Branch and Bound for Boolean Optimization and the Generation of Optimality Certificates

Javier Larrosa, Robert Nieuwenhuis,
Albert Oliveras, Enric Rodríguez-Carbonell*

Abstract. We consider optimization problems of the form $(S, cost)$, where S is a clause set over Boolean variables $x_1 \dots x_n$, with an arbitrary cost function $cost\colon \mathbb{B}^n \to \mathbb{R}$, and the aim is to find a model A of S such that $cost(A)$ is minimized.
Here we study the generation of *proofs of optimality* in the context of branch-and-bound procedures for such problems. For this purpose we introduce $\mathtt{DPLL}_{\mathtt{BB}}$, an abstract DPLL-based branch and bound algorithm that can model optimization concepts such as cost-based propagation and cost-based backjumping.
Most, if not all, SAT-related optimization problems are in the scope of $\mathtt{DPLL}_{\mathtt{BB}}$. Since many of the existing approaches for solving these problems can be seen as instances, $\mathtt{DPLL}_{\mathtt{BB}}$ allows one to formally reason about them in a simple way and exploit the enhancements of $\mathtt{DPLL}_{\mathtt{BB}}$ given here, in particular its uniform method for generating independently verifiable optimality proofs.

1 Introduction

An important issue on algorithms for Boolean satisfiability is their ability to provide proofs of unsatisfiability, so that also negative answers can be verified with a trusted independent proof checker. Many current SAT solvers provide this feature typically by writing (with little overhead) a trace file from which a resolution proof can be reconstructed and checked.

In this paper we address a related topic. We take a very general class of Boolean optimization problems and consider the problem of computing the best model of a CNF with respect to a cost function and, additionally, a proof of its optimality. The purpose of the paper is to provide a general solving framework that is faithful to state-of-the-art branch-and-bound solvers and where it is simple to reason about them and to generate optimality proofs. We show how branch-and-bound algorithms can provide proofs with little overhead, as in the SAT case. To the best of our knowledge, no existing solvers offer this feature.

The first contribution of the paper is an abstract DPLL-like branch-and-bound algorithm ($\mathtt{DPLL}_{\mathtt{BB}}$) that can deal with most, if not all, Boolean optimization problems considered in the literature. $\mathtt{DPLL}_{\mathtt{BB}}$ is based on standard abstract DPLL rules and includes features such as propagation, backjumping, learning

* All authors from Technical Univ. of Catalonia, Barcelona, and partially supported by Spanish Min. of Science and Innovation through the projects TIN2006-15387-C03-0 and TIN2007-68093-C02-01 (LogicTools-2).

or restarts. The essential difference between classical DPLL and its branch-and-bound counterpart is that the rules are extended from the usual SAT context to the optimization context by taking into account the cost function to obtain entailed information. Thus, $\texttt{DPLL}_{\texttt{BB}}$ can model concepts such as, e.g., cost-based propagation and cost-based backjumping. To exploit the cost function in the search with these techniques, $\texttt{DPLL}_{\texttt{BB}}$ assumes the existence of a lower bounding procedure that, additionally to returning a numerical lower bound, provides a reason for it, i.e., a (presumably short) clause whose violation is a sufficient condition for the computed lower bound, see [MS00,MS04].

The second contribution of the paper is the connection between a $\texttt{DPLL}_{\texttt{BB}}$ execution and a proof of optimality. We show that each time that $\texttt{DPLL}_{\texttt{BB}}$ backjumps due to a soft conflict (i.e. the lower bound indicates that it is useless to extend the current assignment) we can infer a cost-based lemma, which is entailed from the problem. By recording these lemmas (among others), we can construct a very intuitive optimality proof.

This work could have been cast into the framework of SAT Modulo Theories (SMT) with a sequence of increasingly stronger theories [NO06]. However, the generation of proofs for SMT with theory strengthening has not been worked out (although the generation of unsatisfiable cores for normal SMT was analyzed in [CGS07]), and would in any case obfuscate the simple concept of proof we have here. Also, we believe that in its current form, the way we have integrated the concepts of lower bounding and cost-based propagation and learning is far more useful and accessible to a much wider audience.

This paper is structured as follows. In Section 2 we give some basic notions and preliminary definitions. In Section 3 the $\texttt{DPLL}_{\texttt{BB}}$ procedure is presented, whereas in Section 4 we develop the framework for the generation of proof certificates. Section 5 shows several important instances of problems that can be handled with $\texttt{DPLL}_{\texttt{BB}}$. Finally Section 6 gives conclusions of this work and points out directions for future research.

2 Preliminaries

We consider a fixed set of Boolean variables $\{x_1, \ldots, x_n\}$. *Literals*, denoted by the (subscripted, primed) letter l are elements of the set $\{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$. A *clause* (denoted by the letters $C, D, \ldots$) is a disjunction of literals $l_1 \vee \ldots \vee l_m$. The empty clause will be noted $\Box$. A (partial truth) *assignment* I is a set of literals such that $\{x, \neg x\} \subseteq I$ for no x. A literal l is *true* in I if $l \in I$, *false* in I if $\neg l \in I$, and *undefined* in I otherwise. A clause C is true in I if at least one of its literals is true in I, false in I if all its literals are false in I, and undefined in I otherwise. Note that the empty clause is false in every assignment I. Sometimes we will write $\neg I$ to denote the clause that is the disjunction of the negations of the literals in I. A clause set S is true in I if all its clauses are true in I. Then I is called a *model* of S, and we write $I \models S$ (and similarly if a literal or clause is true in I). We sometimes write $I \models \neg C$ to indicate that all literals of a clause C are false in I.

