows. In Section 2, we present the formalism we are using for expressing qualitative preferences on literals, and we show how these preferences induce a preference on the set of total assignments and thus also on the models of any given formula. In Section 3, we first present OPT-DLL, i.e., DLL modified in order to compute an optimal model of a finite set of clauses wrt a qualitative preference on literals, and then $n$OPT-DLL, i.e., OPT-DLL extended in order to compute all optimal models. In Section 4, we show how it is possible to reduce quantitative/qualitative preference on formulas to qualitative preference on literals. In the same section, we give examples showing how it is possible to represent and solve problems in which qualitative and quantitative preferences are mixed. Section 5 is devoted to the implementation details and the comparative experimental analysis of the ideas presented. The paper ends in Section 6 with the conclusions.

## 2 Satisfiability and Qualitative Preferences

Consider a finite set $P$ of *variables*, called *signature*. A *literal* is a variable $x$ or its negation $\neg x$. A *formula* or *constraint* is either a variable or a combination of formulas using the $n$-ary connectives $\wedge$ and $\vee$ for conjunction and disjunction, respectively ($n \geq 0$); the $n$-ary connective $\equiv$ for equivalence ($n \geq 2$); and the unary connective $\neg$ for negation.

For example, given the 5 variables $\mathit{AtWork}_0, \mathit{AtWork}_1, \mathit{Car}_0, \mathit{Bus}_0, \mathit{Bike}_0$, in which the subscript models the time step, the following formula

$$\mathit{AtWork}_1 \equiv \neg \mathit{AtWork}_0 \equiv (\mathit{Car}_0 \vee \mathit{Bus}_0 \vee \mathit{Bike}_0), \tag{1}$$

models the fact that if we perform the action of taking the car ($\mathit{Car}_0$) or the bus ($\mathit{Bus}_0$) or the bike ($\mathit{Bike}_0$) then we move from home ($\neg \mathit{AtWork}_0$) to work ($\mathit{AtWork}_1$), or viceversa, while the formula

$$\neg \mathit{AtWork}_0 \wedge \mathit{AtWork}_1 \tag{2}$$

models the fact that at time 0 we are at home (or not at work) and at time 1 we are at work (variables have the obvious meaning). The two equations do not rule out the possibility of executing two or even three actions at the same time (e.g., to have a model in which both $\mathit{Car}_0$ and $\mathit{Bus}_0$ are true). Indeed, these models can be easily eliminated by adding additional constraints.

In the following, given a literal $l$, $\bar{l}$ denotes $\neg l$ if $l$ is a variable, and the variable in $l$ otherwise. If $S$ is a set of literals,

$$\overline{S} = \{\bar{l} : l \in S\}.$$

An *assignment* is a consistent set of literals.

Consider an assignment $\mu$.

If $l \in \mu$, we say that both $l$ and $\bar{l}$ are *assigned* by $\mu$, to true and false, respectively. $\mu$ is *total* if each literal is assigned by $\mu$. If $\mu$ is total, we say that $\mu$ *satisfies*

- *a variable* $x$ if $x \in \mu$;
- *a formula* $\psi$ if $\mu$ satisfies $\psi$ according to the truth tables of the propositional connectives;
- *a set of formulas* if $\mu$ satisfies each formula in the set.

A *model* of a set of formulas is an assignment satisfying the set.

For instance, (1) has 16 models, while considering (1) and (2) the models become 7. In the following, we represent a total assignment as the set of variables assigned to true, and we write $\mu \models \psi$ to indicate that $\mu$ is a model of $\psi$. For instance, $\{Car_0, AtWork_1\}$ represents the total assignment in which the only variables assigned to true are $Car_0$ and $AtWork_1$.

As we already said in the introduction, in many applications not all the models of a set of formulas are considered to be equally good. For example, in (1) we may prefer the models in which as few actions as possible are performed, i.e., we may prefer the models having a maximal intersection with $\{\neg Car_0, \neg Bus_0, \neg Bike_0\}$: This intuitively corresponds to considering the 2 models in which we do not move from/to home as optimal. Further, in case it is not possible to have a model with $\neg Car_0, \neg Bus_0, \neg Bike_0$ (as with (1) and (2)), we may want to express the additional information that we have better to give up on $\neg Car_0$ than the other two, i.e., that of the three preferences, $\neg Car_0$ is the least important: With such additional information, we expect $\{Car_0, AtWork_1\}$ to be the only optimal model for (1) and (2).

Such additional information about which model is preferred to the other ones can be expressed via qualitative preferences on literals. A *qualitative preference on literals* is a (strict) partially ordered set of literals.[5] We recall that a *(strict) partially ordered set* (or *poset*) is a pair $\langle S, \prec \rangle$ whose first element is a set and $\prec$ is a *(strict) partial order on* $S$, i.e., a binary relation satisfying the following two properties:

1. *Irreflexivity*: For each $a \in S$, $a \not\prec a$.
2. *Transitivity*: For each $a, b, c \in S$, $a \prec b$ and $b \prec c$ implies $a \prec c$.[6]

Given a poset $\langle S, \prec \rangle$,

- if for each two distinct $a, b \in S$, $a \prec b$ or $b \prec a$ then $\prec$ is said to be a *total order*;
- an element $a \in S$ is said to be *minimal (in* $\langle S, \prec \rangle$*)* if there is no $b \in S$ with $b \prec a$: It is clear that if $S$ is finite then the poset has at least one minimal element; and
- if $S' \subseteq S$ then $\langle S', \prec' \rangle$ is also a poset, where $\prec'$ is $\prec$ restricted to the literals in $S'$.

It is common to represent a poset $\langle S, \prec \rangle$ as the direct acyclic graph (DAG) whose vertexes are the elements in $S$, and with an arc from $a$ to $b$ if and only if $a \prec b$ and there is no $c$ with $a \prec c \prec b$.

In a qualitative preference on literals $\langle S, \prec \rangle$, $S$ is the *set of preferences* and intuitively represents the set of literals that we would like to have satisfied, while $\prec$ models

[5] The given definition of qualitative preference on literals generalizes the ones given in [1].

[6] If $\prec$ is irreflexive and transitive then it is also antisymmetric, i.e., if $a \prec b$ then $b \not\prec a$.

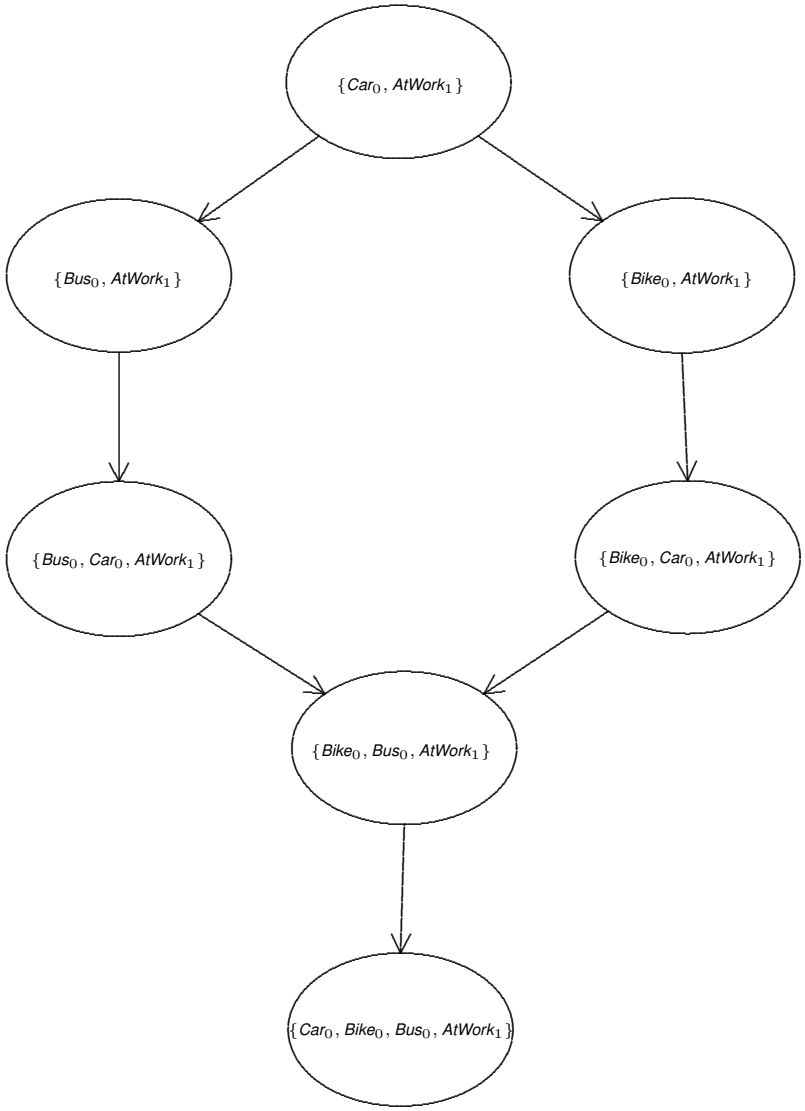


**Fig. 1.** DAG representation of the partial order on the models of (1) and (2), wrt (3).

the relative importance of the preferences. Notice that the set $S$ can be inconsistent: If $S$ contains both a variable $x$ and its negation $\neg x$ —assuming compatibility with the constraints and that neither $x \prec \neg x$ nor $\neg x \prec x$— we expect to have an optimal model with $x$ true and another optimal model with $x$ false. For example,

1. the qualitative preference

$$\langle \{\neg \mathit{Car}_0, \neg \mathit{Bus}_0, \neg \mathit{Bike}_0, \mathit{Bike}_0\}, \emptyset \rangle$$

models our preference to get (assuming compatibility with the underlying constraints) two optimal models $\mu_1$ and $\mu_2$, satisfying $\{\neg \mathit{Car}_0, \neg \mathit{Bus}_0, \neg \mathit{Bike}_0\}$ and $\{\neg \mathit{Car}_0, \neg \mathit{Bus}_0, \mathit{Bike}_0\}$, respectively;

2. while

$$\langle \{\neg \mathit{Car}_0, \neg \mathit{Bus}_0, \neg \mathit{Bike}_0\}, \{\neg \mathit{Bus}_0 \prec \neg \mathit{Car}_0, \neg \mathit{Bike}_0 \prec \neg \mathit{Car}_0\} \rangle \tag{3}$$

models our preference for not moving from/to home, and in which, of the three preferences, $\neg \mathit{Car}_0$ is the least important.

Consider a qualitative preference on literals $\langle S, \prec \rangle$. The partial order on $S$ can be extended to the set of total assignments as follows [9]: Given two total assignments $\mu$ and $\mu'$, $\mu$ *is preferred to* $\mu'$ or $\mu$ *dominates* $\mu'$ *(*$\mu \prec \mu'$*)* if and only if

1. $\mu$ satisfies at least one preference which is not satisfied by $\mu'$, i.e., there exists a literal $l \in S$ with $l \in \mu$ and $\bar{l} \in \mu'$; and
2. the preferences satisfied by $\mu'$ and not by $\mu$ are less preferred to those satisfied by $\mu$ and not by $\mu'$, i.e., for each literal $l \in S \cap (\mu' \setminus \mu)$, there exists a literal $l' \in S \cap (\mu \setminus \mu')$ such that $l' \prec l$.

From the definition, it is clear that, for any two total assignments $\mu$ and $\mu'$:

1. If $S \cap \mu = S \cap \mu'$ then $\mu \not\prec \mu'$: In particular, if the set $S$ of preferences is empty, every model is optimal.
2. If $S \cap \mu' \subset S \cap \mu$ then $\mu \prec \mu'$: Every optimal model has a maximal intersection with $S$. In the case $\prec$ is empty, every model with a maximal intersection with $S$ is optimal.

