On SAT Modulo Theories and Optimization Problems

Robert Nieuwenhuis and Albert Oliveras*

Abstract. Solvers for *SAT Modulo Theories (SMT)* can nowadays handle large industrial (e.g., formal hardware and software verification) problems over theories such as the integers, arrays, or equality. Here we show that SMT approaches can also efficiently solve problems that, at first sight, do not have a typical SMT flavor. In particular, here we deal with SAT and SMT problems where models M are sought such that a given cost function $f(M)$ is minimized.

For this purpose, we introduce a variant of SMT where the theory T becomes progressively stronger, and prove it correct using the Abstract DPLL Modulo Theories framework. We discuss two different examples of applications of this SMT variant: weighted Max-SAT and weighted Max-SMT. We show how, with relatively little effort, one can obtain a competitive system that, in the case of weighted Max-SMT in the theory of Difference Logic, can even handle well-known hard radio frequency assignment problems without any tailored heuristics. These results seem to indicate that Max-SAT/SMT techniques can already be used for realistic applications.

1 Introduction

The Davis-Putnam-Logemann-Loveland (DPLL) procedure for propositional SAT [DP60, DLL62] has also been adapted for handling problems in more expressive logics, and, in particular, for the *SAT Modulo Theories (SMT)* problem: deciding the satisfiability of ground first-order formulas with respect to background theories such as the integer or real numbers, or arrays. SMT problems frequently arise in formal hardware and software verification applications, where typical formulas consist of very large sets of clauses like:

$$p \quad \vee \quad \neg q \quad \vee \quad a = b - c \quad \vee \quad \mathit{read}(v,\, b{+}c\,) = d \quad \vee \quad a{-}c \leq 7$$

with propositional atoms as well as atoms over (combined) theories like the integers, the reals, or arrays. SMT has become a very active area of research, and efficient SMT solvers exist that can handle (combinations of) many such theories (see also the SMT problem library [TR05] and the SMT Competition [BdMS05]). Currently most SMT solvers follow the so-called *lazy* approach to SMT, combining (i) *theory solvers* to process *conjunctions* of literals over the given theory T, with (ii) DPLL-based *engines* for dealing with the boolean structure of the formulas.

* Technical Univ. of Catalonia, Barcelona, `www.lsi.upc.edu/~roberto|~oliveras`. Partially supported by Spanish Ministry of Educ. and Science through the LogicTools project (TIN2004-03382, both authors), and FPU grant AP2002-3533 (Oliveras).

A. Biere and C.P. Gomes (Eds.): SAT 2006, LNCS 4121, pp. 156–169, 2006.

DPLL(T) is a general SMT architecture for the lazy approach [GHN+04]. It consists of a DPLL(X) engine, whose parameter X can be instantiated with a T-solver $Solver_T$, thus producing a DPLL(T) system. The DPLL(X) engine always considers the problem as a purely propositional one. For example, if the theory T is the integers, at some point DPLL(X) might consider a partial assignment containing, among many others, the three literals $a \geq b+3$, $b-2 \geq c$, and $a \not> c$ without noticing its T-inconsistency, because it just considers such literals as propositional (syntactic) objects. But $Solver_T$ continuously analyzes the partial model that DPLL(X) is building (a conjunction of literals). It can warn DPLL(X) about this T-inconsistency, and generate a clause, called a *theory lemma*, $a \not\geq b+3 \ \vee \ b-2 \not\geq c \ \vee \ a > c$ that can be used by DPLL(X) for backjumping. $Solver_T$ sometimes also does *theory propagation*: as soon as, e.g., $a \geq b+3$ and $b-2 \geq c$ become true, it can notify DPLL(X) about T-consequences like $a > c$ that occur in the input formula. The modular DPLL(T) architecture is flexible, and, compared with other SMT techniques, DPLL(T) is also very efficient and has good scaling properties: the BarcelogicTools implementation of DPLL(T) won all the four divisions it entered at the 2005 SMT Competition [BdMS05].

The aim of this paper is to show that SMT techniques such as DPLL(T) can be easily adapted to efficiently solve problems that, at first sight, do not have a typical SMT flavor. In particular, here we deal with SAT and SMT problems where models M are sought such that a given cost function $f(M)$ is minimized.

For this purpose, in Section 2 we introduce a variant of SMT where the (first-order) theory T becomes *progressively stronger*, that is, T may be periodically replaced by $T \wedge T'$ for some first-order theory T'. Then, after mentioning some applications to optimization and other problems, we prove this variant correct by extending *Abstract DPLL Modulo Theories*, a uniform, declarative framework introduced in [NOT05] for modeling and reasoning about lazy SMT procedures.

In Section 3 we apply this SMT variant in a branch-and-bound setting, where the theory T "knows", possibly among the information about other theories, the cost function f and its current best bound. Each time a better bound is found, the SMT procedure continues with a theory that has become stronger, in the sense that models with a cost higher than this new bound now become T-inconsistent.