We consider the following class of problems, which covers a broad spectrum of instances (see Section 5):

Definition 1. *A* Boolean optimization problem *is a pair* $(S, cost)$*, where* S *is a clause set, cost is a function* $cost\colon \mathbb{B}^n \to \mathbb{R}$*, and the goal is to find a model* A *of* S *such that* $cost(A)$ *is minimized.*

Definition 2. *A* cost clause *is an expression of the form* $C \vee c \geq k$ *where* C *is a clause and* $k \in \mathbb{R}$*.*

A cost clause $C \vee c \geq k$ may be better understood with its equivalent notation $\neg C \longrightarrow c \geq k$ which tells that if C is falsified, then the cost function must be greater than or equal to k.

Definition 3. *Let* $(S, cost)$ *be an optimization problem. A cost clause* $C \vee c \geq k$ *is* entailed by $(S, cost)$ *if* $cost(A) \geq k$ *for every model* A *of* S *such that* $A \models \neg C$*.*

Definition 4. *Given an optimization problem* $(S, cost)$*, a real number* k *is called a* lower bound *for an assignment* I *if* $cost(A) \geq k$ *for every model* A *of* S *such that* $I \subseteq A$*.*

A lower bounding procedure *lb is a procedure that, given an assignment* I*, returns a lower bound* k*, denoted* $lb(I)$*, and a cost clause of the form* $C \vee c \geq k$*, called the* lb-reason *of the lower bound, such that* $C \vee c \geq k$ *is entailed by* $(S, cost)$ *and* $I \models \neg C$*.*

Any procedure that can compute a lower bound k for a given I can be extended to a lower bounding procedure: it suffices to generate $\neg I \vee c \geq k$ as the lb-reason. But generating *short* lb-reasons is important for efficiency reasons, and in Section 5 we will see how this can be done for several classes of lower bounding methods.

3 Abstract Branch and Bound

3.1 $\texttt{DPLL}_{\texttt{BB}}$ Procedure

The $\texttt{DPLL}_{\texttt{BB}}$ procedure is modeled by a transition relation, defined by means of rules over states.

Definition 5. *A* $\texttt{DPLL}_{\texttt{BB}}$ state *is a 4-tuple* $I \parallel S \parallel k \parallel A$*, where:*
I *is a sequence of literals representing the current partial assignment,*
S *is a finite set of classical clauses (i.e. not cost clauses),*
k *is a real number representing the best-so-far cost,*
A *is the best-so-far model of* S *(i.e.* $cost(A) = k$*).*
Some literals l *in* I *are annotated as* decision literals *and written* l^{d}*.*

Note that the *cost* function and the variable set are not part of the states, since they do not change over time (they are fixed by the context).

Definition 6. *The* $\mathtt{DPLL}_{\mathtt{BB}}$ system *consists of the following rules:*

Decide :

$$I \parallel S \parallel k \parallel A \implies I\ l^{\mathsf{d}} \parallel S \parallel k \parallel A \quad \textbf{if} \quad \{\, l \text{ is undefined in } I$$

UnitPropagate :

$$I \parallel S \parallel k \parallel A \implies I\ l \parallel S \parallel k \parallel A \quad \textbf{if} \quad \begin{cases} C \vee l \in S,\ I \models \neg C \\ l \text{ is undefined in } I \end{cases}$$

Optimum :

$$I \parallel S \parallel k \parallel A \implies \mathit{OptimumFound} \quad \textbf{if} \quad \begin{cases} C \in S,\ I \models \neg C \\ \text{no decision literals in } I \end{cases}$$

Backjump :

$$I\ l^{\mathsf{d}}\ I' \parallel S \parallel k \parallel A \implies I\ l' \parallel S \parallel k \parallel A \quad \textbf{if} \quad \begin{cases} C \vee l' \in S,\ I \models \neg C \\ l' \text{ is undefined in } I \end{cases}$$

Learn :

$$I \parallel S \parallel k \parallel A \implies I \parallel S, C \parallel k \parallel A \quad \textbf{if} \quad \{\, (S, \mathit{cost}) \text{ entails } C \vee c \geq k$$

Forget :

$$I \parallel S, C \parallel k \parallel A \implies I \parallel S \parallel k \parallel A \quad \textbf{if} \quad \{\, (S, \mathit{cost}) \text{ entails } C \vee c \geq k$$

Restart :

$$I \parallel S \parallel k \parallel A \implies \emptyset \parallel S \parallel k \parallel A$$

Improve :

$$I \parallel S \parallel k \parallel A \implies I \parallel S \parallel k' \parallel I \quad \textbf{if} \quad \{\, I \models S \text{ and } \mathit{cost}(I) = k' < k$$

As we will see, one can use these rules for finding an optimal solution to a problem (S, cost) by generating an arbitrary derivation $\emptyset \parallel S \parallel \infty \parallel \emptyset \implies \ldots$. It will always terminate with $\ldots \implies I \parallel S' \parallel k \parallel A \implies \mathit{OptimumFound}$. Then A is a minimum-cost model for S with $\mathit{cost}(A) = k$. If S has no models at all then A will be $\emptyset$ and $k = \infty$.

All the rules except Improve are natural extensions of the Abstract DPLL approach of [NOT06]. In the following we briefly explain them.

- The Decide rule represents a case split: an undefined literal l is chosen and added to I, annotated as a decision literal.
- UnitPropagate forces a literal l to be true if there is a clause $C \vee l$ in S whose part C is false in I.
- The Optimum rule expresses that if in a state $I \parallel S \parallel k \parallel A$ in S there is a so-called *conflicting clause* C (i.e., such that $I \models \neg C$), and there is no decision literal in I, then the optimization procedure has terminated, which shows that the best-so-far cost is optimal.
- On the other hand, if there is some decision literal in I and an entailed conflicting clause, then one can always find (and Learn) a *backjump clause*, an entailed cost clause of the form $C \vee l' \vee c \geq k$, such that Backjump using $C \vee l'$ applies (see Lemma 1 below). Good backjump clauses can be