$\langle S, \prec \rangle$ induces a partial order on the set of total assignments, as stated by the following theorem, similar to Theorem 7 in [17].

**Theorem 1.** *Let $\langle S, \prec \rangle$ be a qualitative preference on literals. The relation $\prec$ extended to the set of total assignments is a partial order.*

**Proof.** For a literal $l$ and a total assignment $\mu$, define

$$\mathrm{dom}(l, \mu) = \{l' : l' \in S \cap \mu, l' \prec l\}.$$

We have to show that the relation $\prec$ on the set of total assignments is irreflexive and transitive. Let $\mu, \mu'$ and $\mu''$ be three total assignments.

1. *Irreflexivity*: Clearly $\mu \not\prec \mu$, because of the first condition in the definition of $\prec$.
2. *Transitivity*: $\mu \prec \mu'$ and $\mu' \prec \mu''$ implies $\mu \prec \mu''$. We have to prove that for each literal $l' \in S \cap (\mu'' \setminus \mu)$, there exists a literal $l \in S \cap (\mu \setminus \mu'')$ such that $l \prec l'$. Considering the set
$$S' = S \cap (\mu'' \setminus \mu),$$
it is enough to show that for each literal $l'$ which is minimal in $\langle S', \prec \rangle$, there exists a literal $l \in S \cap (\mu \setminus \mu'')$ such that $l \prec l'$.
Let $l'$ be a minimal element in $\langle S', \prec \rangle$.
There are two cases:

(a) $l' \in \mu'$ and $\text{dom}(l', \mu'') \subseteq \mu'$: Since $l' \in \mu'$ and $\mu \prec \mu'$, let $l$ be a literal in $S \cap \mu$ such that $l \prec l'$, and $l \notin \mu'$. Because of the second initial assumption, $l \notin \mu''$ and thus the thesis.

(b) $l' \notin \mu'$ or $\text{dom}(l', \mu'') \not\subseteq \mu'$: The set $(\{l'\} \cup \text{dom}(l', \mu'')) \setminus \mu'$ is not empty: Let $l''$ be a minimal element of this set according to $\prec$. Notice that either $l'' = l'$ or $l'' \prec l'$. In both cases, $l'' \in \mu''$, $l'' \notin \mu'$, and

$$\text{dom}(l'', \mu'') \subseteq \mu'. \tag{4}$$

Since $\mu' \prec \mu''$, there exists a literal $l''' \in \mu' \setminus \mu''$ with $l''' \prec l''$ and thus $l''' \prec l'$. If $l''' \in \mu$, $l'''$ is the literal $l$ we are looking for and thesis follows. If $l''' \notin \mu$, since $\mu \prec \mu'$ there exists a literal $l \in S \cap (\mu \setminus \mu')$ such that $l \prec l'''$ and thus $l \prec l'$. This literal is not in $\mu'$ and thus, because of (4), it is not in $\mu''$ as well. □

Since the set $M$ of models of an arbitrary set of constraints is a subset of the set of total assignments, then also $\langle M, \prec \rangle$ is a partially ordered set. For example, given the constraints (1) and (2), and the qualitative preference (3), the partial order on models is represented by the DAG in Figure 1.

However, there can be partial orders on models (or on total assignments) which are not extensions of partial order on literals. For example, assuming we have only two variables $x_0, x_1$, the total order on the set of total assignments $\{\overline{x}_1, \overline{x}_0\} \prec \{x_1, \overline{x}_0\} \prec \{x_1, x_0\} \prec \{\overline{x}_1, x_0\}$ can not be obtained as the result of the extension of a partial order on a subset of the literals in $\{x_0, \overline{x}_0, x_1, \overline{x}_1\}$. In Section 4 we extend the formalism in order to express qualitative preferences on formulas. With preferences on formulas, given that a total assignment $\mu$ can be represented as the conjunction $C_\mu$ of the literals in $\mu$, it is possible to capture any partial order on the set of total assignments: Trivially, given any two total assignments $\mu_1$ and $\mu_2$ for which we want $\mu_1 \prec \mu_2$, imposing the preference on formulas $C_{\mu_1} \prec C_{\mu_2}$ leads to the desired order on $\mu_1$ and $\mu_2$.

Given that $\mu \prec \mu'$ encodes the fact that $\mu$ is preferred to $\mu'$, it is natural to define a model as *optimal* if it is a minimal element of the partially ordered set of models. A finite set of constraints and a qualitative preference defines an *optimization problem*, consisting in determining an optimal model of the set of constraints. As it is clear from Figure 1, the assignment $\{Car_0, AtWork_1\}$ is the only solution for the optimization problem consisting of the constraints (1), (2), and the qualitative preference (3).

## 3 Solving optimization problems with DLL

Given a finite set of constraints and a qualitative preference on literals, we show how it is possible to compute an optimal model by a simple modification of the Davis-Logemann-Loveland procedure (DLL) [7], and then how to extend DLL in order to compute all optimal models.

DLL is at the basis of current state-of-the-art procedures for checking the satisfiability of a finite set of constraints, which extend it in many ways, including conflict driven backjumping and clause learning, see, e.g., [18, 19]. However, DLL does not directly handle arbitrary formulas, but finite sets of clauses (the clauses to be interpreted

in conjunction among them), where a *clause* is a finite set of literals (the literals to be interpreted in disjunction among them). This is not a limitation because of well known and efficient clause form transformation procedures (see, e.g., [20–22]). Thus, from here on —with the exception of the text in the formal statements— we assume we are able to use DLL and our procedures also on formulas, implicitly assuming that formulas are converted into a set of clauses beforehand.

### 3.1 Computing an optimal model with DLL

Our procedure, that we call OPT-DLL, is a simple modification of DLL in order to take into account the given qualitative preference on literals $\langle S, \prec \rangle$.

$\varphi :=$ ⟨a finite set of clauses in the signature $P$⟩;
$\langle S, \prec \rangle :=$ ⟨a qualitative preference on literals⟩;

**function** OPT-DLL() **return** OPT-DLL-REC($\emptyset$)

**function** OPT-DLL-REC($\mu$)
1 **if** ($\emptyset \in \varphi_\mu$) **return** FALSE;
2 **if** ($\mu$ is total) **return** $\mu$;
3 **if** ($\{l\} \in \varphi_\mu$) **return** OPT-DLL-REC($\mu \cup \{l\}$);
4 $l :=$ *ChooseLiteral*($\mu$);
5 $v :=$ OPT-DLL-REC($\mu \cup \{l\}$);
6 **if** ($v \neq$ FALSE) **return** $v$;
7 **return** OPT-DLL-REC($\mu \cup \{\bar{l}\}$).

**Fig. 2.** The algorithm of OPT-DLL for computing *one* optimal solution.

The pseudo-code of OPT-DLL is represented in Figure 2, where:

- $\varphi$ and $\langle S, \prec \rangle$ are global variables storing the input set of clauses and the qualitative preference on literals, respectively;
- $\mu$ is the current assignment, initially empty;
- $\varphi_\mu$ is the result of simplifying the input set of clauses wrt the assignment $\mu$, i.e., $\varphi_\mu$ is the set of clauses obtained from $\varphi$ by $(i)$ deleting the clauses $C \in \varphi$ such that $C \cap \mu \neq \emptyset$, and $(ii)$ substituting the other clauses $C \in \varphi$ with $C \setminus \overline{\mu}$;
- *ChooseLiteral*($\mu$) returns a literal $l$ unassigned by $\mu$ and such that
  - either $l \in S$ and each literal $l'$ with $l' \prec l$ is assigned by $\mu$;
  - or, if each literal in $S$ is assigned by $\mu$, $l$ is an arbitrary literal in $\varphi_\mu$, selected by any given heuristic.

It is easy to see that if there are no preferences (i.e., if the set $S$ is empty) OPT-DLL is the standard DLL. On the other hand, if the set of preferences is not empty, the search tree is explored in a way to ensure that the returned model (assuming the input formula is satisfiable) is optimal. For instance, assuming we have the qualitative preference

$$\langle \{\neg \mathit{Bus}_0, \neg \mathit{Bike}_0\}, \{\neg \mathit{Bus}_0 \prec \neg \mathit{Bike}_0\} \rangle$$

(modeling the fact that we prefer $(i)$ to not take the bus; $(ii)$ to not take the bike; and $(iii)$ to not go by bus more than to not go by bike) OPT-DLL looks for a model extending

1. $\{\neg \mathit{Bus}_0, \neg \mathit{Bike}_0\}$; if no such model exists, OPT-DLL looks for a model extending
2. $\{\neg \mathit{Bus}_0, \mathit{Bike}_0\}$; if no such model exists, OPT-DLL looks for a model extending
3. $\{\mathit{Bus}_0, \neg \mathit{Bike}_0\}$; if no such model exists, OPT-DLL looks for a model extending
4. $\{\mathit{Bus}_0, \mathit{Bike}_0\}$; if no such model exists, OPT-DLL returns false.

In words, OPT-DLL first looks for a model where both actions of going by bus and bike are false; then one in which we use the bike and not the bus; then one in which we use the bus and not the bike; and only finally for models where we use both the bus and the bike. If $\psi$ is the conjunction of (1) and (2), the model of $\psi$ returned with such exploration is $\{\mathit{Car}_0, \mathit{AtWork}_1\}$, while it is $\{\mathit{Bike}_0, \mathit{AtWork}_1\}$ if we also consider the constraint $\neg \mathit{Car}_0$.

OPT-DLL returns an optimal assignment if the input formula is satisfiable, and FALSE otherwise, as stated by the following theorem, easy consequence of Theorem 4, which is stated and proved in the next subsection.

**Theorem 2.** *Let $\varphi$, $\langle S, \prec \rangle$ and* OPT-DLL *as in Figure 2.* OPT-DLL() *returns an optimal model of $\varphi$ wrt $\langle S, \prec \rangle$ if $\varphi$ is satisfiable, and* FALSE *otherwise.*

Consider a satisfiable and finite set of clauses $\varphi$.

Depending on the literal returned by *ChooseLiteral*$(\mu)$, different optimal models are computed and returned by OPT-DLL. For instance, given the qualitative preference

$$\langle \{\neg \mathit{Car}_0, \neg \mathit{Bus}_0, \neg \mathit{Bike}_0\}, \emptyset \rangle, \tag{5}$$

each of the three optimal models of (1) and (2) wrt (5) can be returned by OPT-DLL, which one depending on the specific implementation of *ChooseLiteral*: If the first two literals assigned at line 4 in OPT-DLL are in

1. $\{\neg \mathit{Car}_0, \neg \mathit{Bus}_0\}$, then the returned optimal model is $\{\mathit{Bike}_0, \mathit{AtWork}_1\}$;
2. $\{\neg \mathit{Car}_0, \neg \mathit{Bike}_0\}$, then the returned optimal model is $\{\mathit{Bus}_0, \mathit{AtWork}_1\}$;
3. $\{\neg \mathit{Bus}_0, \neg \mathit{Bike}_0\}$, then the returned optimal model is $\{\mathit{Car}_0, \mathit{AtWork}_1\}$.