We then show how to deal in this framework with the *exact weighted Max-SAT* problem: given a set of pairs $\{(C_1, w_1) \ldots, (C_m, w_m)\}$ where each C_i is a propositional clause and w_i is its *weight* (a positive natural or real number), find a propositional assignment M that minimizes the sum of the weights of clauses that are false in M.

In Section 4 we report experimental results on an implementation of DPLL(T) for Max-SAT and also explain how specialized propagation rules for Max-SAT in the style of [LH05] can be easily and flexibly incorporated. For instance, when two pairs of the form (l, w_1) and $(\neg l, w_2)$ appear, one can propagate $min\{w_1, w_2\}$.

An interesting aspect of this approach is that DPLL(T) allows one to obtain with relatively little effort Max-SAT implementations that are competitive w.r.t. state-of-the-art systems. We give experimental results that show this, for settings with and without the additional propagation rules.

In Section 5 we show how this approach can be smoothly extended to Max-SMT. As an example, we deal with *weighted Max-SMT* modulo the theory of Integer Difference Logic, a fragment of integer linear arithmetic. In this case, formulas are built over propositional atoms, as well as (ground) atoms of the form $a - b \leq k$, where a and b are (Skolemized) integer variables and k is an integer.

This logic is used in the context of hardware and software verification; for instance, some properties of timed automata are naturally expressed in it. But again, also problems that do not look a priori like typical SMT problems can be handled very efficiently with it, and also optimization problems can be solved using our approach.

We give experimental results on the well-known hard CELAR radio frequency assignment problems [CdGL$^+$99]. In these problems, integer variables must belong to certain intervals, and constraints express minimal distances between variables, all of which can be very nicely modeled in Difference Logic.

From our BarcelogicTools DPLL(T) implementation we have obtained, with very little effort, our first Max-SMT system. In spite of its unlabored development, and of its single standard SMT decision heuristic, our experiments reveal that it can already handle these CELAR problems that, according to our experiments, appear to be far beyond the capabilities of systems dealing with translations into, e.g., Weighted Max-SAT, pseudo-Boolean, or Integer Linear Programming Problems. On the CELAR problems, this implementation importantly outperforms the best-known weighted CSP solver Toolbar [dGHZL05] in its default settings, and is still close or superior to Toolbar with its best (according to its authors) branching heuristic for these problems.

2 SMT with Progressively Stronger Theories

Abstract DPLL Modulo Theories [NOT05] is a framework for modeling and reasoning about DPLL-based SAT and SMT systems in terms of simple transition rules and rule application strategies. The framework eases the understanding and the comparison of different approaches as well as the proving of their correctness. In this section, we briefly describe the framework (see [NOT05] for details) and then extend it to accommodate *progressively stronger* theories, that is, the theory T may be periodically replaced by $T \wedge T'$ for some first-order theory T', and prove the correctness of this extension.

2.1 Potential Applications

Such a SAT or SMT procedure where the theory becomes progressively stronger has applications in the context of branch-and-bound-like applications, where a

single model is sought that minimizes a given cost function. The weighted Max-SAT and Max-SMT problems addressed in the next sections are just a particular case of this; one could also handle problems like Max-Ones, or even problems with non-linear cost functions.

But, apart from optimization problems, it can also be useful for problems where, given a set of pairwise T-incompatible (first-order) properties $P_1 \dots P_n$, different models $M_1 \dots M_n$ are sought such that each M_i satisfies the corresponding P_i. Initially, one can add $P_1 \vee \dots \vee P_n$ to the theory. Then, each time such an M_i is found, before backjumping to continue the search one can strengthen the theory T, replacing it by $T \wedge \neg P_i$, which can help pruning the search of models for the remaining P_j's.

A more concrete problem of this kind is, for instance, a company where every month the best employee is rewarded with a favorable working schedule for the following month. Hence the company needs to prepare in advance n schedules with properties $P_1 \dots P_n$, where each P_i expresses that employee i is the one that works less hours, (or works least night shifts, or gets most money, etc).

Another completely different application is automatic classification of finite algebras [CMSM04], where one may be searching for, say, finite groups satisfying a set $P_1 \dots P_n$ of different properties, one group for each P_i.

We stress that this is particularly useful if the properties $P_1 \dots P_n$ cannot (efficiently) be expressed at the level of the SMT formula itself, and if $Solver_T$ can adequately handle their negations.

2.2 Abstract DPLL Modulo Theories

As usual in SMT, given a background theory T (a set of closed first-order formulas), we will only consider the SMT problem for *ground* (and hence quantifier-free) CNF formulas F. Such formulas may contain *free* constants, i.e., constant symbols not in the signature of T, which, as far as satisfiability is concerned, can be equivalently seen as existential variables. Other than free constants, all other predicate and function symbols in the formulas will instead come from the signature of T. From now on, we will assume that all formulas satisfy these restrictions.