found by *conflict analysis* of the conflicting clause [MSS99,ZMMM01], see Example 3.2 below.
- By Learn one can add any entailed cost clause to S. Learned clauses prevent repeated work in *similar* conflicts, which frequently occur in industrial problems having some regular structure. Notice that when such a clause is learned the $c \geq k$ literal is dropped (it is only kept at a metalevel for the generation of optimality certificates, see Section 4).
- Since a lemma is aimed at preventing future similar conflicts, it can be removed using Forget, when such conflicts are not very likely to be found again. In practice this is done if its *activity*, that is, how many times it has participated in *recent* conflicts, has become low.
- Restart is used to escape from bad search behaviors. The newly learned clauses will lead the heuristics for Decide to behave differently, and hopefully make $\mathtt{DPLL}_{\mathtt{BB}}$ explore the search space in a more compact way.
- Improve allows one to model non-trivial optimization concepts, namely *cost-based backjumping* and *and cost-based propagation*. If $lb(I) \geq k$, the lower bounding procedure can provide an lb-reason $C \vee c \geq k$. As explained above, given this conflicting clause, Backjump applies if there is some decision literal in I, and otherwise Optimum is applicable. A *cost-based propagation* of a literal l that is undefined in I can be made if $lb(I\, l) \geq k$ ([XZ05]; for linear cost functions, cf. the "limit lower bound theorem" of [CM95]). Then again the corresponding lb-reason is conflicting and either Backjump or Optimum applies.

Lemma 1. *(See [NOT06] for proofs of this and other related properties.) Let $(S, cost)$ be an optimization problem, and assume*

$$\emptyset \parallel S \parallel \infty \parallel \emptyset \implies \ldots \implies I \parallel S' \parallel k \parallel A$$

If there is some decision literal in I and C is entailed by $(S', cost)$ and conflicting in I, then I is of the form $I'\, l^{\mathsf{d}}\, I''$ and there exists a backjump clause, i.e., a cost clause of the form $C \vee l' \vee c \geq k$ that is entailed by $(S', cost)$ and such that $I' \models \neg C$ and l' is undefined in I'.

The potential of the previous rules will be illustrated in Section 3.2. The correctness of $\mathtt{DPLL}_{\mathtt{BB}}$ is summarized in Theorem 1:

Definition 7. *A derivation $\emptyset \parallel S \parallel \infty \parallel \emptyset \implies \ldots$ is* progressive *if it contains only finitely many consecutive Learn or Forget steps and Restart is applied with increasing periodicity.*

Theorem 1. *Let $(S, cost)$ be an optimization problem, and consider a progressive derivation with initial state $\emptyset \parallel S \parallel \infty \parallel \emptyset$. Then this derivation is finite. Moreover, if a* final state *is reached, i.e., a state to which no rule can be applied, then the derivation is of the form*

$$\emptyset \parallel S \parallel \infty \parallel \emptyset \implies \ldots \implies I \parallel S' \parallel k \parallel A \implies \mathit{OptimumFound}$$

and then A is a minimum-cost model for S, where $cost(A) = k$. In particular, S has no models if, and only if, $k = \infty$ and $A = \emptyset$.

Of course the previous formal result provides more freedom in the strategy for applying the rules than needed. Practical implementations will only generate progressive derivations. Typically at each conflict the backjump clause is learned, and from time to time a certain portion of the learned clauses is forgotten (e.g., the 50% of less active ones). Restarts are applied with increasing periodicity by, e.g., restarting after a certain number N of conflicts and then increasing N.

3.2 $\mathtt{DPLL}_{\mathtt{BB}}$ Example

Consider the clause set S defined over $x_1, \ldots x_6$ (denoting $\neg x_i$ by $\bar{x}_i$):

1.	$x_2 \vee x_4$	5.	$x_1 \vee \bar{x}_3 \vee \bar{x}_6$
2.	$x_2 \vee \bar{x}_5$	6.	$\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_6$
3.	$x_4 \vee \bar{x}_5$	7.	$x_2 \vee x_3 \vee x_5 \vee \bar{x}_6$
4.	$x_5 \vee x_6$	8.	$x_2 \vee \bar{x}_3 \vee x_5 \vee \bar{x}_6$

where $cost(x_1, \ldots x_6) = 1x_1 + 2x_2 + \ldots + 6x_6$, i.e., subindices are cost coefficients. We start an $\mathtt{DPLL}_{\mathtt{BB}}$ derivation, first deciding x_6 to be false (setting high-cost variables to false can be a good heuristic):

$$
\begin{array}{lll}
 & \emptyset & \| S \| \infty \| \emptyset \\
\Longrightarrow_{\mathsf{Decide}} & \bar{x}_6^{\mathsf{d}} & \| S \| \infty \| \emptyset \\
\Longrightarrow_{\mathsf{UnitPropagate}} & \bar{x}_6^{\mathsf{d}} x_5 & \| S \| \infty \| \emptyset \\
\Longrightarrow_{\mathsf{UnitPropagate}} & \bar{x}_6^{\mathsf{d}} x_5 x_2 & \| S \| \infty \| \emptyset \\
\Longrightarrow_{\mathsf{UnitPropagate}} & \bar{x}_6^{\mathsf{d}} x_5 x_2 x_4 & \| S \| \infty \| \emptyset \\
\Longrightarrow_{\mathsf{Decide}} & \bar{x}_6^{\mathsf{d}} x_5 x_2 x_4 \bar{x}_3^{\mathsf{d}} & \| S \| \infty \| \emptyset \\
\Longrightarrow_{\mathsf{Decide}} & \bar{x}_6^{\mathsf{d}} x_5 x_2 x_4 \bar{x}_3^{\mathsf{d}} \bar{x}_1^{\mathsf{d}} & \| S \| \infty \| \emptyset
\end{array}
$$