In this example, each optimal model is a possible output of OPT-DLL. There are however optimization problems in which some of the optimal models cannot be returned by OPT-DLL. Consider, for instance, an optimization problem with qualitative preference

$$\langle \{x_0, x_1, x_2, x_3\}, \{x_0 \prec x_1, x_2 \prec x_3\} \rangle$$

and constraints imposing that if one variable in $\{x_0, x_2\}$ is true then the other three variables have to be false,[7] e.g.,

$$(\overline{x}_0 \vee \overline{x}_1) \wedge (\overline{x}_0 \vee \overline{x}_2) \wedge (\overline{x}_0 \vee \overline{x}_3) \wedge (\overline{x}_2 \vee \overline{x}_1) \wedge (\overline{x}_2 \vee \overline{x}_3).$$

[7] For instance $\{x_0, x_1, x_2, x_3\}$ can model the fact that we like to have wine ($x_0$), beer ($x_1$), fish ($x_2$) and pizza ($x_3$); we like wine more than beer, and fish more than pizza. Due to budget limitations, if we buy fish or wine we cannot afford anything else.

The above formula has three models, i.e., $\{x_0\}$, $\{x_2\}$ and $\{x_1, x_3\}$, and all of them are optimal. However, OPT-DLL can return only two of them, namely either $\{x_0\}$ or $\{x_2\}$, which one depending on the implementation of *ChooseLiteral*.

The fact that there can be some optimal model that cannot be returned by OPT-DLL is not a limitation if the goal is to compute *one* optimal model. If the goal is to compute more than one or *all* optimal models, we will see in the next subsection how to generalize OPT-DLL.

### 3.2 Computing all optimal models with DLL

Consider a formula $\psi$ and a qualitative preference $\langle S, \prec \rangle$.

The problem of computing all optimal models of $\psi$ wrt $\langle S, \prec \rangle$ can be solved by

1. determining and printing an optimal model $\mu$ of $\psi$ by imposing an ordering on the splitting heuristic, as in the previous subsection;
2. adding to the input formula a new formula which prunes the assignments which are dominated by $\mu$; and
3. returning FALSE in order to continue the search for other optimal models.

The idea to compute all optimal models by adding constraints pruning the models dominated by the already computed optimal models, has been already proposed in [10]. Other works which exploit further techniques to eliminate previously computed solutions in SAT include, e.g., [23–25] in the context of symbolic model checking [26]. However, the framework used in [10] assumes a consistent set of preferences, does not allow for ordering the preferences and ultimately has a different semantics.

For the above procedure, given an assignment $\mu$, we have to define a formula whose models are the assignments dominated by $\mu$ (wrt $\langle S, \prec \rangle$). Such a formula is

$$(\vee_{l \in S \cap \mu} \overline{l}) \wedge (\wedge_{l \in S \cap \overline{\mu}} (\overline{l} \vee \vee_{l' \in S \cap \mu, l' \prec l} \overline{l'})). \tag{6}$$

**Theorem 3.** *Let $\langle S, \prec \rangle$ be a qualitative preference on literals. A total assignment $\mu$ dominates a total assignment $\mu'$ wrt $\langle S, \prec \rangle$ if and only if $\mu'$ satisfies (6).*

**Proof.** According to the definition, given two total assignments $\mu$ and $\mu'$, $\mu \prec \mu'$ iff

1. there exists a literal $l \in S$ with $l \in \mu$ and $\overline{l} \in \mu'$; and
2. for each literal $l \in S \cap (\mu' \setminus \mu)$, there exists a literal $l' \in S \cap (\mu \setminus \mu')$ such that $l' \prec l$.

The first condition corresponds to the formula

$$\vee_{l \in S \cap \mu} \overline{l}.$$

The second condition corresponds to the formula

$$\wedge_{l \in S \cap \overline{\mu}} (\overline{l} \vee \vee_{l' \in S \cap \mu, l' \prec l} \overline{l'}).$$

The conjunction of the above two formulas is (6). □

As examples of the application of Theorem 3, consider a total assignment $\mu$:

1. If $S \cap \mu = \emptyset$ then the formula (6) is equivalent to the empty disjunction, i.e., FALSE: Indeed, if $\mu$ does not satisfy any preference, no assignment is dominated by $\mu$;
2. If $S \subseteq \mu$ then the formula (6) is equivalent to $\vee_{l \in S} \bar{l}$: Each assignment which does not satisfy all the preferences is dominated by $\mu$;
3. If $\prec = \emptyset$ then, the formula (6) is equivalent to $\vee_{l \in S \cap \mu} \bar{l} \wedge \wedge_{l \in S \cap \overline{\mu}} \bar{l}$: Each assignment satisfying a strict subset of the set of preferences satisfied by $\mu$, is dominated by $\mu$.

Notice that if $\mu_1$ dominates $\mu_2$ and $\psi_1$ (resp. $\psi_2$) is the formula (6) computed for $\mu_1$ (resp. $\mu_2$), then $\psi_2$ *entails* $\psi_1$, i.e., the models of $\psi_2$ are a subset of the models $\psi_1$: This is a simple consequence of the fact that if $\mu_1 \prec \mu_2$ then $\mu_1$ dominates a superset of the total assignments dominated by $\mu_2$.

Thanks to Theorem 3, it is possible to generalize OPT-DLL in Figure 2 in order to return all the optimal models of a finite set of clauses $\varphi$. The resulting procedure is represented in Figure 3. In the figure, let $P$ be the signature of $\varphi$, *Reason*$(\mu)$ corresponds to the negation of (6), i.e., *Reason*$(\mu)$ is a finite set of clauses —possibly in a signature $P'$ extending $P$— such that

1. for each total assignment $\mu$ satisfying the negation of (6), there exists one assignment $\mu'$ in $P'$ extending $\mu$ and satisfying *Reason*$(\mu)$;
2. for each total assignment $\mu'$ in $P'$ satisfying *Reason*$(\mu)$, the restriction of $\mu'$ to $P$ satisfies the negation of (6).

Such a set of clauses can be computed starting from the negation of (6) using the already mentioned clause form transformations [20–22].

$\varphi$ := $\langle$a finite set of clauses in the signature $P\rangle$;
$\langle S, \prec\rangle$ := $\langle$a qualitative preference on literals$\rangle$;

**function** $n$OPT-DLL() **return** $n$OPT-DLL($\emptyset$)

**function** $n$OPT-DLL($\mu$)
**if** ($\emptyset \in \varphi_\mu$) **return** FALSE;
**if** ($\mu$ is total)
  *Print*($\mu \cap (P \cup \overline{P})$);
  $\varphi := \varphi \cup$ *Reason*($\mu$);
  **return** FALSE;
**if** ($\{l\} \in \varphi_\mu$) **return** $n$OPT-DLL($\mu \cup \{l\}$);
$l$ := *ChooseLiteral*($\mu$);
$v := n$OPT-DLL($\mu \cup \{l\}$);
**if** ($v \neq$ FALSE) **return** $v$;
**return** $n$OPT-DLL($\mu \cup \{\bar{l}\}$).

**Fig. 3.** The algorithm of $n$OPT-DLL for computing *all* optimal solutions.

As an example, consider the optimization problem discussed at the end of previous subsection, having

$$\langle \{x_0, x_1, x_2, x_3\}, \{x_0 \prec x_1, x_2 \prec x_3\} \rangle$$

as qualitative preference, while the constraint

$$(\overline{x}_0 \vee \overline{x}_1) \wedge (\overline{x}_0 \vee \overline{x}_2) \wedge (\overline{x}_0 \vee \overline{x}_3) \wedge (\overline{x}_2 \vee \overline{x}_1) \wedge (\overline{x}_2 \vee \overline{x}_3),$$

imposes that the only models are $\{x_0\}$, $\{x_2\}$ and $\{x_1, x_3\}$. $n$OPT-DLL

1. starts determining and printing the optimal model $\{x_0\}$ or $\{x_2\}$, which one depending on whether *ChooseLiteral*$(\emptyset)$ returns $x_0$ or $x_2$. Assuming *ChooseLiteral*$(\emptyset) = x_0$, $n$OPT-DLL
   (a) computes and prints the first optimal model $\mu_0 = \{x_0\}$,
   (b) computes (6) for $\mu_0$, i.e.,

   $$\overline{x}_0 \wedge (\overline{x}_1 \vee \overline{x}_0) \wedge \overline{x}_2 \wedge \overline{x}_3,$$

   (c) adds to the input set of clauses a set of clauses corresponding to the negation of the previous formula, e.g.,

   $$x_0 \vee x_2 \vee x_3;$$

2. backtracks setting $\overline{x}_0$, and continues the search looking for models extending the partial assignment $\{\overline{x}_0\}$. Assuming *ChooseLiteral*$(\{\overline{x}_0\}) = x_2$, $n$OPT-DLL
   (a) determines and prints the second optimal model $\mu_1 = \{x_2\}$,
   (b) computes (6) for $\mu_1$, i.e.,

   $$\overline{x}_2 \wedge (\overline{x}_3 \vee \overline{x}_2) \wedge \overline{x}_0 \wedge \overline{x}_1,$$

   (c) adds to the input set of clauses a set of clauses corresponding to the negation of the previous formula, e.g.,

   $$x_2 \vee x_0 \vee x_1;$$

3. backtracks setting $\overline{x}_2$, and continues the search looking for models extending the partial assignment $\{\overline{x}_0, \overline{x}_2\}$. $n$OPT-DLL
   (a) determines and prints the third (and last) optimal model $\mu_2 = \{x_1, x_3\}$,
   (b) computes (6) for $\mu_2$, i.e.,

   $$(\overline{x}_1 \vee \overline{x}_3) \wedge \overline{x}_0 \wedge \overline{x}_2,$$

   (c) adds to the input set of clauses a set of clauses corresponding to the negation of the previous formula, e.g.,

   $$(x_1 \vee x_0 \vee x_2) \wedge (x_3 \vee x_0 \vee x_2); \tag{7}$$

4. backtracks setting
   (a) either $\overline{x}_1$ and thus the current assignment is $\{\overline{x}_0, \overline{x}_2, \overline{x}_1\}$
   (b) or $\overline{x}_3$ and thus the current assignment is $\{\overline{x}_0, \overline{x}_2, \overline{x}_3\}$.
   In both cases the formula (7) is contradicted, and given that the search tree has been entirely explored, $n$OPT-DLL terminates returning FALSE.

$n$OPT-DLL prints all and only the optimal models of $\varphi$ wrt $\langle S, \prec \rangle$ as stated by the following theorem.

**Theorem 4.** *Consider Figure 3, and let $\varphi$, $\langle S, \prec \rangle$ and $n$*OPT-DLL *as in the figure. $n$*OPT-DLL*() prints all and only the optimal models of $\varphi$ wrt $\langle S, \prec \rangle$.*

**Proof.** Assume $\varphi$ is satisfiable, otherwise $n$OPT-DLL prints nothing and the thesis trivially holds.

Let $\mu_1, \ldots, \mu_n$ ($n \geq 1$) be the models printed by $n$OPT-DLL, listed according to the order in which they are printed (thus $\mu_1$ is the first model printed by $n$OPT-DLL). The first observation is that if a model $\mu$ of the input set of clauses is not printed by $n$OPT-DLL then $\mu$ is not optimal. Indeed, if $\mu$ is not printed, then $\mu$ falsifies one of the clauses added to $\varphi$ when a model $\mu'$ is determined and printed, and this implies that $\mu' \prec \mu$. Thus, the set $T = \{\mu_1, \ldots, \mu_n\}$ is a superset of the set of optimal models.

It remains to be showed that each model $\mu$ in $T$ is optimal, i.e., that for each other model $\mu' \in T$, $\mu' \not\prec \mu$.