The formalism we describe is based on a set of *states* together with a binary relation $\Longrightarrow$ (called the *transition relation*) over these states, defined by means of *transition rules*. Starting with a state containing an input formula F, one can use the rules to generate a finite sequence of states, where the final state indicates, for a certain theory T, whether or not F is T-consistent.

A *state* is either the distinguished state $T \parallel \mathit{fail}$ (denoting T-unsatisfiability) or a triple of the form $T \parallel M \parallel F$, where T is a theory, M is a sequence of literals, and F is a formula in conjunctive normal form (CNF), i.e., a finite set of disjunctions of literals. We additionally require that M never contains both a literal and its negation and that each literal in M is annotated as either a *decision* literal (indicated by l^{d}) or not. Frequently, we will refer to M as a

partial assignment or consider M just as a set or conjunction of literals, ignoring both the annotations and the order of its elements. We use $\emptyset$ to denote the empty sequence.

In what follows, a possibly subscripted or primed lowercase l *always* denotes a literal. Similarly T and T' always denote theories, C and D always denote clauses (disjunctions of literals), F and G denote conjunctions of clauses, and M and N denote assignments.

We write $M \models F$ to indicate that M propositionally satisfies F. If C is a clause $l_1 \vee \ldots \vee l_n$, we sometimes write $\neg C$ to denote the formula $\neg l_1 \wedge \ldots \wedge \neg l_n$. We say that C is *conflicting* in a state $T \parallel M \parallel F, C$ if $M \models \neg C$.

A formula F is called T*-(in)consistent* if $F \wedge T$ is (un)satisfiable in the first-order sense. We say that M *is a* T*-model of* F if $M \models F$ and M, seen as a conjunction of literals, is T-consistent. It is not difficult to see that F is T-consistent if, and only if, it has a T-model. If F and G are formulas, then F *entails* G *in* T, written $F \models_T G$, if $F \wedge \neg G$ is T-inconsistent. If $F \models_T G$ and $G \models_T F$, we say that F and G are T*-equivalent.*

We start presenting a small variant, to accomodate the presence of the theory in the states, of the transition system first presented in [NOT05]:

Definition 1. *The* Basic DPLL Modulo Theories system *consists of the following five rules:*

UnitPropagate :

$$T \parallel M \parallel F, C \vee l \implies T \parallel M\ l \parallel F, C \vee l \quad \textbf{if} \begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases}$$

Decide :

$$T \parallel M \parallel F \implies T \parallel M\ l^{\mathsf{d}} \parallel F \quad \textbf{if} \begin{cases} l \text{ or } \neg l \text{ occurs in a clause of } F \\ l \text{ is undefined in } M \end{cases}$$

Fail :

$$T \parallel M \parallel F, C \implies T \parallel \mathit{fail} \quad \textbf{if} \begin{cases} M \models \neg C \\ M \text{ contains no decision literals} \end{cases}$$

Backjump :

$$T \parallel M\ l^{\mathsf{d}}\ N \parallel F, C \implies T \parallel M\ l' \parallel F, C \quad \textbf{if} \begin{cases} M\ l^{\mathsf{d}}\ N \models \neg C, \text{ and there is} \\ \text{some clause } C' \vee l' \text{ such that:} \\ F, C \models_T C' \vee l' \text{ and } M \models \neg C', \\ l' \text{ is undefined in } M, \text{ and} \\ l' \text{ or } \neg l' \text{ is in } F \text{ or in } M\ l^{\mathsf{d}}\ N \end{cases}$$

Theory Propagate :

$$T \parallel M \parallel F \implies T \parallel M\ l \parallel F \quad \textbf{if} \begin{cases} M \models_T l \\ l \text{ or } \neg l \text{ occurs in } F \\ l \text{ is undefined in } M \end{cases}$$

Definition 2. *The* Strengthening DPLL Modulo Theories system *consists of the five Basic DPLL Modulo Theories rules, and the following four rules:*

Restart :

$$T \parallel M \parallel F \quad \Longrightarrow \quad T \parallel \emptyset \parallel F$$

Theory Learn :

$$T \parallel M \parallel F \quad \Longrightarrow \quad T \parallel M \parallel F, C \qquad \textbf{if} \begin{cases} \text{each atom of } C \text{ is in } F \text{ or in } M \\ F \models_T C \end{cases}$$

Theory Forget :

$$T \parallel M \parallel F, C \quad \Longrightarrow \quad T \parallel M \parallel F \qquad \textbf{if} \begin{cases} F \models_T C \end{cases}$$

Theory Strengthen :

$$T \parallel M \parallel F \quad \Longrightarrow \quad T \wedge T' \parallel M \parallel F$$

We denote the transition relation defined by all nine rules by $\Longrightarrow_S$.