Now, since $\bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1$ is a model of S of cost $11 < \infty$, we can apply Improve and the corresponding *lb*-reason, e.g., $\bar{x}_2 \vee \bar{x}_4 \vee \bar{x}_5 \vee c \geq 11$, then becomes a conflicting clause. Intuitively, it expresses that any assignment where x_2, x_4 and x_5 are set to true must have cost at least 11. Now, a *conflict analysis* procedure starting form this conflicting clause can be used to compute a backjump clause. This is done by successive resolution steps on the conflicting clause, resolving away the literals $\bar{x}_4$ and $\bar{x}_2$ in the reverse order their negations were propagated, with the respective clauses that caused the propagations:

$$
\dfrac{\dfrac{\bar{x}_2 \vee \mathbf{\bar{x}_4} \vee \bar{x}_5 \vee c \geq 11 \quad \mathbf{x_4} \vee \bar{x}_5}{\mathbf{\bar{x}_2} \vee \bar{x}_5 \vee c \geq 11} \qquad \mathbf{x_2} \vee \bar{x}_5}{\bar{x}_5 \vee c \geq 11}
$$

until a single literal of the current decision level (called the *1UIP*) is left, yielding $\bar{x}_5 \vee c \geq 11$. Learning the clause $C = \bar{x}_5$ allows one to jump from decision level 3 back to decision level 0 and assert x_5. All this can be modeled as follows:

$$
\begin{array}{lll}
\ldots \Longrightarrow_{\mathsf{Improve}} & \bar{x}_6^{\mathsf{d}} x_5 x_2 x_4 \bar{x}_3^{\mathsf{d}} \bar{x}_1^{\mathsf{d}} & \| S \| 11 \| \bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1 \\
\Longrightarrow_{\mathsf{Learn}} & \bar{x}_6^{\mathsf{d}} x_5 x_2 x_4 \bar{x}_3^{\mathsf{d}} \bar{x}_1^{\mathsf{d}} & \| S, C \| 11 \| \bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1 \\
\Longrightarrow_{\mathsf{Backjump}} & \bar{x}_5 & \| S, C \| 11 \| \bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1
\end{array}
$$

Now the derivation could continue, e.g., as follows:

$$\begin{array}{lllll} \ldots & \Longrightarrow_{\mathsf{UnitPropagate}} & \bar{x}_5 x_6 & \| \, S, C \; \| \, 11 \; \| \, \bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1 \\ & \Longrightarrow_{\mathsf{Decide}} & \bar{x}_5 x_6 \bar{x}_4^{\mathsf{d}} & \| \, S, C \; \| \, 11 \; \| \, \bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1 \\ & \Longrightarrow_{\mathsf{UnitPropagate}} & \bar{x}_5 x_6 \bar{x}_4^{\mathsf{d}} x_2 & \| \, S, C \; \| \, 11 \; \| \, \bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1 \end{array}$$

Now notice that x_3 is not assigned, and that since x_2 and x_6 are true in the current partial assignment any assignment strictly improving the best-so-far cost 11 must assign x_3 to false. As explained above, this cost-based propagation can be modeled as follows. The lower bounding procedure expresses the fact that any solution setting x_2, x_3 and x_6 to true has cost no better than 11 by means of the *lb*-reason $\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_6 \vee c \geq 11$. This is an entailed cost clause that is learned as $C' = \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_6$. Then literal $\bar{x}_3$ is propagated.

$$\begin{array}{lllll} \ldots & \Longrightarrow_{\mathsf{Learn}} & \bar{x}_5 x_6 \bar{x}_4^{\mathsf{d}} x_2 & \| \, S, C, C' \; \| \, 11 \; \| \, \bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1 \\ & \Longrightarrow_{\mathsf{UnitPropagate}} & \bar{x}_5 x_6 \bar{x}_4^{\mathsf{d}} x_2 \bar{x}_3 & \| \, S, C, C' \; \| \, 11 \; \| \, \bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1 \end{array}$$

If we now UnitPropagate x_1 with clause 5, clause 6 becomes conflicting. As usual, a backjump clause is computed by doing conflict analysis from the falsified clause, using among others the clause C' that was learned to propagate $\bar{x}_3$:

$$\dfrac{\dfrac{\mathbf{x_1} \vee x_3 \vee \bar{x}_6 \quad \bar{\mathbf{x}}_{\mathbf{1}} \vee x_3 \vee \bar{x}_6}{\mathbf{x_3} \vee \bar{x}_6} \qquad \bar{x}_2 \vee \bar{\mathbf{x}}_{\mathbf{3}} \vee \bar{x}_6 \vee c \geq 11}{\bar{x}_2 \vee \bar{x}_6 \vee c \geq 11}$$

Learning $C'' = \bar{x}_2 \vee \bar{x}_6$ allows one to jump back to decision level 0 asserting $\bar{x}_2$.

$$\begin{array}{lllll} \ldots & \Longrightarrow_{\mathsf{UnitPropagate}} & \bar{x}_5 x_6 \bar{x}_4^{\mathsf{d}} x_2 \bar{x}_3 x_1 & \| \, S, C, C' & \| \, 11 \; \| \, \bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1 \\ & \Longrightarrow_{\mathsf{Learn}} & \bar{x}_5 x_6 \bar{x}_4^{\mathsf{d}} x_2 \bar{x}_3 x_1 & \| \, S, C, C', C'' & \| \, 11 \; \| \, \bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1 \\ & \Longrightarrow_{\mathsf{Backjump}} & \bar{x}_5 x_6 \bar{x}_2 & \| \, S, C, C', C'' & \| \, 11 \; \| \, \bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1 \end{array}$$

Finally after unit propagating with clause 7 one gets a conflict with clause 8, and as no decision literals are left, the optimization procedure terminates:

$$\begin{array}{lllll} \ldots & \Longrightarrow_{\mathsf{UnitPropagate}} & \bar{x}_5 x_6 \bar{x}_2 x_3 & \| \, S, C, C', C'' & \| \, 11 \; \| \, \bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1 \\ & \Longrightarrow_{\mathsf{Optimum}} & \mathit{OptimumFound} & & \end{array}$$

□

4 Certificates of Optimality

In the following, we show how from a certain trace of an $\mathtt{DPLL}_{\mathtt{BB}}$ execution one can extract a formal proof of optimality in a proof system asserting "A is an optimal model of S with respect to *cost*". Our proof system relies on the following type of resolution over cost clauses,

Definition 8. *The* Cost Resolution *rule is the following inference rule with two cost clauses as premises and another cost clause as conclusion:*

$$\dfrac{x \vee C \vee c \geq k \qquad \neg x \vee D \vee c \geq k'}{C \vee D \vee c \geq min(k, k')} \qquad \text{Cost Resolution}$$

Cost Resolution behaves like classical resolution, except in that it further exploits the fact that $c \geq k \vee c \geq k'$ is equivalent to $c \geq min(k, k')$. In what follows, when needed a clause C from S will be seen as the trivially entailed cost clause $C \vee c \geq \infty$.