For any $i, j$ with $1 \leq i < j \leq n$, $\mu_j \not\prec \mu_i$. To show this,

1. let $l_1; \ldots; l_k; l_{k+1}; l_{k+2}; \ldots; l_{|P|}$ be the sequence of literals in $\mu_i$ listed according to the order in which they are assigned by $n$OPT-DLL, and
2. let $l_1; \ldots; l_k; \overline{l_{k+1}}; l'_{k+2}; \ldots; l'_{|P|}$ be the sequence of literals in $\mu_j$ listed according to the order in which they are assigned by $n$OPT-DLL,

($k \geq 0$). Thus, $l_{k+1}$ is the first literal which is assigned differently by $\mu_i$ and $\mu_j$, i.e., $l_{k+1}$ has been assigned at line 8 and $\overline{l_{k+1}}$ has been assigned at line 10 after $l_1; \ldots; l_k$ by $n$OPT-DLL. There are two cases:

1. $l_{k+1} \in S$ and thus by definition of *ChooseLiteral* all the literals $l \prec l_{k+1}$ are assigned by $\{l_1, \ldots, l_k\}$ and thus in the same way by $\mu_i$ and $\mu_j$. Hence $\mu_j \not\prec \mu_i$.
2. $l_{k+1} \notin S$ and thus by definition of *ChooseLiteral* all the literals in $S$ are assigned by $\{l_1, \ldots, l_k\}$ and thus in the same way by $\mu_i$ and $\mu_j$. Hence $\mu_j \not\prec \mu_i$.

Thus, each model $\mu_i$ is optimal:

1. For each model $\mu_k$ with $k < i$ (i.e., printed before $\mu_i$), $\mu_k \not\prec \mu_i$ because $\mu_i$ satisfies the formula (6) added to $\varphi$ when $\mu_k$ has been computed.
2. For each model $\mu_k$ with $k > i$ (i.e., printed after $\mu_i$), we have shown that $\mu_k \not\prec \mu_i$.

□

## 4 Quantitative and Qualitative Preferences on Formulas and their mixing

In this section, we first introduce quantitative preferences on literals and then we generalize the concept of preferences on literals to preferences on formulas, showing how all these notions (both separately or mixed) can be reduced to the basic framework of qualitative preference on literals.

### 4.1 Quantitative Preferences on literals

Given a set of preferences $S$ and a formula $\psi$, if it is not possible to satisfy both $S$ and $\psi$, a standard approach to model the relative importance of the preferences in $S$, is to define a function $c : S \mapsto \mathcal{N}^+$: Intuitively, $c(l)$ is the reward for satisfying $l \in S$. A pair $\langle S, c\rangle$ is a *quantitative preference* and a model $\mu$ of $\psi$ is *optimal* if it maximizes the *objective function* defined as[8]

$$\sum_{l \in S \cap \mu} c(l). \tag{8}$$

In the literature, such kind of problem is also known as Binate Covering Problem [27], recently generalized in [28].

Considering (1) and (2), if we have the preferences

1. $\{\neg \mathit{Bike}_0, \neg \mathit{Bus}_0, \neg \mathit{Car}_0\}$, assuming the reward function $c$ is constant, then the optimal models are $\{\mathit{Bike}_0, \mathit{AtWork}_1\}$, $\{\mathit{Bus}_0, \mathit{AtWork}_1\}$, $\{\mathit{Car}_0, \mathit{AtWork}_1\}$.
2. $\{\neg \mathit{Bike}_0, \neg \mathit{Bus}_0, \neg \mathit{Car}_0\}$, if we assume $c(\neg \mathit{Bike}_0) = 2$ while $c(\neg \mathit{Bus}_0) = 1$ and $c(\neg \mathit{Car}_0) = 1$, then the optimal models are $\{\mathit{Bus}_0, \mathit{AtWork}_1\}$ and $\{\mathit{Car}_0, \mathit{AtWork}_1\}$.

Consider a quantitative preference $\langle S', c\rangle$ and a satisfiable set of clauses $\varphi'$.

The problem of finding an (resp. all) optimal model (resp. models) of $\varphi'$ wrt $\langle S', c\rangle$ can be solved again using OPT-DLL (resp. $n$OPT-DLL) as core engine. The basic idea is to encode the value of the objective function (8) as a sequence of bits $b_{n-1}, \ldots, b_0$ and then consider the qualitative preference $\langle \{b_{n-1}, \ldots, b_0\}, \{b_i \prec b_j : 0 \leq j < i < n\}\rangle$. In more details, let $\mathit{adder}(S', c)$ be a set of clauses such that:

1. If $n = \lceil log_2(\sum_{l \in S'} c(l)+1)\rceil$, $\mathit{adder}(S', c)$ contains $n$ new variables $b_{n-1}, \ldots, b_0$; and
2. A total assignment $\mu$ satisfies $\varphi'$ iff there exists a unique total assignment $\mu'$ to the variables in $\varphi'$ and in $\mathit{adder}(S', c)$ such that
   (a) $\mu'$ extends $\mu$ and satisfies both $\varphi'$ and $\mathit{adder}(S', c)$, and
   (b) $\sum_{l \in S' \cap \mu} c(l) = \sum_{i=0}^{n-1} \mu'(b_i) \times 2^i$, where $\mu'(b_i)$ is 1 if $b_i \in \mu'$, and is 0 otherwise.

If the above conditions are satisfied, we say that $\mathit{adder}(S', c)$ is a *Boolean encoding of* $\langle S', c\rangle$ *with output* $b_{n-1}, \ldots, b_0$. $\mathit{adder}(S', c)$ can be realized in polynomial time in many ways, see, e.g., [29].

In the above hypotheses, if

1. $\varphi$ is the set of clauses in $\varphi'$ or in $\mathit{adder}(S', c)$, and
2. $\langle S, \prec\rangle$ is the qualitative preference $\langle \{b_{n-1}, \ldots, b_0\}, \{b_i \prec b_j : 0 \leq j < i < n\}\rangle$

[8] Assuming we want $c(l) < 0$ for some $l \in S$, we can replace $l$ with $\bar{l}$ in $S$ and define $c(\bar{l}) = -c(l)$: The set of optimal models does not change. Given $\langle S, c\rangle$ and assuming we are interested in minimizing the objective function (8), we can consider the quantitative preference $\langle \overline{S}, c'\rangle$ with $c'(l) = c(\bar{l})$, and then look for a model maximizing $\sum_{l \in \overline{S} \cap \mu} c'(l)$.

then OPT-DLL (resp. $n$OPT-DLL) returns one optimal model (resp. prints all the optimal models) of $\varphi'$ wrt $\langle S', c\rangle$. The following theorem formally states this result for $n$OPT-DLL.

**Theorem 5.** *Let $\varphi'$ be a set of clauses and let $\langle S', c\rangle$ be a quantitative preference on literals. Let adder$(S', c)$ be a Boolean encoding of $\langle S', c\rangle$ with output $b_{n-1}, \ldots, b_0$. If*

1. *$\varphi$ is the set of clauses in $\varphi'$ or in adder$(S', c)$, and*
2. *$\langle S, \prec\rangle$ is the qualitative preference $\langle\{b_{n-1}, \ldots, b_0\}, \{b_i \prec b_j : 0 \leq j < i < n\}\rangle$*
3. *$M$ is the set of models of $\varphi$ printed by $n$OPT-DLL in Figure 3,*

*then the assignments in $M$, restricted to the signature of $\varphi'$, are all the optimal models of $\varphi'$ wrt $\langle S', c\rangle$.*

**Proof.** The qualitative preference $\langle\{b_{n-1}, \ldots, b_0\}, \{b_i \prec b_j : 0 \leq j < i < n\}\rangle$ induces a partial order on the models of $\varphi' \cup$ *adder*$(S', c)$ according to which $\mu \prec \mu'$ if and only if

$$\sum_{i=0}^{n-1} \mu(b_i) \times 2^i > \sum_{i=0}^{n-1} \mu'(b_i) \times 2^i,$$

i.e., if and only if

$$\sum_{l \in S' \cap \mu} c(l) > \sum_{l \in S' \cap \mu'} c(l).$$

□

As an application of the above theorem, given a quantitative preference with preferences $S' = \{\neg \mathit{Bike}_0, \neg \mathit{Bus}_0, \neg \mathit{Car}_0\}$ and reward function $c$ always returning 1, then *adder*$(S', c)$ has two bits $b_1, b_0$ as output; and the models of *adder*$(S', c)$ satisfy

$$\begin{gathered} b_0 \equiv (\neg \mathit{Bike}_0 \equiv \neg \mathit{Bus}_0 \equiv \neg \mathit{Car}_0), \\ b_1 \equiv ((\neg \mathit{Bike}_0 \wedge \neg \mathit{Bus}_0) \vee (\neg \mathit{Bike}_0 \wedge \neg \mathit{Car}_0) \vee (\neg \mathit{Bus}_0 \wedge \neg \mathit{Car}_0)). \end{gathered} \tag{9}$$

The optimal models of (1), (2) and (9), given the qualitative preference $\langle\{b_1, b_0\}, b_1 \prec b_0\rangle$ are $\{\mathit{Bike}_0, \mathit{AtWork}_1, b_1\}$, $\{\mathit{Bus}_0, \mathit{AtWork}_1, b_1\}$, $\{\mathit{Car}_0, \mathit{AtWork}_1, b_1\}$, i.e., the models of (1), (2) whose objective value is 2.

### 4.2 Qualitative and Quantitative Preferences on Formulas

So far, a preference is a literal, and we have seen how it is possible to use DLL to find optimal models wrt both qualitative and quantitative preferences on literals. We now show that the hypothesis that preferences are literals can be waved, i.e., that it is possible to generalize the previous concepts and results from literals to arbitrary formulas. The basic idea is to introduce definitions [20] or "names" [21] for the formulas at hand (see also [30, 31]).

First, we define a *qualitative preference on formulas* to be a pair $\langle S, \prec\rangle$ where $S$ is a finite set of formulas and $\prec$ a (strict) partial order on $S$. The set $S$ of preferences does not need to be consistent. Then, as in Section 2, the partial order on $S$ induces a partial order on the sets of total assignments according to which, if $\mu$ and $\mu'$ are two total assignments $\mu \prec \mu'$ if and only if

1. there exists a formula $\psi \in S$ satisfied by $\mu$ and not by $\mu'$; and
2. for each formula $\psi' \in S$ satisfied by $\mu'$ and not by $\mu$, there exists a formula $\psi \in S$ satisfied by $\mu$ and not by $\mu'$ such that $\psi \prec \psi'$.

It is easy to see that if the formulas in $S$ are literals, then the above definition coincides with the one given in Section 2. It is also straightforward to generalize the result of Theorem 1 saying that the if $\langle S, \prec \rangle$ is a qualitative preference on formulas, the relation $\prec$ extended to the set of total assignments is a partial order.

For example,

$$\begin{array}{l}\langle \{\mathit{AtWork}_0 \vee \neg \mathit{AtWork}_1 \vee \neg \mathit{Bus}_0, \mathit{AtWork}_0 \vee \neg \mathit{AtWork}_1 \vee \neg \mathit{Car}_0,\\ \neg \mathit{AtWork}_0 \vee \mathit{AtWork}_1 \vee \neg \mathit{Bus}_0, \neg \mathit{AtWork}_0 \vee \mathit{AtWork}_1 \vee \neg \mathit{Car}_0\},\\ \qquad \{\mathit{AtWork}_0 \vee \neg \mathit{AtWork}_1 \vee \neg \mathit{Bus}_0 \prec \mathit{AtWork}_0 \vee \neg \mathit{AtWork}_1 \vee \neg \mathit{Car}_0,\\ \qquad \neg \mathit{AtWork}_0 \vee \mathit{AtWork}_1 \vee \neg \mathit{Car}_0 \prec \neg \mathit{AtWork}_0 \vee \mathit{AtWork}_1 \vee \neg \mathit{Bus}_0\}\rangle\end{array}$$

models the preference in which

1. we prefer to use neither the car nor the bus for moving from/to home, but
2. we prefer to use the car more than the bus for moving to work, and
3. we prefer to use the bus more than the car for moving to home.