For a transition relation $\Longrightarrow$, we denote by $\Longrightarrow^*$ the reflexive-transitive closure of $\Longrightarrow$. We call any sequence of the form $S_0 \Longrightarrow S_1, \ S_1 \Longrightarrow S_2, \ \ldots$ a *derivation*, and denote it by $S_0 \Longrightarrow S_1 \Longrightarrow S_2 \Longrightarrow \ldots$. We call any subsequence of a derivation a *subderivation*. If $S \Longrightarrow S'$ we say that there is a *transition* from S to S'. A state S is *final* with respect to $\Longrightarrow$ if there are no transitions from S.

2.3 Correctness of Strengthening Abstract DPLL Modulo Theories

A decision procedure for SMT can be obtained by generating a derivation using $\Longrightarrow_S$ with a particular strategy. The relevant derivations are those that start with a state of the form $T_0 \parallel \emptyset \parallel F_0$, where F_0 is the initial formula. The aim of a derivation is to compute a state S such that: (i) S is final with respect to the five rules of Basic DPLL Modulo Theories and (ii) if S is of the form $T \parallel M \parallel F$ then M is T-consistent. We start by stating some invariants.

Lemma 3. *If* $T_0 \parallel \emptyset \parallel F_0 \Longrightarrow_S^* T \parallel M \parallel F$ *then the following hold.*

1. *All the atoms in* M *and all the atoms in* F *are atoms of* F_0.
2. M *contains no literal more than once and is indeed an assignment, i.e., it contains no pair of literals of the form* l *and* $\neg l$.
3. F *and* F_0 *are* T*-equivalent.*
4. *If* M *is of the form* $M_0\ l_1\ M_1\ \ldots\ l_n\ M_n$, *where* $l_1, \ldots, l_n$ *are all the decision literals of* M, *then* $F_0, l_1, \ldots, l_i \models_T M_i$ *for all* i *in* $0 \ldots n$.

The following termination result says that derivations are finite provided some standard conditions are fulfilled (e.g., Restart is applied with increasing periodicity), and that the theory is not strengthened infinitely many times, which is indeed the case in branch and bound and the other mentioned applications. The proof is a simple extension of the one for the standard conditions [NOT05]. This is also the case for the other proofs we omit here for space reasons.

Theorem 4 (Termination of $\Longrightarrow_S$). *Every derivation Der of the form $T \parallel \emptyset \parallel F = S_0 \Longrightarrow_S S_1 \Longrightarrow_S \dots$ is finite if the following conditions hold:*

1. *Der has no infinite subderivations consisting of only* Theory Learn *and* Theory Forget *steps.*
2. Theory Strengthen *is applied only finitely many times.*
3. *For every subderivation of Der of the form $S_{i-1} \Longrightarrow_S S_i \Longrightarrow_S \dots \Longrightarrow_S S_j \Longrightarrow_S \dots \Longrightarrow_S S_k$ where the only three* Restart *steps are the ones producing S_i, S_j, and S_k, either:*
   - *there are more Basic DPLL Modulo Theories steps in $S_j \Longrightarrow_S \dots \Longrightarrow_S S_k$ than in $S_i \Longrightarrow_S \dots \Longrightarrow_S S_j$, or*
   - *in $S_j \Longrightarrow_S \dots \Longrightarrow_S S_k$ a new clause is learned that is not forgotten in Der.*

Lemma 5. *If $T_0 \parallel \emptyset \parallel F_0 \Longrightarrow_S^* T \parallel M \parallel F$ and there is some conflicting clause in $T \parallel M \parallel F$, i.e., $M \models \neg C$ for some clause C in F, then either* Fail *or* Backjump *applies to $T \parallel M \parallel F$.*

Property 6. If $T_0 \parallel \emptyset \parallel F_0 \Longrightarrow_S^* T \parallel M \parallel F$ and M is T-inconsistent, then either there is a conflicting clause in $T \parallel M \parallel F$, or else Theory Learn applies to $T \parallel M \parallel F$, generating a conflicting clause.

Even if it is very easy to generate non-terminating derivations for $\Longrightarrow_S$, Theorem 4 defines a very general strategy for avoiding that.

Lemma 5 and Property 6 show that, for a state of the form $T \parallel M \parallel F$, if there is some literal of F undefined in M, or there is some conflicting clause, or M is T-inconsistent, then a rule of Basic DPLL Modulo Theories is always applicable, possibly after a single Theory Learn step. Together with Theorem 4 (Termination), this shows how to compute a state to which the following main theorem is applicable.

Theorem 7. *Let Der be a derivation $T_0 \parallel \emptyset \parallel F_0 \Longrightarrow_S^* S$, where S is (i) final with respect to Basic DPLL Modulo Theories, and (ii) if S is of the form $T \parallel M \parallel F$ then M is T-consistent. Then*

1. *S is $T \parallel$ fail if, and only if, F is T-inconsistent.*
2. *If S is of the form $T \parallel M \parallel F$ then M is a T-model of F.*

These results are easy to apply. For example, in the context of branch and bound, each time a final state $T \parallel M \parallel F$ is obtained (final in the sense of conditions (i) and (ii) of Theorem 7), M is the current best model found. After that, one can apply Theory Strengthen to decrease the current upper bound and make M inconsistent with the strengthened theory that says that an M with smaller cost is needed (see the next section). By property 6, this will trigger further rule applications. When no smaller cost solution exists, the theorem implies that $T \parallel$ *fail* will be eventually obtained.