Theorem 2. *Cost Resolution is* correct*, that is, if $x \vee C \vee c \geq k$ and $\neg x \vee D \vee c \geq k'$ are cost clauses entailed by an optimization problem $(S, cost)$, then $C \vee D \vee c \geq min(k, k')$ is also entailed by $(S, cost)$.*

Definition 9. *Let S be a set of cost clauses and let C be a cost clause. A* Cost Resolution proof *of C from S is a binary tree where:*

- *each node is (labeled by) a cost clause*
- *the root is C*
- *the leaves are clauses from S*
- *every non-leaf node has two parents from which it can be obtained in one Cost Resolution step.*

Together with a model A such that $cost(A) = k$, a *k-lower-bound certificate* as we define now gives a precise k-optimality certificate for $(S, cost)$:

Definition 10. *A* k-lower-bound certificate *for an optimization problem $(S, cost)$ consists of the following three components:*

1. *a set of cost clauses S'*
2. *a Cost-Resolution Proof of the clause $c \geq k$ from $S \cup S'$*
3. *for each cost clause in S', a proof of entailment of it from $(S, cost)$*

As we will see, the set of cost clauses S' of component 1. of this definition corresponds to the different *lb*-reasons generated by the lower bounding procedure that may have been used along the $\texttt{DPLL}_{\texttt{BB}}$ derivation. A very simple independent k-lower-bound certificate checker can just check the cost resolution proof, *if the lower bounding procedure is trusted in that indeed all cost clauses of S' are entailed*. Then, since by correctness of Cost Resolution the root $c \geq k$ of a Cost Resolution proof is entailed if the leaves are entailed, a k-lower-bound certificate guarantees that $c \geq k$ is indeed entailed by $(S \cup S', cost)$, and the entailment of $c \geq k$ by definition means that "$cost(A) \geq k$ for every model A of S".

If one cannot trust the lower bounding procedure, then also component 3. is needed. The notion of a "proof of entailment" from $(S, cost)$ for each cost clause in S' of course necessarily depends on the particular lower bounding procedure used, and an independent optimality proof checker should hence have some knowledge of the deductive process used by the lower bounding procedure. This aspect is addressed in detail in the next section.

4.1 Generation of k-lower-bound certificates

Each time an *lb*-reason is generated and used in an $\texttt{DPLL}_{\texttt{BB}}$ execution, it is written to a file which we will call S' since it corresponds to component 1. of the k-lower-bound certificate. Now observe that any execution of $\texttt{DPLL}_{\texttt{BB}}$ terminates with a

step of Optimum, i.e., with a conflict at decision level 0. From a standard SAT solver point of view, this means that $S \cup S'$ forms an unsatisfiable SAT instance and a refutation proof for this contradiction can be reconstructed as follows (cf. [ZM03] for details). All clauses in S and in S' get a unique identifier (ID). Each time a backjump step takes place, the backjump clause also gets a (unique) ID and a line `ID ID1 ... IDm` is written to a trace file, where `ID1 ... IDm` are the IDs of all parent clauses in the conflict analysis process generating this backjump clause. A last line is written when the conflict at decision level 0 is detected for the parents of this last conflict analysis which produces the empty clause. By processing backwards this trace file, composing all the component resolution proofs from each conflict analysis, a resolution proof from $S \cup S'$ of the last derived clause, i.e., the empty clause, can be constructed.

If we recover the cost literals of cost clauses (recall that the Learn rule of $\mathtt{DPLL}_{\mathtt{BB}}$ drops the cost literal) in the refutation proof, it turns out that it becomes a k-lower-bound certificate where k is the optimum of the problem. The reason is that in a Cost Resolution proof the cost literal of the root clause is the minimum among the cost literals of the leaf clauses. The following example illustrates the whole process.

Example 1. For the $\mathtt{DPLL}_{\mathtt{BB}}$ derivation of Section 3.2, the initial clauses have ID's `1-8`, the set the S' will contain the the *lb*-reasons $\bar{x}_2 \vee \bar{x}_4 \vee \bar{x}_5 \vee c \geq 11$ and $\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_6 \vee c \geq 11$ with ID's `9` and `10` respectively. The two backjump clauses $\bar{x}_5 \vee c \geq 11$ and $\bar{x}_2 \vee \bar{x}_6 \vee c \geq 11$ and the final "empty" clause $c \geq 11$ get ID's `11,12,13` respectively, and the trace file will be:

$$\begin{array}{l} \mathtt{11} \leftarrow \mathtt{2,3,9} \\ \mathtt{12} \leftarrow \mathtt{5,6,10} \\ \mathtt{13} \leftarrow \mathtt{4,7,8,11,12} \end{array}$$

By processing this file backwards it is straightforward to produce a Cost Resolution proof of $c \geq 11$. This is done below, where for lack of space the proof has been split in two at the clause marked with $(*)$. This proof, together with each *lb*-reason and its entailment certificate, will constitute an 11-lower-bound certificate. The optimality certificate is finally obtained with the addition of the 11-upper-bound certificate $\bar{x}_6 x_5 x_2 x_4 \bar{x}_3 \bar{x}_1$.