A model $\mu$ of a formula $\psi$ is *optimal wrt a qualitative preference on formulas* $\langle S, \prec \rangle$ if $\mu$ is a minimal element of the partial order on the models of $\psi$.

Consider a formula $\psi$ and a qualitative preference on formulas $\langle S, \prec \rangle$.

Instead of $\psi$ and $\langle S, \prec \rangle$ we can consider

1. the qualitative preference on literals $\langle L_s, \prec_S \rangle$, where
   - $L_S$ has a newly introduced variable $x_\alpha$ for each formula $\alpha \in S$, and
   - $x_\alpha \prec_S x_\beta$ if and only if $\alpha \prec \beta$: and
2. the formula
$$\psi \wedge \wedge_{\alpha \in S}(x_\alpha \equiv \alpha). \tag{10}$$

Then, if
$$\mu_S = \mu \cup \{x_\alpha : \alpha \in S, \mu \models \alpha\} \cup \{\neg x_\alpha : \alpha \in S, \mu \not\models \alpha\}$$
it is straightforward to see that a model $\mu$ of $\psi$ is optimal wrt the qualitative preference on formulas $\langle S, \prec \rangle$ iff $\mu_S$ is an optimal model of (10) wrt the qualitative preference on literals $\langle L_S, \prec_S \rangle$. It is also easy to see that (10) can be simplified to
$$\psi \wedge \wedge_{\alpha \in S}(\neg x_\alpha \vee \alpha) \tag{11}$$
and we obtain again the desired correspondence between the models of $\psi$ and (11).

Introducing definitions [20] or "names" [21] for the formulas in the preferences allows us also to reduce quantitative preferences on formulas (defined in the obvious way) to quantitative preferences on literals. Further, it allows us to use OPT-DLL and $n$OPT-DLL as engines for computing optimal models of $\psi$ given a qualitative/quantitative preference on formulas. Similar modeling approaches have been presented in, e.g., [32, 33].

Notice that in our approach, quantitative preferences are reduced to qualitative ones. An advantage of this reduction is that it makes also possible to mix the two, e.g., we can ask (we assume $b_{n-1}, \ldots, b_0$ to be the output bits of $\mathit{adder}(S', c)$):

1. Which among the optimal models according to a qualitative preference $\langle S, \prec \rangle$, are optimal according to a quantitative preference $\langle S', c \rangle$: Such assignments correspond to the optimal models of $\psi \wedge \mathit{adder}(S', c)$ wrt the qualitative preference

$$\langle S \cup \{b_{n-1}, \ldots, b_0\}, \prec \cup \{b_i \prec b_j : 0 \leq j < i < n\} \cup \{\alpha \prec b_i : \alpha \in S, 0 \leq i < n\} \rangle.$$

   The above preference forces OPT-DLL to consider first $\langle S, \prec \rangle$ and then $\langle S', c \rangle$.
2. or which among the optimal models according to a quantitative preference $\langle S', c \rangle$, are optimal according to a qualitative preference $\langle S, \prec \rangle$: Such assignments correspond to the optimal models of $\psi \wedge \mathit{adder}(S', c)$ wrt the qualitative preference

$$\langle S \cup \{b_{n-1}, \ldots, b_0\}, \prec \cup \{b_i \prec b_j : 0 \leq j < i < n\} \cup \{b_i \prec \alpha : \alpha \in S, 0 \leq i < n\} \rangle.$$

   The above preference forces OPT-DLL to consider first $\langle S', c \rangle$ and then $\langle S, \prec \rangle$.

For example, when buying a computer, assuming that we have a qualitative preference $\langle S, \prec \rangle$ on components, we may want to know

1. which are the least expensive computers among the ones which are optimal according to $\langle S, \prec \rangle$,
2. or, alternatively, what is the optimal computer according to $\langle S, \prec \rangle$ among the least expensive ones.

## 5 Implementation and experimental results

In order to test the viability of our approach we implemented our ideas in MINISAT [14], the 2005 version, winner of the SAT 2005 competition on the industrial benchmarks category (together with the SAT/CNF minimizer SATELITE [34]). Such choice is motivated by our interest in solving, in particular, large structured problems coming from applications in general, planning and formal verification in particular. It has to be noted that MINISAT is one of the most famous and efficient implementation of what is nowadays called a Conflict-Driven Clause Learning (CDCL) SAT solver [35]. However, this choice is driven by the interest on solving structured instances; other SAT solvers, e.g., SATZ, does not have learning incorporated, and may perform (much) better on different (e.g., random or crafted) benchmarks. Indeed, the algorithm in Figure 2 is focused on DLL, which is the basic part of almost all available SAT solvers. Issues related to the use of SAT solvers with non-chronological backtracking and learning to solve optimization problems have been first studied in [36], and then in more recent publications by the same authors.

For comparative benchmarking, we focused on MIN-ONE/MIN-ONE$_\subseteq$ and MAX-SAT/MAX-SAT$_\subseteq$ problems, precisely defined as below:

1. Given a set $S$ of variables, in MIN-ONE$^S_\subseteq(\varphi)$ (resp. MIN-ONE$^S(\varphi)$) the goal is to find one or all the optimal models of $\varphi$ wrt the qualitative (resp. quantitative) preference on literals $\langle \overline{S}, \emptyset \rangle$ (resp. $\langle \overline{S}, c \rangle$, where $c$ is a constant function). When all variables are considered, i.e., when $S = P$, we say that we are considering a *standard* MIN-ONE/MIN-ONE$_\subseteq$ problem, and a *partial* version of the problem otherwise.

2. Given a set $S$ of clauses, in MAX-SAT$^S_\subseteq(\varphi)$ (resp. MAX-SAT$^S(\varphi)$) the goal is to find one or all the optimal models of $\varphi$ wrt the qualitative (resp. quantitative) preference on formulas $\langle S, \emptyset\rangle$ (resp. $\langle S, c\rangle$, where $c$ is a constant function). When there are no hard constraints, i.e., when $\varphi = \emptyset$, we say that we are considering a *standard* MAX-SAT/MAX-SAT$_\subseteq$ problem, and a *partial* version of the problem otherwise.

It is thus clear that these two categories of problems are the simplest cases of preferences on literals (MIN-ONE/MIN-ONE$_\subseteq$) and on formulas (MAX-SAT/MAX-SAT$_\subseteq$), having an empty partial order. Further,

1. for MIN-ONE and MAX-SAT problems, highly tuned tools are available, designed for international competitions; and
2. the number of preferences is very high, equal to the number of clauses in MAX-SAT/MAX-SAT$_\subseteq$ and to the number of variables in MIN-ONE/MIN-ONE$_\subseteq$.

Given the above, the goal of the experimental analysis,

1. on MIN-ONE$_\subseteq$/MAX-SAT$_\subseteq$ problems, is to show that our approach for computing optimal models wrt qualitative preferences is viable also when the number of preferences is very high; and
2. on MIN-ONE/MAX-SAT problems, is to show that our reduction from quantitative to qualitative preferences is viable also when the number of preferences is very high.

Problems with a high number of preferences are particularly interesting because the more preferences we have, the more the performances of the underlying SAT solver are negatively affected, see, e.g., [15]. Further, the availability of very efficient tools for MIN-ONE/MAX-SAT gives a good reference point to evaluate the results, also for MIN-ONE$_\subseteq$/MAX-SAT$_\subseteq$: Indeed, an optimal solution for MIN-ONE (resp. MAX-SAT) is also optimal for MIN-ONE$_\subseteq$ (resp. MAX-SAT$_\subseteq$). However, we want to remark that the applicability of our ideas go far beyond, allowing for solving problems with any partial order on preferences, for which problems no implemented system is available for comparative analysis.

Starting from MINISAT, the modifications needed in order to solve MIN-ONE$_\subseteq$ problems have been minor: In practice, we have modified the VSIDS-like heuristic of MINISAT in order to first branch on the literals in the set of preferences. Analogously for MAX-SAT$_\subseteq$, once preferences on formulas are reduced to preferences on literals. MIN-ONE/MAX-SAT problems also required the implementation of a function *adder* as specified in Section 4.1. As we already said, there are various ways to implement such a function. We used the method described in [29], which takes linear time in the size of the input. We also experimented with the method described in [37] for encoding the objective function. This last encoding has some interesting computational properties, (e.g., it is efficiently coupled with the unit-propagation technique implemented in all DLL-based SAT solvers), but the resulting encoding is quadratic in the size of the input, and for this reason —in our experience— it is usually very effective on small instances, but not practical for instances of medium and big size, like the majority of the ones we used.

Beside the modification in the heuristic, we had also to modify MINISAT internal pre-processor in order to disable the assignment of pure literals.[9]

Concerning the other solvers for MAX-SAT/MIN-ONE, we initially considered both dedicated solvers for MAX-SAT problems —like BF [38], MAXSOLVER [39], TOOLBAR [30, 40] ver. 3.0, MAXSATZ version submitted to the 2007 Evaluation [41], MINIMAXSAT (abbreviated with MMSAT in the Tables) ver. 1.0 [42], MSU1.2 [43]— and generic PB solvers —like OPBDP ver. 1.1.1 [44], PBS ver. 2.1 and ver. 4 [45], MINISAT+ (abbreviated with MSAT+ in the Tables) based on MINISAT ver. 1.13 [46], GLPPB ver. 0.2 (by the same authors of PUEBLO [47])[10], BSOLO ver. 3.0.17 [48]. All these systems are among the state-of-the-art solvers for MAX-SAT or PB problems (see the results of the last evaluations). For the standard MAX-SAT$_\subseteq$ problems, we also considered CAMUS [49]. CAMUS is a system for computing all Minimal Unsatisfiable Subsets (MUSes) of a given formula which, as first step, computes all the MAX-SAT$_\subseteq$ optimal solutions: Of course, in the tables, we consider only the time CAMUS takes for generating the MAX-SAT$_\subseteq$ solutions.

Each solver has been run using its default settings. All the experiments have been run on a Linux box equipped with a Pentium IV 3.2GHz processor and 1GB of RAM. In the Tables, "TIME" indicates that the solver does not solve the instance within 1800 seconds; "MEM" indicates that the solver requires more than the allocated 800MB; "SF" indicates that the solver exits abnormally; "–" indicates that the solver returns an incorrect result. Moreover, best performing system(s) on each benchmark/domain are emphasized, in bold: For a domain, we count number of instances solved, with ties broken by mean CPU time (as customary in Max-SAT evaluations).

Considering the dedicated solvers for MAX-SAT, we discarded BF, MAXSOLVER and TOOLBAR after an initial analysis because they seem to be tailored for relatively small typically randomly generated problems, and are thus not suited to deal with most of the problems we consider. Concerning the PB solvers, we do not show the results for OPBDP, PBS ver 2.1 and ver. 4, and also GLPPB because they are almost always slower than the other systems on the instances that we consider.

The next two subsections show the results for MIN-ONE/MIN-ONE$_\subseteq$ and for MAX-SAT/MAX-SAT$_\subseteq$ problems, respectively.

### 5.1 MIN-ONE and MIN-ONE$_\subseteq$

The results for MIN-ONE/MIN-ONE$_\subseteq$ problems are reported in Table 1, on a variety of well-known, publicly available satisfiable benchmarks. In particular, (1-5) are Formal Verification instances ((1-2) from the Bejing'96 competition, (3-5) by Ofer Shtrichman); (6-14) are planning problems from SATPLAN; (15-20) are Data Encryption Standard instances; (21-26) are quasi group instances. The domains contain, in general, more benchmarks that the one we show: The instances showed representative of the

[9] A literal $l$ is *pure* in a set of clauses $\varphi$ if $\bar{l}$ does not belong to any clause in $\varphi$. If $l$ is pure in $\varphi$, $\varphi$ is satisfiable if and only if $\varphi_{\{l\}}$ is. However, in an optimization problem, it may be the case that there exist optimal models with $\bar{l}$ assigned to true, and thus the necessity to disable the assignment of pure literals in MINISAT internal pre-processor.