Similarly, when different models $M_1 \dots M_n$ are sought satisfying properties $P_1 \dots P_n$, one can initially add $P_1 \vee \dots \vee P_n$ to the theory. Then, each time one

M_i is found for a P_i, the theory is strengthened by adding $\neg P_i$ to it, and again by property 6 the derivation continues. Once all M_i have been found, instead of adding the last $\neg P_i$ (which would make the theory inconsistent), the process is stopped.

3 Expressing Max-SAT and Max-SMT in This Framework

Here we apply this SMT variant in a branch-and-bound setting, where, given a cost function f, a model M is sought with minimum $f(M)$. In this case, the progressively stronger theory T "knows", possibly among the information about other theories, the cost function f and its current best upper bound.

In particular, here we consider the *exact weighted Max-SAT or Max-SMT* problem: given a set of pairs $\{(C_1, w_1) \ldots, (C_m, w_m)\}$ where each C_i is a (propositional or SMT) clause and w_i is its *weight* (a positive natural or real number), find an assignment M (consistent with the initial background theory T) that minimizes the sum of the weights of clauses that are false in M.

We use the well-known encoding where each weighted clause (C_i, w_i) gets a distinct additional positive literal p_i, i.e., it becomes $C_i \vee p_i$, where p_i is a fresh propositional symbol.

Given this encoding, apart from the initial background theory T (which is empty in the propositional case), the theory consists of the integers plus

$$\begin{array}{ll} p_1 \rightarrow (k_1 = w_1) & \neg p_1 \rightarrow (k_1 = 0) \\ \ldots & \ldots \\ p_m \rightarrow (k_m = w_m) & \neg p_m \rightarrow (k_m = 0) \end{array} \qquad k_1 + \cdots + k_m \leq B$$

In addition, we will have an initial cost bound B_0, and the relation $B < B_0$ will also be part of the theory. Then, each time the theory is strengthened with a new upper bound B_i, the relation $B < B_i$ is added.

Note that one can also express that a certain (disjoint) subset of the clauses must be true with a single common weight w_i, by simply adding the same p_i to all clauses in the subset.

Also note that initially each clause contains at most one weight literal p_i, but during the search these literals receive the same treatment as any other literal. Hence, due to conflict-driven learning, clauses with many (positive and negative) occurrences of such weight literals appear. The truth value of such weight literals can be set by theory propagation, since $Solver_T$ may communicate DPLL(X) that a certain p_i must be false in order not to exceed the current best upper bound for the function cost f.

4 Experiments with Max-SAT and Further Pruning Rules

In this section we give experimental results for propositional Max-SAT, showing that a competitive (wrt. pseudo-Boolean solvers) DPLL(T) system can be

obtained with relatively little effort. Moreover, we also discuss how specialized propagation rules can be incorporated into a DPLL(T) implementation.

4.1 Comparison with Other Approaches

Existing specific algorithms for Max-SAT (e.g., [LMP05, LH05, XZ05]) have been mainly designed to attack relatively small but challenging problems. To solve these problems, they use several pruning techniques that detect when a certain partial assignment cannot be extended to a complete assignment that improves the current upper bound. However, on larger examples such as most of the ones analyzed here, these techniques (and their implementations) become extremely time and memory consuming, and hence here we only give experimental results for one of them, namely Toolbar [LH05].

Another possibility for attacking Max-SAT problems is to use pseudo-Boolean solvers [ARMS02, SS06, ES06]. The encoding for Max-SAT presented in Section 3 can be easily adapted to convert the clauses, including the additional weight literals, into pseudo-Boolean constraints. Then, the objective function the pseudo-Boolean solver has to minimize, subject to the pseudo-Boolean constraints, is $w_1 * p_1 + \ldots + w_m * p_m$. Since these solvers are designed to deal with large input pseudo-Boolean problems, they do not incorporate any ad-hoc technique for Max-SAT, which makes them inefficient on the previously mentioned small challenging problems, but competitive on problems whose difficulty is essentially due to its size.

As we will see in the experiments below, the DPLL(T) system we propose here is competitive with the pseudo-Boolean solvers (in fact, it is usually faster), but in addition, due to its modular architecture, it is easy to develop. Once a DPLL(T) system for SMT has been constructed, almost no additional work has to be done to convert it into a tool for Max-SAT. The DPLL(X) engine already incorporates all the necessary machinery and the only thing needed is to implement $Solver_T$ for the theory described in Section 3, something doable in less than 200 lines of C code. It can also easily be adapted in order to incorporate additional pruning rules.