$$\dfrac{\dfrac{x_2 \vee x_3 \vee x_5 \vee \bar{x}_6 \quad x_2 \vee \bar{x}_3 \vee x_5 \vee \bar{x}_6}{x_2 \vee x_5 \vee \bar{x}_6} \quad \dfrac{\dfrac{x_1 \vee x_3 \vee \bar{x}_6 \quad \bar{x}_1 \vee x_3 \vee \bar{x}_6}{x_3 \vee \bar{x}_6} \quad \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_6 \vee c \geq 11}{\bar{x}_2 \vee \bar{x}_6 \vee c \geq 11}}{x_5 \vee \bar{x}_6 \vee c \geq 11 \; (*)}$$

$$\dfrac{\dfrac{x_5 \vee x_6 \quad x_5 \vee \bar{x}_6 \vee c \geq 11 \; (*)}{x_5 \vee c \geq 11} \quad \dfrac{\dfrac{\bar{x}_2 \vee \bar{x}_4 \vee \bar{x}_5 \vee c \geq 11 \quad x_4 \vee \bar{x}_5}{\bar{x}_2 \vee \bar{x}_5 \vee c \geq 11} \quad x_2 \vee \bar{x}_5}{\bar{x}_5 \vee c \geq 11}}{c \geq 11}$$

5 Instances of $\mathtt{DPLL}_{\mathtt{BB}}$ and Lower Bounding Procedures

In order to complete the method for generating optimality certificates, in this section we show, for different classes of cost functions, several lower bounding procedures together with ways for proving the entailment from $(S, cost)$ for any lb-reason $C \vee c \geq k$ they generate.

Of course a general approach for this is to provide a list of all the models $A_1, \ldots, A_m$ of $S \wedge \neg C$, checking that each one of them has cost at least k, together with a resolution refutation of $S \wedge \neg C \wedge \neg A_1 \wedge \cdots \neg A_m$, which shows that these $A_1, \ldots, A_m$ are indeed all the models of $S \wedge \neg C$.

But this will usually not be feasible in practice. Therefore, we now describe some lower bounding procedures producing simple and compact certificates that can be understood by *ad-hoc* proof checkers.

5.1 Linear Cost Functions

A very important class of optimization problems is that with *linear* cost functions, i.e., of the form $cost(x_1, \ldots, x_n) = \sum_{i=1}^{n} c_i x_i$ for certain $c_i \geq 0$. In this context c_i is called the *cost* of variable x_i. Note that this also covers the cases of negative costs or costs associated to negated variables, which are harmlessly reduced to this one.

Linear Boolean optimization has many applications, amongst others Automatic Test Pattern Generation [FNMS01], FPGA Routing, Electronic Design Automation, Graph Coloring, Artificial Intelligence Planning [HS00] and Electronic Commerce [San99]. In particular the case where $c_i = 1$ for all $1 \leq i \leq n$, called the *Min-Ones* problem, appears naturally in the optimization versions of important well-known NP-complete problems such as the maximum clique or the minimum hitting set problems.

The problem of computing lower bounds for linear optimization problems in a branch-and-bound setting has been widely studied in the literature. Here we consider the two main techniques for that purpose: *independent sets* and *linear programming*.

Independent Sets. Given a partial assignment I and a clause C, let $undef_I(C)$ denote the set of literals in C which are undefined in I, i.e., $undef_I(C) = \{l \in C \mid l \notin I \text{ and } \neg l \notin I\}$. A set of clauses M is an *independent set for* I if:

- for all $C \in M$, neither $I \models C$ nor $I \models \neg C$;
- for all $C \in M$, $undef_I(C)$ is non-empty and only contains positive literals;
- for all $C, C' \in M$ such that $C \neq C'$, $undef_I(C) \cap undef_I(C') = \emptyset$.

If M is an independent set for I, any total assignment extending I and satisfying M has cost at least

$$K = \sum_{x_i \in I} c_i + \sum_{C \in M} \min\{c_j \mid x_j \in C \text{ and } \neg x_j \notin I\}$$

since satisfying each clause C of M will require to add the minimum cost of the positive non-false (in I) literals in C. Independent sets have been used in e.g., [Cou96,MS02]. In [FM06] they are precomputed in order to speed up the actual branch-and-bound procedure.

In this case the lower bounding procedure generates the *lb*-reason $\neg I' \vee c \geq K$, where $I' \subseteq I$ contains:

- the positive literals in I with non-null cost;
- the positive literals whose negations appear in M (which belong to I); and
- the negative literals $\neg x_i \in I$ such that $x_i \in C$ for some $C \in M$ and $c_i < \min\{c_j \mid x_j \in C \text{ and } \neg x_j \notin I\}$.

For this lower bounding procedure a proof of entailment of the *lb*-reason must of course contain the independent set M itself. Then the proof checker can check that $M \subseteq S$, that M is indeed independent for I and that $K \geq k$.

Example 2. Consider the clause set $S = \{\ x_1 \vee x_3 \vee x_5,\ x_2 \vee x_4 \vee x_5 \vee \neg x_6,\ \neg x_1 \vee \neg x_2\ \}$, and the cost function $cost(x_1, x_2, x_3, x_4) = \sum_{i=1}^{6} i \cdot x_i$. It is easy to see that $M = \{x_1 \vee x_3 \vee x_5,\ x_2 \vee x_4 \vee x_5 \vee \neg x_6\}$ is an independent set for the partial assignment $I = \{\neg x_5, x_6\}$. The lower bound is $6 + \min(1,3) + \min(2,4) = 9$, and the *lb*-reason $x_5 \vee \neg x_6 \vee c \geq 9$ is produced. □

Linear Programming [LD97,Li04]. This approach for computing lower bounds relies on the fact that linear Boolean optimization is a particular case of 0-1 Integer Linear Programming. Indeed, such a Boolean optimization problem can be transformed into an integer program by imposing for each variable x that $0 \leq x \leq 1$ and $x \in \mathbb{Z}$, and transforming each clause $x_1 \vee \cdots \vee x_n \vee \neg y_1 \vee \cdots \vee \neg y_m$ into the linear constraint $\sum_{i=1}^{n} x_i + \sum_{j=1}^{m} (1 - y_j) \geq 1$. The current partial assignment I is encoded by imposing additional constraints $x = 1$ if $x \in I$, $x = 0$ if $\neg x \in I$. Then a lower bound can be computed by dropping the integrality condition and solving the resulting relaxation in the rationals with an LP solver.