[10] `http://www.eecs.umich.edu/~hsheini/pueblo/`.

| | instance | $\#C$ | MSAT+ | BSOLO | MMSAT | $n$OPTSAT | | $n$OPTSAT | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | $T_1$ | *#Sols* | $\#C_{\subseteq}$ | $T_1$ | *#Sols* |
| 1 | bcomp5 | 39 | 0.4 | 4.98 | **0.22** | 1.74 | 3360 A | 40 | 0 | 21600 A |
| 2 | bmax6 | 61 | 8.42 | 1401.72 | **1.41** | 430.33 | 24 T | 62 | 0 | 183072 T |
| 3 | ibm2 | 940 | 19.73 | 19.96 | **2.04** | TIME | 0 T | 966 | 0.02 | 33132 M |
| 4 | ibm3 | 6356 | TIME | TIME | TIME | **25.37** | 16 A | 6371 | 0.39 | 5679 M |
| 5 | gal8 | | SF | MEM | TIME | TIME | 0 T | 9372 | 0.96 | 1540 M |
| 6 | 3blocks | 56 | **0.29** | 0.5 | 3.85 | 0.56 | 1 A | 60 | 0.02 | 174 A |
| 7 | 4blocksb | 66 | **0.24** | 0.65 | 4.87 | 1.3 | 4 A | 66 | 0.05 | 4 A |
| 8 | 4blocks | 108 | **50.94** | 353.32 | 1086.05 | TIME | 1 A | 110 | 0.16 | 55097 A |
| 9 | large.c | 265 | **0.96** | 2.84 | 35.24 | 1.18 | 1 A | 265 | 0.16 | 6 A |
| 10 | large.d | 431 | **7.71** | 51.05 | 506.6 | 42.73 | 4 A | 432 | 0.81 | 106 A |
| 11 | log.a | 135 | **1.39** | TIME | 3.07 | 1.88 | 11305 M | 135 | 0.02 | 108151 M |
| 12 | log.b | 138 | 8.99 | TIME | **3.03** | 6.1 | 16728 M | 138 | 0.02 | 108675 M |
| 13 | rock.a | 65 | **0.2** | 1.18 | 0.4 | 1.58 | 13100 A | 65 | 0.01 | 13100 A |
| 14 | rock.b | 69 | **0.27** | 0.55 | 0.41 | 1.92 | 902 A | 69 | 0.01 | 902 A |
| 15 | r2b3.1 | 141 | 0.2 | **0.07** | 3.36 | 0.12 | 2 A | 141 | 0.02 | 2 A |
| 16 | r2b3.2 | 138 | **0.08** | **0.08** | 2 | 0.11 | 1 A | 138 | 0.02 | 1 A |
| 17 | r3b1.1 | 119 | **1.3** | 5.93 | 22.96 | 28.77 | 1 A | 119 | 0.14 | 1 A |
| 18 | r3b1.2 | 126 | **0.82** | 5.62 | 25.04 | 0.93 | 1 A | 126 | 0.13 | 1 A |
| 19 | r3b2.1 | 217 | **0.46** | 1.32 | 47.03 | 0.72 | 1 A | 217 | 0.08 | 1 A |
| 20 | r3b2.2 | 206 | **0.53** | 1.39 | 42.48 | 0.58 | 1 A | 206 | 0.06 | 1 A |
| 21 | qg1-8 | 64 | **31.06** | 414.21 | 131.28 | 78.73 | 16 A | 64 | 0.84 | 16 A |
| 22 | qg2-7 | 49 | **0.27** | 1.19 | 2.29 | 0.28 | 14 A | 49 | 0.13 | 14 A |
| 23 | qg2-8 | 64 | **21.83** | 200.82 | 111.11 | 43.9 | 2 A | 64 | 1.03 | 2 A |
| 24 | qg3-8 | 64 | **0.1** | 0.41 | 1.29 | 0.32 | 18 A | 64 | 0.02 | 18 A |
| 25 | qg4-9 | 81 | **19.36** | 77.19 | 98.31 | 50.58 | 194 A | 81 | 0.02 | 194 A |
| 26 | qg5-11 | 121 | 0.43 | 1.12 | 4 | **0.34** | 5 A | 121 | 0.1 | 5 A |

**Table 1.** Results on MIN-ONE (columns 3-7) and MIN-ONE$_{\subseteq}$ (column 8) problems.

behavior of the systems, and, in general, smaller (resp. bigger) instances are easy (resp. hard) to solve for most of the systems.

The first column of the table contains the number of the instance, followed by its name in the second column. The third column is the optimal value of the instance. Then, columns 4-7 show the results of each system on the MIN-ONE version of the problem, while the last column shows $n$OPTSAT results on the MIN-ONE$_{\subseteq}$ version of the problem.

The columns for all the systems but $n$OPTSAT, show the CPU time (in seconds) that the system takes to compute an optimal solution. For $n$OPTSAT, we show both the time that $n$OPTSAT takes to compute the first solution (subcolumns $T_1$), and (in subcolumn *#Sols*) the number of optimal solutions computed by $n$OPTSAT in 3600 seconds, together with the indication about whether $n$OPTSAT computed all the solutions before the time out (indicated with "A"), or whether $n$OPTSAT exceeded the time out (indicated

with "T"), or whether $n$OPTSAT exceeded the available memory (indicated with "M").[11] Looking at the results, we see that $n$OPTSAT compares well wrt the other solvers when computing one solution: MINISAT+/BSOLO/MMSAT/$n$OPTSAT are not able to find an optimal solution in 2/4/2/3 cases respectively, and $n$OPTSAT is the only system able to solve the "ibm3" instance. From column *#Sols* it also emerges that our solver is able to compute all the optimal solutions for many problems (21 out of 26), exceeding the available memory in 2 cases and timing out in 3 cases. Considering the 5 cases in which $n$OPTSAT is not able to compute all the optimal solutions with the given resources, we have to stress that in general a user is interested to a relatively small number of solutions (say 20), possibly generated one after the other upon request. Thus, a more reasonable reading of the results in column "*#Sols*" is that our system is able to compute all or at least 20 optimal solutions in all cases but two.

Concerning the results for MIN-ONE$_\subseteq$ problems, these are reported in the last column only for $n$OPTSAT being the only system able to directly deal with these problems. The first two subcolumns show the optimal value returned and the time taken by $n$OPTSAT when computing the first optimal solution; the last subcolumn *#Sols* has the same meaning as before. Looking at the results, the first observation is that almost all the problems are solved in less than 1s, and that also the gal8 problem is solved. However, comparing $n$OPTSAT results on MIN-ONE and MIN-ONE$_\subseteq$ problems, two other observations are in order:

1. Considering the performances in solving MIN-ONE and MIN-ONE$_\subseteq$ problems, we see that the latter are solved in much less time. This could have been expected given that handling MIN-ONE problems requires the encoding of adders counting the number of variables set to true, and many of the examples have more than a thousand variables (the "gal8" instance has $> 58000$ variables: The resulting adder has $> 270000$ variables and $> 1300000$ clauses).
2. Considering the optimal value of the first solution found (columns $\#C$ and $\#C_\subseteq$) we see that for most instances $\#C = \#C_\subseteq$, and, even when not equal, the two values are very close but for rows 3, 4, 6. This points out that $n$OPTSAT for MIN-ONE$_\subseteq$ can be used as a good approximation algorithm for MIN-ONE problems.
3. Considering the number of optimal solutions (columns *#Sols*), $n$OPTSAT is able to determine many more optimal solutions in the case of MIN-ONE$_\subseteq$ problems. Interestingly, in 6 cases $n$OPTSAT is not able to find all solutions in MIN-ONE$_\subseteq$ problems (compared to the 5 problems for MIN-ONE), and in 5 out of the 6 cases, $n$OPTSAT exceeds the available memory: This can be easily explained by the fact that whenever a solution is found, a set of clauses is added to the input formula in order to rule out the solutions which are dominated by the solutions found. However, as we already said, we do not expect that users want to generate, e.g., 183072 optimal models of problem "bmax6", but rather a much smaller number.

[11] Notice that we have two different time outs: 1800 seconds for computing one solution, for all the systems. For $n$OPTSAT, we raised the time out to 3600 seconds when trying to compute all solutions. This explains why for the problem "4blocks" (#8 in the table) for $n$OPTSAT subcolumn $T_1$ we have the value "TIME" indicating that the system did not compute any solution in 1800 seconds, and "1 A" in column "*#Sols*" indicating that the system computed the only optimal solution in less than 3600 seconds.

| | instance | $\#C$ | MSAT+ | BSOLO | MMSAT | $n$OPTSAT | | $n$OPTSAT | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | $T_1$ | *#Sols* | $\#C_\subseteq$ | $T_1$ | *#Sols* |
| 27 | air15 | 58 | 16.37 | **5.35** | 22.75 | 9.53 | 1474 M | 58 | 0.3 | 764 M |
| 28 | block5-2 | 16 | **1.58** | 31.79 | 36.96 | 30.55 | 27 A | 16 | 0.14 | 27 A |
| 29 | dep7 | 23 | **78.92** | TIME | TIME | 187.84 | 1458 M | 23 | 120.6 | 1743 M |
| 30 | driv10 | 20 | **8.39** | 415.89 | 902.16 | 17.19 | 512 A | 20 | 42.77 | 512 A |
| 31 | free3 | 21 | 14.69 | 169.24 | 803.63 | **7.68** | 2 A | 21 | 2.71 | 2 A |
| 32 | log6-9 | 24 | **2.72** | 65.05 | 8.87 | 23.22 | 14171 M | 28 | 0.01 | 7863 M |
| 33 | mprime2 | 9 | TIME | 327.73 | TIME | **49.45** | 786 M | | MEM | 0 M |
| 34 | mprime5 | 11 | TIME | TIME | TIME | **129.08** | 663 M | 11 | 75.97 | 917 M |
| 35 | myst2 | 9 | 160.63 | 236.32 | TIME | **28.71** | 1130 M | 9 | 1340.94 | 571 M |
| 36 | opt11 | 216 | TIME | MEM | TIME | **159.93** | 533 M | 216 | 972.95 | 85 M |
| 37 | path5 | 30 | **25.48** | 1194.79 | 50.46 | 112.57 | 5046 M | 30 | 0.06 | 3196 M |
| 38 | phil29 | 330 | 4.38 | 10.63 | 339.26 | **1.48** | 38196 T | 330 | 0.84 | 20961 M |
| 39 | pipe6 | 8 | **16.15** | 838.54 | 940.27 | 238.65 | 9715 M | 8 | 1.76 | 6320 M |
| 40 | pipet6 | 8 | 232.83 | TIME | TIME | **121.09** | 5152 M | 8 | 29.7 | 3274 M |
| 41 | psr29 | 21 | **18.03** | 234.09 | TIME | 35.12 | 1999 M | | TIME | 0 T |
| 42 | psr31 | 19 | **20.11** | 172.78 | TIME | 24.68 | 632 M | 19 | 5.86 | 750 M |
| 43 | psr47 | 27 | **25.23** | 187.86 | 922.92 | 83.59 | 2197 M | 27 | 2.56 | 3286 M |
| 44 | sat3 | 13 | **16.92** | 559.94 | 195.75 | 22.29 | 192 A | 13 | 3.3 | 192 A |
| 45 | stor7 | 14 | 64.41 | 1305.37 | 436.97 | **59.06** | 7900 M | 14 | 0.19 | 6994 M |
| 46 | truck2 | 17 | 39.31 | TIME | 703.9 | **35.55** | 6510 M | 17 | 1.2 | 2063 M |
| 47 | zeno8 | 15 | **123.3** | 1175.87 | TIME | 156.13 | 1523 M | 15 | 6.88 | 2459 M |

**Table 2.** Results on partial MIN-ONE (columns 3-7) and partial MIN-ONE$_\subseteq$ (column 8) problems.

We also considered partial MIN-ONE/MIN-ONE$_\subseteq$ planning problems generated with SATPLAN 2004, release of 10 Feb. 2006.[12] SATPLAN works as follows: Given a planning problem $\Pi$ and a makespan $n$ (initially set to 0), it

1. generates a corresponding SAT formula $\Pi_n$, and checks $\Pi_n$ for satisfiability;
2. if $\Pi_n$ is satisfiable then SATPLAN stops and a plan with optimal makespan is returned;
3. otherwise, $n$ is increased and the process is repeated.