4.2 Additional Pruning Rules

The resulting DPLL(T) system can be further improved by providing it with specialized deduction rules for Max-SAT.

Example 8. If, due to conflict-driven lemma learning, DPLL(X) learns a clause consisting only of positive-weight literals $p_1 \vee \ldots \vee p_n$, each p_i with its associated weight w_i, one can immediately add $min\{w_1 \ldots w_n\}$ to the current cost of the assignment.

Some more complicated resolution-like rules were studied and shown to be very effective in [LH05]. Hence, we have investigated up to what extent such specialized resolution rules can be incorporated into our architecture.

Example 9. Assume our input formula contains, among many others, the binary clauses $l \vee p_1$ and $\neg l \vee p_2$, where each p_i is a literal of weight w_i. Since any total assignment will contain either l or $\neg l$, it is easy to observe that any model will have a cost of at least $min\{w_1, w_2\}$.

All the other rules presented in [LH05] are similar forms of resolution. In order to implement them, one should detect binary or ternary clauses to which resolution is applicable, and for greater effectivity, this should be done on the fly, not only as a preprocessing step. This is quite expensive in general for DPLL-based systems using the two-watched literal scheme [MMZ$^+$01]. In this scheme, one can detect newly generated unit clauses by only watching two literals in each clause, but the detection of binary clauses would require to watch three literals per clause. However, special situations like the one in Example 9 are still detectable with only two watched literals per clause if one watches a positive literal weight p only if there is no other possibility. This restriction ensures that, as soon as such a literal p becomes watched, we have found a binary clause of the form $l \vee p$.

With this small modification, DPLL(X) can efficiently detect the presence of binary clauses of the form $l \vee p_1$ and $\neg l \vee p_2$ and then notify to $Solver_T$ an increment in the cost of the current assignment of $m{=}min\{w_1, w_2\}$, thus allowing $Solver_T$ to further prune partial assignments that have no possibility to improve the current upper bound. Since part of the weights w_1 and w_2 has already been amortized in the cost of the assignment, $Solver_T$ also has to be notified that the weights of p_1 and p_2 now become $w_1 - m$ and $w_2 - m$, respectively.

4.3 Experimental Evaluation

Experiments have been done on several well-known already existing benchmark families. The DIMACS suite consists of unsatisfiable propositional formulas with a weight of 1 for each clause, similarly to the Weighted DIMACS family, with random weights between 1 and 1000 for each clause [dGLMS03]. Finally, the Quasi-group instances[1] encode quasi-group completion problems in which the clauses enforcing the quasi-group structure and that some cells must contain a given element have been given a certain weight.

We compare with three other systems: Toolbar [dGHZL05], a weighted-CSP solver which incorporates specialized algorithms and data structures for the Max-SAT problem; Pueblo [SS06], a pseudo-Boolean solver implementing a branch-and-bound approach to minimize a given goal function; and Minisat+ [ES06], a pseudo-Boolean solver based on translations to propositional satisfiability. This is by no means an exhaustive comparison with all available tools. We chose three tools that –we believe– represent the state of the art in these three different approaches, and that can handle problems of a reasonable size.

We ran our system in two settings: the basic one (General DPLL(T) in the table) and one implementing the specialized deduction rule mentioned in Example 9 (Special DPLL(T)). In none of them specialized heuristics were developed.

[1] We thank Felip Manyà for providing us with them.

We used our standard branching heuristic for solving general SMT problems, an extension of VSIDS [MMZ+01].

Results are in seconds and aggregated per family of benchmarks. Each benchmark was run on a 2GHz 512MB Pentium 4 for 10 minutes, i.e. 600 seconds. An annotation of the form (n t) indicates that the system timed out in n benchmarks. Each timeout is counted as 600s in the table.

Benchmark family	#pblms.	Toolbar	Minisat+	Pueblo	General DPLL(T)	Special. DPLL(T)
DIMACS:						
aim	24	(15 t) 9397	**0.5**	**0.5**	**0.5**	**0.5**
bridge fault	4	(4 t) 2400	220	73	**0.3**	0.6
dubois	13	(9 t) 6442	**0.4**	1	0.9	0.97
hole	5	(1 t) 795	102	**57**	470	(1 t) 605
jnh	30	**587**	(2 t) 5433	(3 t) 3798	(1 t)1485	948
pret	8	(4 t) 2782	0.38	0.4	0.28	**0.27**
ssa	4	(4 t) 2400	(1 t) 626	**(1 t) 601**	**(1t)601**	**(1 t) 601**
Weighted DIMACS:						
wjnh	30	105	(19 t) 14025	1415	**42.6**	54.3
Quasi-groups:						
Size 6	100	(5 t) 4194	16	2.3	**1.2**	3.5
Size 7	100	(69 t) 46202	178	6.4	**2.9**	13
Size 8	100	(100 t) 60000	582	17	**9.1**	41
Size 9	100	(100 t) 60000	1599	47	**29**	157
Size 10	100	(100 t) 60000	4783	203	**151**	668

These experiments only have a relative value, given the fact that tools such as Minisat+ or Pueblo are designed to attack the broader class of pseudo-boolean optimization problems, and also because Toolbar's specialized Max-SAT algorithms are more tailored towards small but challenging Max-SAT problems.