If K is the lower bound obtained after solving the relaxation, an *lb*-reason of the form $\neg I' \vee c \geq K$ where $I' \subseteq I$ can be computed using an exact dual solution of multipliers [Sch86] (which may be computed by an exact LP solver [Mak08]). Moreover, a proof of entailment of this *lb*-reason consists in the dual solution itself, which proves the optimality of K.

Example 3. Consider again the clause set, the cost function and the partial assignment as in Example 2. In this case the linear program is

$$\min\{x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 + 6x_6 \mid x_1 + x_3 + x_5 \geq 1,\ x_2 + x_4 + x_5 - x_6 \geq 0,\ -x_1 - x_2 \geq -1,\ x_5 = 0,\ x_6 = 1,\ 0 \leq x_1, x_2, x_3, x_4 \leq 1\},$$

whose optimum is 11. A proof of optimality (in fact, of the lower bound) is:

$$\begin{array}{rcccccccccccc}
 & & x_1 & +2\,x_2 & +3\,x_3 & +4\,x_4 & +5\,x_5 & +6\,x_6 & -11 & = \\
+3 & (& x_1 & & + & x_3 & & + & x_5 & -1 &) \\
+4 & (& & x_2 & & + & x_4 & + & x_5 - & x_6 &) \\
+2 & (& -x_1 - & x_2 & & & & & & +1 &) \\
-2 & & & & & & x_5 & & & & \\
+10 & (& & & & & & x_6 & -1 &) &
\end{array}$$

which witnesses that $x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 + 6x_6 \geq 11$ for all x_1, x_2, x_3, x_4, x_5, x_6 such that $x_1 + x_3 + x_5 \geq 1$, $x_2 + x_4 + x_5 - x_6 \geq 0$, $-x_1 - x_2 \geq -1$, $x_5 \leq 0$ and $x_6 \geq 1$. This can be used as a proof of entailment of the *lb*-reason $x_5 \vee \neg x_6 \vee c \geq 11$ (notice that none of the literals of the assignment is dropped in the *lb*-reason since both $x_5 \leq 0$ and $x_6 \geq 1$ are used). □

5.2 Max-SAT

In a (partial weighted) Max-SAT problem $(S, cost)$, the cost function is defined by a set of so-called *soft* clauses S' with a *weight* function $\omega : S' \to \mathbb{R}$. Then the cost of a total assignment A is the sum of the weights of the clauses in S' that are false in A. Note that S' is disjoint from the (possibly empty) set of clauses S, which are called *hard* clauses in this context. Max-SAT has many applications, among others Probabilistic Reasoning [Par02], Frequency Assignment [CdGL$^+$99], Computer Vision, Machine Learning and Pattern Recognition (see the introduction of [Wer05]).

Max-SAT as a non-linear polynomial cost function. Given a clause $C = y_1 \vee \ldots \vee y_p \vee \neg z_1 \vee \ldots \vee \neg z_q$ over a set of variables $\{x_1 \ldots x_n\}$, the polynomial $p_C(x_1, \ldots, x_n) = \prod_{i=1}^{p}(1 - y_i) \cdot \prod_{j=1}^{m}(z_j)$ fulfills for any total assignment A that $p_C(A) = 1$ if $A \models \neg C$, and $p_C(A) = 0$ otherwise. Therefore we have that $cost(A) = \sum_{C \in S'} p_C(A) \cdot \omega(C)$.

Linear Boolean optimization vs Max-SAT. Linear Boolean optimization can be cast as an instance of Max-SAT by having one soft unit positive clause for each variable with non-null cost, with this cost as weight. Reciprocally, Max-SAT can be expressed as a linear optimization problem by adding slack variables to soft clauses. However, this translation is normally unpractical, making the SAT solver extremely slow, since, e.g., it hinders the application of unit propagation [ANORC08].

Branch and bound for Max-SAT. But most of the research in recent years in the Max-SAT community has been devoted to the computation of good quality lower bounds to be used within a branch-and-bound setting. As shown in [LHdG08], most of these lower bounding procedures can be seen as limited forms of *Max-resolution* (see below). Since Max-resolution is sound, theoretically one can in fact use it to certify optimality in any Max-SAT problem. But the growth in the number of clauses makes this unpractical except for small problems. However, one can use it for the proof of entailment for individual *lb*-reasons.

For simplicity, we show here Max-resolution for soft clauses of the form $(l_1 \vee l_2, w)$, where w denotes the weight:

$$\frac{(x \vee a, u) \quad (\neg x \vee b, v)}{(a \vee b, m)(x \vee a, u - m)(\neg x \vee b, v - m)(x \vee a \vee \neg b, m)(\neg x \vee b \vee \neg a, m)}$$

where $m = \min(u, v)$ and the conclusions replace the premises instead of being added to the clause set.

Example 4. Consider a Max-SAT problem without hard clauses and where soft clauses are $S' = \{\ (x_1 \vee x_2 \vee x_3, 1),\ (x_1 \vee \neg x_2 \vee x_3, 2),\ (\neg x_1 \vee x_2 \vee x_3, 3),\ (\neg x_1 \vee \neg x_2 \vee x_3, 4)\ \}$. Given the partial assignment $I = \{\neg x_3, x_4\}$, by means of the following steps of Max-resolution

$$\dfrac{\dfrac{(x_1 \vee \mathbf{x_2} \vee x_3, 1) \quad (x_1 \vee \neg \mathbf{x_2} \vee x_3, 2)}{\vdots\ (\mathbf{x_1} \vee x_3, 1)} \quad \dfrac{(\neg x_1 \vee \mathbf{x_2} \vee x_3, 3) \quad (\neg x_1 \vee \neg \mathbf{x_2} \vee x_3, 4)}{\vdots\ (\neg \mathbf{x_1} \vee x_3, 3)}}{(x_3, 1)}$$

one gets clause x_3 with weight 1. Taking into account the partial assignment $I = \{\neg x_3, x_4\}$, this clause implies that 1 is a lower bound and a *lb*-reason is $\neg x_3 \vee c \geq 1$. Moreover, the proof of Max-resolution above proves the entailment of the *lb*-reason. □