In our experiments, we selected various planning problems from previous International Planning Competitions (IPCs); we considered the first satisfiable instance generated by SATPLAN, and we fixed the set $S$ of preferences to be the set of action variables: With such preferences, we are looking for a plan with as few action variables as possible set to true. In Table 2 we show the results, where columns have the same meaning as in Table 1.

On these partial MIN-ONE planning problems, $n$OPTSAT performs very well: MINISAT+/BSOLO/MMSAT/$n$OPTSAT are not able to find an optimal solution in 3/5/9/0 cases respectively, and $n$OPTSAT is the only system able to solve the "mprime5" and "opt11" instances. From column *#Sols* it emerges that $n$OPTSAT is able to compute all

[12] `http://www.cs.rochester.edu/u/kautz/satplan/index.htm`.

the optimal solutions only in a few cases (4 out of 21), exceeding the available memory in 16 cases and timing out in 1 case.

Considering also $n$OPTSAT results on partial MIN-ONE$_\subseteq$ (last column), it is no longer true that $n$OPTSAT is faster when solving a partial MIN-ONE$_\subseteq$ problem than the corresponding partial MIN-ONE problem. It is also interesting to notice that, given the available resources in time and memory, for many problems $n$OPTSAT is able to find more optimal solutions in the MIN-ONE case than in the MIN-ONE$_\subseteq$ one (e.g., for "air15"). We believe that this is mostly due to the specific structure of the problems in which

1. in the MIN-ONE case, $n$OPTSAT is able to quickly determine the optimal value $V$ for the solution, i.e., $n$OPTSAT is able to quickly determine the non existence of a plan with less than $V$ actions, and then $n$OPTSAT is free to select the $V$ actions,
2. in the MIN-ONE$_\subseteq$ case, $n$OPTSAT has to pay the price of always setting the variables in the preferences to FALSE, i.e., $n$OPTSAT has always to decide to not execute an action, even though there's a strong evidence (given by the score of the heuristic) that the action has to belong to the plan being built.

### 5.2 MAX-SAT and MAX-SAT$_\subseteq$

| domain | #I | MSAT+ | BSOLO | MAXSATZ | MMSAT | MSU1.2 | $n$OPTSAT | | CAMUS | | $n$OPTSAT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | $T_1$ | A/T/M | $T_1$ | A/T/M | $T_1$ | A/T/M |
| industrial | 94 | 40.96(1) | 219.27(1) | 95.18(3) | 7.505(2) | **60.38(85)** | 262.72(7) | (7/87/0) | 98.93 (54) | (5/89/0) | 492.39 (58) | (2/90/2) |
| spinglass | 5 | 0.86(1) | 76.57(1) | 33.19(3) | **1.09(3)** | 0.0(0) | 7.52(1) | (1/4/0) | 109 (1) | (0/5/0) | 0.00 (5) | (0/5/0) |
| dimacs_mod | 62 | 247.54(7) | 0.01(2) | **59.27(52)** | 194.52(52) | 0.03(3) | 67.44(4) | (2/60/0) | 0.006 (2) | (2/60/0) | 0.01 (62) | (1/61/0) |

**Table 3.** Results on standard MAX-SAT (columns 3-8) and standard MAX-SAT$_\subseteq$ (columns 9-10) problems from the MAX-SAT Evaluation 2008.

For MAX-SAT/MAX-SAT$_\subseteq$ problems we considered non random benchmarks from the Max-SAT Evaluations 2007 and 2008. The results are shown in Table 3 and in Table 4 for the standard and partial case respectively, organized as in the report of such evaluations: Each row corresponds to a domain of benchmarks indicated in the first column,[13] the second column contains the number of instances,[14] and the remaining

[13] The majority of domains are (partial) Max-SAT problems originally designed in PB format and submitted to PB evaluations, and then expressed as Max-SAT problems. The instances we have given to PB solvers are the ones that come from a translation first in the input format of $n$OPTSAT, and then in the PB format: This is because we did want to evaluate the systems on the very same formulation. Results for PB solvers on the original PB instances may thus be different.

[14] The industrial domain contains 112 instances: The 94 mentioned in Table 3 do not contain the biggest instances. Nonetheless, the considered instances are very hard for the solvers considered and already big, i.e., in the order of tens of MB: Thus, the remaining instances are likely to reach the memory limit.

| domain | #I | MSAT+ | BSOLO | MMSAT | MSU1.2 | $n$OPTSAT | | $n$OPTSAT | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | $T_1$ | A/T/M | $T_1$ | A/T/M |
| bcp-fir | 59 | 48.82(22) | 44.89(10) | 142.05(14) | **166.47(48)** | 217.43(44) | (15/12/32) | 2.44(59) | (3/7/49) |
| bcp-mtg | 215 | **0.2(215)** | 48.33(190) | 28.3(208) | 19.15(174) | **0.2(215)** | (81/6/128) | 8.91(215) | (57/100/58) |
| bcp-syn | 74 | **34.27(32)** | 169.22(21) | 61.63(29) | 60.84(32) | 16.46(26) | (16/50/8) | 0.04(74) | (4/26/44) |
| pbo-mqc/nenc | 128 | **124.68(120)** | 248.03(95) | 307.38(67) | 54.72(47) | 156.24(80) | (69/52/7) | 564.64(48) | (9/92/27) |
| pbo-mqc/nlogenc | 128 | **39.19(119)** | 268.95(111) | 279.79(104) | 109.29(70) | 218.03(96) | (77/43/8) | 52.33(110) | (14/25/89) |
| PSEUDO/primes | 148 | 11.52(104) | 22.23(94) | **62.08(107)** | 128.7(24) | 33.11(104) | (90/47/11) | 4.60(130) | (86/17/45) |
| PSEUDO/routing | 15 | 43.74(15) | 373.73(8) | 109.49(14) | **0.85(15)** | 58.65(15) | (0/0/15) | 11.22(15) | (0/0/15) |
| MAXONE/struct | 60 | 2.02(58) | 40.96(60) | **22.5(60)** | 0.30(1) | 546.23(50) | (56/4/0) | 1.45(60) | (56/0/4) |
| MAXCLIQ/struct | 62 | 154.39(22) | 248.26(14) | **61.97(36)** | 1.56(4) | 57.05(19) | (17/45/0) | 0.11(62) | (10/4/48) |

**Table 4.** Results on partial MAX-SAT (columns 3-7) and partial MAX-SAT$_\subseteq$ (column 8) problems from the MAX-SAT Evaluation 2008 (first 5 domains) and from the MAX-SAT Evaluation 2007 (last 4 domains).

columns report, for each system, the number of solved instances in parenthesis, and the average time taken to solve them: So, for example, in the "spinglass" domain there are 5 instances, and MAXSATZ solves 3 of them, taking 33.19 seconds on average. For $n$OPTSAT, we show the data as for the other systems in subcolumn $T_1$, and the number of times $n$OPTSAT

1. is able to compute all solutions (first number in subcolumn (A/T/M)),
2. exceed the available time (second number in subcolumn (A/T/M)),
3. exceed the available memory (third number in subcolumn (A/T/M)).

For example, in the MAX-SAT case of the "spinglass" domain, $n$OPTSAT

1. is able to compute one optimal solution in 1 case taking (on average) 7.52s, and
2. when computing all the optimal solutions, in 1 case terminates normally (meaning that $n$OPTSAT is able to compute all the optimal solutions); in 4 cases times out, in 0 cases exceeds the available memory.

MAXSATZ (resp. CAMUS) can only handle standard MAX-SAT (resp. MAX-SAT$_\subseteq$) problems and thus it does not appear in Table 4. Extended versions of MAXSATZ, namely W-MAXSATZ and INCWMAXSATZ, can handle partial Max-SAT problems: However they have not been added to Table 4 because, while being very effective on random benchmarks, other solvers we evaluated perform better on non random benchmarks, as witnessed by the results of Max-SAT Evaluations 2007 and 2008.

Looking at the result for MAX-SAT, the first observation is that there is not a system clearly outperforming the others: Different systems perform better than the others in different domains, and this is in line with the results of the last evaluations. Indeed, there is a great variety in the techniques used by the solvers and thus we can expect such a variety in the performances, depending on the specific features of the domain at hand. Considering $n$OPTSAT on the MAX-SAT problems, its performances are comparable to those of the other systems: If we rank systems on each domain according first to the number of problems solved and then to the average CPU time reported, we see that $n$OPTSAT is among the top three systems in 8 out of 12 domains, being a top system in the bcp-mtg domain.

On the other hand, considering the MAX-SAT$_\subseteq$ problems, $n$OPTSAT is able to solve almost always all the problems with the notable exception of the pbo-mqc/nenc domain, in which $n$OPTSAT is able to solve more problems in the quantitative than in the qualitative setting. The positive results in the qualitative setting wrt the ones in the quantitative setting echo the ones presented in the previous subsection for MIN-ONE$_\subseteq$/MIN-ONE. However, differently from the MIN-ONE$_\subseteq$/MIN-ONE case, the optimal result (called $\#C_\subseteq$ in the previous tables) of the first solution computed by $n$OPTSAT in the MAX-SAT$_\subseteq$ case differs, sometimes significantly, from the optimal result in the MAX-SAT$_\subseteq$ case (i.e., $\#C$ and $\#C_\subseteq$ in the previous tables): In many cases $\#C_\subseteq$ is about $0.5 \times \#C$, and only for a few problems $\#C_\subseteq = \#C$. Compared to CAMUS, $n$OPTSAT has better performances when the goal is to find one optimal solution: CAMUS on the industrial/spinglass/dimacs_mod domains, is able to find an optimal solution in 54/1/2 cases respectively, compared to the 58/5/62 of $n$OPTSAT. However, for the same domains, CAMUS is able to find all optimal solutions in 5/0/2 cases, while $n$OPTSAT corresponding results are 2/0/1.

Summing up the results on MAX-SAT in this subsection, we have seen that $n$OPTSAT performances are comparable to those of the other systems, which have been specifically designed for solving MAX-SAT problems. On standard MAX-SAT$_\subseteq$ problems, $n$OPTSAT performs better than CAMUS when computing one solution, but worse when computing all solutions. By further analyzing the results of $n$OPTSAT on MAX-SAT, we have noticed a relation between the number of unsatisfied clauses in the optimal solution, and the performance of $n$OPTSAT: Performance seems to be better when there are "few" unsatisfied clauses in the solution.