One can also observe from the results that the integration of specialized deduction rules was not always successful. In some benchmarks, like Quasi-groups, the rule we implemented did never apply. In this case one pays a non-negligible overhead (the system gets as much as 5 times slower) without obtaining any gain. On the other hand, in the benchmarks where it was more productive (e.g. in the jnh family, where on average at least one rule application was possible out of every 10 decisions) the overhead was compensated by a reduction in the search space. We believe it is still unclear whether to pursue the integration of such rules is worthwile. A more careful analysis should be made on more realistic benchmarks, coming from real applications where weights are not assigned at random.

5 Max-SMT: The Example of Difference Logic

We now show how this approach can be smoothly extended to Max-SMT. We focus here on the case of *weighted Max-SMT* modulo the theory of Integer Difference Logic, a fragment of integer linear arithmetic, but our approach is not limited to this theory.

In Integer Difference Logic, formulas are built over propositional atoms, as well as (ground) atoms of the form $a - b \leq k$, where a and b are (Skolemized)

integer variables and k is an integer[2]. This logic is used in the context of hardware and software verification; for instance, some properties of timed automata are naturally expressed in it. But here we will show how a problem that a priory doesn't look well-suited for Difference Logic can be encoded in it, and how also optimization problems can be solved using our approach.

5.1 Encoding the CELAR Problems in Integer Difference Logic

The well-known CELAR Radio Link Frequency Assignment problems [CdGL$^+$99] consist of, given a set of radio links and a set of radio frequencies, assigning a frequency to each radio link. For some pairs of radio links, their frequencies must be at a certain exact distance, and for others, they must be at least at a certain distance. The latter constraints are soft, i.e., each one of them has an associated cost if it is not satisfied, and the solution with minimum total cost has to be found. Here we shortly explain how we encode these problems as a Max-SMT problem for Integer Difference Logic.

Each radio link l_i has a finite set of available frequencies D_i that can always be seen as the disjoint union of four sets. For example:

$$\{2+14k \mid 1 \leq k \leq 11\} \qquad \{2+14k \mid 18 \leq k \leq 28\}$$
$$\{8+14k \mid 29 \leq k \leq 39\} \qquad \{8+14k \mid 46 \leq k \leq 56\}$$

This observation is crucial to express in a compact manner that the frequency f_i for the radio link l_i has to be in D_i. For that purpose, we will encode the value of f_i using two variables: a propositional variable t_i expressing whether f_i is 2 modulo 14 or not, and an integer variable m_i, representing f_i *mod* 14. With these additional variables we can express that f_i is in D_i with the formulas:

$$t_i \rightarrow (1 \leq m_i \leq 11 \ \vee \ 18 \leq m_i \leq 28)$$
$$\neg t_i \rightarrow (29 \leq m_i \leq 39 \ \vee \ 46 \leq m_i \leq 56)$$

It now remains to encode distance constraints of the form $|f_i - f_j| > k$, with their costs w_{ij}. Our encoding of f_i and f_j using the auxiliary variables makes it natural to reason by cases, depending on whether f_i and f_j are 2 modulo 14 or not. For example if f_i is 2 modulo 14 and f_j is not (and hence is 8 modulo 14), then $|f_i - f_j| > k$ is equivalent to $|2+14m_i-8-14m_j| > k$. After the necessary manipulations, the corresponding Difference Logic clause is:

$$(t_i \wedge \neg t_j) \rightarrow \left(m_i - m_j \geq \left\lfloor \frac{k+6}{14} \right\rfloor + 1 \ \vee \ m_i - m_j \leq \left\lceil \frac{-k+6}{14} \right\rceil - 1 \right)$$

The exact distance constraints are encoded similarly. Costs are expressed by additional weight literals as explained in Section 3.

[2] Atoms of the form $a \leq k$ are also allowed because one can use an auxiliary integer variable z_0, and consider the inequality $a - z_0 \leq k$ instead. It is not difficult to see that this transformation preserves T-satisfiability. See [NO05] for details on our DPLL(T) system for Difference Logic.

5.2 Experimental Evaluation

As before, from our BarcelogicTools DPLL(T) implementation we obtained, with little effort, our first Max-SMT system.

We compared our system with the best-known weighted CSP solver Toolbar [dGHZL05] in three different settings. The first one used a static branching heuristic, the second one was the default setting for Toolbar and the third one, the best possible choice according to the authors, used a Jeroslow-like branching heuristic. We used the same machine as before and again results are in seconds.