6 Conclusions

Our abstract DPLL-based branch-and-bound algorithm, although being very similar to abstract DPLL, can model optimization concepts such as cost-based propagation and cost-based learning. Thus, $\texttt{DPLL}_{\texttt{BB}}$ is natural to SAT practitioners, but still faithful to most state-of-the-art branch-and-bound solvers. Interestingly, several branch-and-bound solvers, even state-of-the-art ones, still do not use cost-based backjumping and propagation, which appear naturally in $\texttt{DPLL}_{\texttt{BB}}$. Our formal definition of optimality certificates and the description of how a $\texttt{DPLL}_{\texttt{BB}}$ trace can be used to generate them turns out to be elegant and analagous to the generation of refutation proofs by resolution in SAT.

We think that $\texttt{DPLL}_{\texttt{BB}}$ will help understanding and reasoning about new branch-and-bound implementations and extensions. For example, it is not difficult to use it for computing the *m best* (i.e., lowest-cost) models for some m, or for computing all models with cost lower than a certain threshold, and also the certificates for these can be derived without much effort.

References

[ANORC08] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Efficient Generation of Unsatisfiability Proofs and Cores in SAT. In *Proc. of LPAR'08*, vol. 5330 of *LNCS*, pp. 16–30. Springer, 2008.

[CdGL$^+$99] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio Link Frequency Assignment. *Constraints*, 4:79–89, 1999.

[CGS07] A. Cimatti, A. Griggio, and R. Sebastiani. A Simple and Flexible Way of Computing Small Unsatisfiable Cores in SAT Modulo Theories. In *Proc. of SAT'07*, vol. 4501 of *LNCS*, pp. 334–339. Springer, 2007.

[CM95] O. Coudert and J. C. Madre. New Ideas for Solving Covering Problems. In *Proc. of DAC'95*, pp. 641–646. ACM, 1995.

[Cou96] O. Coudert. On Solving Binate Covering Problems. In *Proc. of DAC'96*, pp. 197–202. ACM, 1996.

[FM06] Z. Fu and S. Malik. Solving the Minimum Cost Satisability Problem Using SAT Based Branch-and-Bound Search. In *Proc. of ICCAD'96*, pp. 852 – 859, 2006.

[FNMS01] P. F. Flores, H. C. Neto, and J. P. Marques-Silva. An Exact Solution to the Minimum Size Test Pattern Problem. *ACM Trans. Des. Autom. Electron. Syst.*, 6(4):629–644, 2001.

[HS00] H. H. Hoos and T. Sttzle. SATLIB: An Online Resource for Research on SAT. In *Proc. of SAT'00*, pp. 283–292. IOS Press, 2000. SATLIB is available online at www.satlib.org.

[LD97] S. Liao and S. Devadas. Solving Covering Problems Using LPR-Based Lower Bounds. In *Proc. of DAC'97*, pp. 117–120. ACM, 1997.

[LHdG08] J. Larrosa, F. Heras, and S. de Givry. A Logical Approach to Efficient Max-SAT Solving. *Artif. Intell.*, 172(2-3):204–233, 2008.

[Li04] X. Y. Li. *Optimization Algorithms for the Minimum-Cost Satisfiability Problem*. PhD thesis, Dept. Comp. Sc., N. Carolina State Univ., 2004.

[Mak08] A. Makhorin. GNU Linear Programming Kit, 2008. `http://www.gnu.org/software/glpk/glpk.html`.

[MS00] V. M. Manquinho and J. P. Marques Silva. Search Pruning Conditions for Boolean Optimization. In *Proc. of ECAI'00*, pp. 103–107. IOS Press, 2000.

[MS02] V. M. Manquinho and J. P. Marques Silva. Search Pruning Techniques in SAT-Based Branch-and-bound Algorithms for the Binate Covering Problem. *IEEE Trans. on CAD of Integ. Circ. and Syst.*, 21(5):505–516, 2002.

[MS04] Vasco M. Manquinho and João P. Marques Silva. Satisfiability-Based Algorithms for Boolean Optimization. *Ann. Math. Artif. Intell.*, 40(3-4):353–372, 2004.

[MSS99] J. Marques-Silva and K. A. Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Trans. Comput.*, 48(5):506–521, 1999.

[NO06] R. Nieuwenhuis and A. Oliveras. On SAT Modulo Theories and Optimization Problems. In *Proc. of SAT'06*, vol. 4121 of *LNCS*, pp. 156–169. Springer, 2006.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006.

[Par02] J. D. Park. Using Weighted Max-SAT Engines to Solve MPE. In *Proc. of AAAI'02*, pp. 682–687, Edmonton, Alberta, Canada, 2002.

[San99] T. Sandholm. An Algorithm for Optimal Winner Determination in Combinatorial Auctions. In *IJCAI-99*, pp. 542–547, 1999.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, 1986.

[Wer05] T. Werner. A Linear Programming Approach to Max-Sum Problem: A review. Technical Report CTU-CMP-2005-25, Center for Machine Perception, Czech Technical University, 2005.

[XZ05] Z. Xing and W. Zhang. Maxsolver: An Efficient Exact Algorithm for (Weighted) Maximum Satisfiability. *Artif. Intell.*, 164(1-2):47–80, 2005.

[ZM03] L. Zhang and S. Malik. Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications. In *Proc. of DATE'03)*, pp. 10880–10885. IEEE Computer Society.

[ZMMM01] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient Conflict Driven Learning in Boolean Satisfiability Solver. In *Proc. of ICCAD'01*, pp. 279–285, 2001.