However, we have to remind that the benchmarks we have considered in this but also in previous subsection are characterized by a very high number of preferences. Indeed, in many applications, including planning with soft goals [16], we can expect problems with a few (in the order of tens) preferences, and with such a low number of preferences there is hardly any difference between the performances of $n$OPTSAT and those of MINISAT when run on the original SAT instance (see [15]).

Finally, our system is not tuned in any way: For example, there is no pre-computation of an upper or lower bound for cutting initial portions of the search tree. Herewith we mention how we could use a (pre)computed lower bound for a MIN-ONE problem in our algorithm.Consider a formula $\psi$; assume the output of the *adder* formula are the variables $b_{n-1}, \ldots, b_0$; and let $lb > 0$ the pre-computed lower bound. It is relatively easy to add to $\psi$ a set of clauses which inhibit the configurations of bits corresponding to a value $o$ s.t. $o \leq lb$. For example, if $lb = 5$ and $n = 4$, the added clauses are those corresponding to the formula:

$$b_3 \vee (b_2 \wedge b_1)$$

e.g.,

$$(b_3 \vee b_2) \qquad (b_3 \vee b_1).$$

With such clauses, the initial branches of $n$OPTSAT search tree corresponding to values $o \leq lb = 5$ are not explored.

## 6 Conclusions

In this paper we showed that DLL can be used to solve satisfiability problems in the presence of qualitative preferences on literals/formulas by simply imposing an ordering on the literals to be used while branching. The computation of all optimal models requires adding a formula pruning the generation of dominated models. We also showed how it is possible to reduce quantitative preferences to qualitative ones by a Boolean encoding of the objective function. We implemented our ideas in MINISAT and we showed that the resulting system compares well with other state-of-the-art systems even on MIN-ONE/MIN-ONE$_{\subseteq}$/MAX-SAT/MAX-SAT$_{\subseteq}$ problems.

There is a huge literature on qualitative and quantitative preferences, see, e.g., the proceedings of the last "Multi-disciplinary Workshop on Advances in Preference Handling" [50], or the 2007 AAAI tutorial on "Representing, Eliciting, and Reasoning with Preferences" by Ronen Brafman and Carmel Domshlak.[15] In this paper, we have deeply analyzed the relation between our proposal and other works that deal with computing "optimal" models in SAT and CSP [10, 11], and relate to papers that use our same concept of optimality [9, 17].

Considering our approach for reducing quantitative to qualitative preferences in general, and for solving MIN-ONE/MAX-SAT problems in particular, the idea of translating the objective function (via an *adder* function) into a set of clauses, to be added to the input formula, has already been introduced and discussed in, e.g., [44, 51, 52], implemented in MINISAT+ [46] and applied to, e.g., planning [53] and telecommunication feature subscription [54]. However, wrt MINISAT+, in our approach we do not run the SAT solver multiple times (one for each different value of the objective function we want to test) till the optimal value is found. Instead, we run the solver *once*, and the modification of the heuristic guarantees that the first model our solver finds is also optimal. Moreover, again in comparison to MINISAT+, which is the system whose behavior is "closer" to our, there are several other differences, i.e., $(i)$ MINISAT+ can not reuse the learned clauses from previous calls, while for us this is inherited within our approach; $(ii)$ we can compute "all" optimal solutions; and $(iii)$ we can deal with qualitative preferences. Also, MINISAT+ computes an initial bound by calling a SAT solver.

Finally, we acknowledge that our approach for handling qualitative and quantitative preferences has some drawbacks:

1. The underlying SAT solver can not be used as black-box but modifications in the branching heuristics have to be implemented.
2. In SAT, it is known that imposing an ordering on the branching heuristics can lead to significant degradations in performances (see, e.g. [55, 56]).
3. In the quantitative case, we need an *adder* formula encoding the value of the objective function, which can be of significant size.

Despite the above, we have seen that our system compares well with other state-of-the-art systems even on MIN-ONE/MAX-SAT problems. Further, in practice, significant degradations in the performances of the underlying SAT solver shows up only when

[15] Available at `http://iew3.technion.ac.il/~dcarmel/tutorial/`.

the number of preferences is very high. Finally, the modifications to the SAT solver are minimal and limited to the branching heuristics: Because of this, when a new, more efficient SAT solver will be available, it will be relatively easy to modify it thus gaining further efficiency.

## 7 Acknowledgements

The authors would like to thank Carmel Domshlak and Derek Long for useful discussions related to the topic of the paper, and the anonymous reviewers for their useful suggestions.. This work is partially supported by MIUR.

## References

1. Enrico Giunchiglia and Marco Maratea. Solving optimization problems with DLL. In *Proc. ECAI*, pages 377–381, 2006.
2. Emanuele Di Rosa, Enrico Giunchiglia, and Marco Maratea. Computing all optimal solutions in satisfiability problems with preferences. In *Proc. CP*, pages 603–607, 2008.
3. Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. ECAI*, pages 359–363, 1992.
4. Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *Proc. TACAS*, 1999.
5. Daniel Le Berre and Laurent Simon. Fifty-five solvers in vancouver: The SAT 2004 competition. In *Proc. SAT (Selected Papers)*, pages 321–344, 2004.
6. Daniel LeBerre and Laurent Simon. Preface to the special volume on the SAT 2005 competitions and evaluation. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–14, 2006.
7. Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem proving. *Communication of ACM*, 5(7):394–397, 1962.
8. Chiaki Sakama and Katsumi Inoue. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence*, 123(1-2):185–222, 2000.
9. Davy Van Nieuwenborgh, Stijn Heymans, and Dirk Vermeir. On programs with linearly ordered multiple preferences. In *Proc. ICLP*, pages 180–194, 2004.
10. Thierry Castell, Claudette Cayrol, Michel Cayrol, and Daniel Le Berre. Using the Davis and Putnam procedure for an efficient computation of preferred models. In *Proc. ECAI*, pages 350–354, 1996.
11. Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. Preference-based constrained optimization with CP-nets. *Computational Intelligence*, 20(2):137–157, 2004.
12. Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.
13. Steven David Prestwich, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Constraint-based preferential optimization. In *Proc. AAAI*, pages 461–466, 2005.
14. Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proc. SAT*, pages 502–518, 2003.
15. Enrico Giunchiglia and Marco Maratea. Planning as satisfiability with preferences. In *Proc. AAAI*, pages 987–992, 2007.

16. Alfonso Gerevini, Patrick Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the 5th IPC: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.
17. Davy Van Nieuwenborgh and Dirk Vermeir. Preferred answer sets for ordered logic programs. *Theory and Practice of Logic Programming*, 6(1-2):107–167, 2006.
18. João P. Marques-Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proc. ICCAD*, pages 220–227, 1996.
19. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT Solver. In *Proc. DAC*, pages 530–535, 2001.
20. G. S. Tseitin. On the complexity of proofs in propositional logics. *Seminars in Mathematics*, 8, 1970. Reprinted in [57].
21. David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.
22. Paul Jackson and Daniel Sheridan. Clause form conversions for Boolean circuits. In *Proc. SAT*, pages 183–198, 2004.
23. Ken L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Proc. CAV*, pages 250–264, 2002.
24. Kavita Ravi and Fabio Somenzi. Minimal assignments for bounded model checking. In *Proc. TACAS*, pages 31–45, 2004.
25. HoonSang Jin and Fabio Somenzi. Prime clauses for fast enumeration of satisfying assignments to boolean circuits. In *Proc. DAC*, pages 750–753. ACM, 2005.
26. Ken L. McMillan. *Symbolic Model Checking: an Approach to the State Explosion Problem*. Kluwer Academic Publishers, 1993.
27. Olivier Coudert. On solving covering problems. In *Proc. DAC*, pages 197–202, 1996.
28. Vasco M. Manquinho, João P. Marques Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In *Proc. SAT 2009*, pages 495–508, 2009.
29. Joost P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, 1998.
30. Simon De Givry, Javier Larrosa, Pedro Meseguer, and Thomas Schieux. Solving Max-SAT as weighted CSP. In *Proc. CP*, pages 363–376, 2003.
31. Miquel Ramirez and Hector Geffner. Structural relaxations by variable renaming and their compilation for solving MinCostSAT. In *Proc. CP*, pages 605–619, 2007.
32. Leila Amgoud, Claudette Cayrol, and Daniel Le Berre. Comparing arguments using preference ordering for argument-based reasoning. In *Proc. ICTAI*, pages 400–403, 1996.
33. Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Generic ILP versus specialized 0-1 ILP: an update. In *Proc. ICCAD*, pages 450–457, 2002.
34. Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proc. SAT*, pages 61–75, 2005.
35. David G. Mitchell. A SAT solver primer. *Bulletin of the European Association for Theoretical Computer Science*, 85:112–132, 2005.
36. Vasco M. Manquinho and João P. Marques Silva. On solving boolean optimization with satisfiability-based algorithms. In *Proc. AMAI*, 2000.
37. Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of Boolean cardinality constraints. In *Proc. CP*, pages 108–122, 2003.
38. Brian Borchers and Judith Furman. A two-phase exact algorithm for Max-SAT and weighted Max-SAT problems. *Journal of Combinatorial Optimization*, 2(4):299–306, 1998.
39. Zhao Xing and Weixiong Zhang. MaxSolver: An efficient exact algorithm for (weighted) maximum satisfiability. *Artificial Intelligence*, 164(1-2):47–80, 2005.
40. Javier Larrosa and Thomas Schiex. In the quest of the best form of local consistency for weighted CSP. In *Proc. IJCAI 2003*, pages 239–244, 2003.

41. Chu Min Li, Felip Manya, and Jordi Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.
42. Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSat: A new weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.
43. João Marques-Silva and Jordi Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Proc. DATE*, pages 408–413, 2008.
44. Peter Barth. A Davis-Putnam enumeration algorithm for linear Pseudo-Boolean optimization. Technical report, Max Plank Instutute for Computer Scince, 1995. MPI-I-95-2-2003.
45. Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. PBS: A backtrack search Pseudo-Boolean solver. In *Proc. SAT*, 2002.
46. Niklas Eén and Niklas Sörensson. Translating Pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.
47. Hossein M. Sheini and Karem A. Sakallah. Pueblo: A modern Pseudo-Boolean Sat solver. In *Proc. DATE*, pages 684–685, 2005.
48. Vasco M. Manquinho and João P. Marques-Silva. On using cutting planes in Pseudo-boolean optimization. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:209–219, 2006.
49. Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning*, 40(1):1–33, 2008.
50. Jan Chomicki, Vincent Conitzer, Ulrich Junker, and Patrice Perny, editors. *4th Multidisciplinary Workshop on Advances in Preference Handling (MPREF'08)*. AAAI Press, 2008.
51. Vasco M. Manquinho, Paulo F. Flores, João P. Marques Silva, and Arlindo L. Oliveira. Prime implicant computation using satisfiability algorithms. In *Proc. ICTAI*, pages 232–239, 1997.
52. Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Logic programming with satisfiability. *Theory and Practice of Logic Programming*, 8(1):121–128, 2008.
53. Markus Büttner and Jussi Rintanen. Satisfiability planning with constraints on the number of actions. In *Proc. ICAPS*, pages 292–299, 2005.
54. Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Telecommunications feature subscription as a partial order constraint problem. In *Proc. ICLP 2008*, pages 749–753, 2008.
55. Josh Buresh-Oppenheim and Toniann Pitassi. The complexity of resolution refinements. In *Proc. LICS*, pages 138–147, 2003.
56. Matti Järvisalo, Tommi Junttila, and Ilkka Niemelä. Unrestricted vs restricted cut in a tableau method for Boolean circuits. *Annals of Mathematics and Artificial Intelligence*, 44(4):373–399, August 2005.
57. Jörg Siekmann and Graham Wrightson, editors. *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 1-2. Springer-Verlag, 1983.