Benchmark name	Toolbar			DPLL(T)
	Static	Default	Jeroslow	
SUBCELAR_6_0	**0.2**	0.4	**0.2**	5
SUBCELAR_6_1	85	252	**65**	90
SUBCELAR_6_2	127	982	**25**	132
SUBCELAR_6_3	7249	2169	**355**	636
SUBCELAR_6_4	7021	> 5 hours	1942	**1417**

The choice of different branching heuristics leads to dramatic changes in Toolbar's runtime. Hence, we believe that specialized heuristics for DPLL(T) could still improve its performance. From these limited results, one cannot infer that DPLL(T) has better scaling properties than Toolbar in its best setting, although it is 20 times slower on the smallest problem, but faster on the largest one. In any case, we believe these results indicate that SMT tools can be already used for efficiently solving industrial optimization problems.

We also carefully translated these problems into weighted Max-SAT and pseudo-Boolean problems. Somewhat to our surprise, the tools mentioned in Section 4 needed around 30 seconds on the smallest of these problems, and did not terminate in a day on the second smallest one. These problems are also not known to be tractable by means of translations into pure Integer Linear Programming, in spite of attempts using the best ILP solvers (Javier Larrosa, private communication).

6 Conclusions and Further Work

By developing DPLL(T) techniques for weighted Max-SAT, we have shown that DPLL(T) can be very competitive for problems that do not look a priori like typical SMT problems. Since this was achieved with relatively little effort, we see this as an indication of the quality, in terms of efficiency and flexibility, of our approach.

The success of our Max-SMT implementation on the CELAR benchmarks reveals that realistic problems can be modeled as Max-SMT problems and solved with small variants of SMT solvers. Effectivity of SMT solvers for that purpose has also been recently shown in [SPSP05], using SMT over another fragment of linear arithmetic, for solving soft temporal constraints, where extensive experiments were done on random problems.

Future work concerns other problems that are not typical SMT-like. For example, we are currently investigating the use of DPLL(T) for expressing

finite-domain constraints, and in particular global constraints from the constraint programming world, such as `alldifferent`.

References

[ARMS02] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah. PBS: A backtrack-search pseudo-boolean solver and optimizer. In *SAT'02*, LNCS, pp. 346–353.

[BdMS05] C. Barrett, L. de Moura, and A. Stump. SMT-COMP: Satisfiability Modulo Theories Competition. In *CAV'05*, LNCS 3576, pp. 20–23. http://www.csl.sri.com/users/demoura/smt-comp/

[CdGL+99] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners. Radio link frequency assignment. *Constraints*, 4(1):79–89, 1999.

[CMSM04] S. Colton, A. Meier, V. Sorge, and R. McCasland. Automatic generation of classification theorems for finite algebras. In *IJCAR'04*, LNCS 3097, pp. 400–414.

[dGHZL05] S. de Givry, F. Heras, M. Zytnicki, and J. Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In *IJCAI'05*, pp. 84–89, 2005.

[dGLMS03] Simon de Givry, Javier Larrosa, Pedro Meseguer, and Thomas Schiex. Solving max-sat as weighted csp. In *CP'03*, LNCS 2833, 363–376, 2003.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Comm. of the ACM*, 5(7):394–397, 1962.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[ES06] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[GHN+04] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast Decision Procedures. In *CAV'04*, LNCS 3114, 175–188.

[LH05] J. Larrosa and F. Heras. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In *IJCAI'05*, pp. 193–198, 2005.

[LMP05] C. Li, F. Manyà, and J. Planes. Solving Over-Constrained Problems with SAT. In *CP'05*, LNCS 3709, pp. 403–414.

[MMZ+01] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *DAC'01*, 2001.

[NO05] R. Nieuwenhuis and A. Oliveras. DPLL(T) with Exhaustive Theory Propagation and its Application to Difference Logic. In *CAV'05* LNCS 3576, pp. 321–334.

[NOT05] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Abstract DPLL and Abstract DPLL Modulo Theories. In *LPAR'04*, LNAI 3452, pp. 36–50.

[SPSP05] H. Sheini, B. Peintner, K. Sakallah, and M. Pollack. On solving soft temporal constraints using SAT techniques. In *CP'05*, LNCS 3709, pp. 607–621.

[SS06] H. M. Sheini and K. A. Sakallah. Pueblo: A hybrid pseudo-boolean SAT solver. *J. Satisfiability, Boolean Modeling and Comp.*, 2:165–189, 2006.

[TR05] C. Tinelli and S. Ranise. SMT-LIB: The Satisfiability Modulo Theories Library, July 2005. http://goedel.cs.uiowa.edu/smtlib/.

[XZ05] Z. Xing and W. Zhang. Maxsolver: an efficient exact algorithm for (weighted) maximum satisfiability. *Artif. Intell.*, 164(1-2):47–80, 2005.