Using Learnt Clauses in MAXSAT

Jessica Davies, Jeremy Cho, and Fahiem Bacchus

Department of Computer Science, University of Toronto, Canada
{fbacchus,jkcho,jdavies}@cs.toronto.edu

Abstract. MAXSAT is an optimization version of SAT capable of expressing a variety of practical problems. MAXSAT solvers have been designed to take advantage of many of the successful techniques of SAT solvers. However, the most important technique of modern SAT solvers, clause learning, has not been utilized since learnt clauses cannot be soundly added to a MAXSAT theory. In this paper we present a new method that allows SAT clause learning to be exploited in a MAXSAT solver without losing soundness. We present techniques for learning clauses during a branch and bound (B&B) MAXSAT search, a process that is more complicated than standard clause learning. To exploit these learnt clauses we develop a connection between them and bounds that can be used during B&B. This connection involves formulating a hitting set problem and finding bounds on its optimal solution. We present some new techniques for generating useful hitting set bounds and also show how linear and integer programs can be exploited for this purpose, opening the door for a hybrid approach to solving MAXSAT.

1 Introduction

MAXSAT, in its most basic form, is the problem of finding a truth assignment that satisfies the maximum number of clauses of a CNF theory. Partial-MAXSAT is the extension in which some clauses are **hard**. Here the objective is to find a truth assignment that satisfies all the hard clauses and a maximum number of the other (soft) clauses. In weighted MAXSAT, clauses are assigned weights, and the objective is to find a truth assignment that maximizes the sum of the weights of the satisfied clauses. Weighted partial MAXSAT extends weighted MAXSAT by adding hard clauses that must be satisfied.

MAXSAT plays a fundamental role for optimization problems, similar to the role that SAT plays for satisfiability problems. Any optimization problem over finite domain variables can be encoded in MAXSAT. Thus MAXSAT can encode any finite domain MAX-CSP problem [5] (where the aim is satisfy as many constraints as possible), while the weighted versions of MAXSAT can encode most VALUED-CSPS [17]. MAXSAT serves as a general modeling language for such problems.

MAXSAT also has the advantage that it possesses a different structure than the corresponding MAX-CSP and VALUED-CSP formalisms. Each formalism can yield algorithmic insights that can potentially be exploited in the other. See [9,6] for an illustration of this type of cross-fertilization.

The literature on exact MAXSAT solvers is fairly extensive and can be categorized into two main approaches. The first approach solves the problem by

D. Cohen (Ed.): CP 2010, LNCS 6308, pp. 176–190, 2010.

solving a sequence of SAT problems, e.g., [13]. This approach can fully exploit modern SAT solver technology including clause learning, but has a number of drawbacks that we will discuss later. The second approach employs a depth-first branch and bound search, e.g., [7]. In this line of research the key contributions consist of various techniques for computing good lower bounds during the search. Most of these techniques can be understood as applying restricted forms of MAXSAT-resolution [10].

First it should be noted that clauses inferred via standard resolution cannot be soundly added to a MAXSAT theory. If Φ is a MAXSAT CNF theory, and c is a clause inferred by resolution, then it is not sound to add c to Φ. In particular, the new theory $\Phi \cup \{c\}$ can have different solutions than the original. For example, the truth assignment $\pi = (x = \mathit{true}, y = \mathit{false})$ is a solution to the MAXSAT theory $\Phi = \{(\neg x, y), (x), (\neg y)\}$: it falsifies only one clause. However, (y) can be inferred from Φ, but π is not a solution to $\Phi \cup \{(y)\}$: π falsifies two clauses while $(x = \mathit{true}, y = \mathit{true})$ falsifies only one.

In response to this, alternative rules of inference have been developed. In particular, the MAXSAT-resolution rule is an extension of ordinary resolution that is sound and complete for MAXSAT [3,9]. However, MAXSAT-resolution can generate a large number of additional "compensation" clauses with each inference. In fact, the MAXSAT-resolution of two clauses of length k and j can generate up to $k + j$ additional clauses. This makes it difficult to use MAXSAT-resolution during search. Hence, previous work has concentrated on finding restricted cases where MAXSAT-resolution type inferences can be more efficiently applied.

Furthermore, current solvers look for these restricted cases in the reduced MAXSAT theory. That is, at each node of the B&B search tree, the MAXSAT theory has been reduced by the prefix of assigned variables. Current B&B MAXSAT solvers examine the reduced theory to determine if any restricted applications of MAXSAT-resolution can be supported. For example, in the reduced theory two originally long clauses might now be reduced to the unit clauses (x) and $(\neg x)$. These two unit clauses can be MAXSAT-resolved to generate the empty clause, thereby increasing the lower bound. However, in the original theory this resolution step actually corresponds to MAXSAT-resolving the two original long clauses, potentially generating many additional clauses which cannot be stored efficiently. As a result, all of the inferences made must be undone on backtrack and recomputed from scratch at future nodes in the search tree.

This is in stark contrast with SAT clause learning where clauses learnt in one part of the search tree can be utilized anywhere else. In fact, it is this caching and reuse of previous work that makes clause learning so effective.

In this paper we show how standard SAT clause learning can in fact be utilized in any kind of MAXSAT problem (weighted and/or partial). This allows us to learn clauses without having to generate large numbers of additional clauses. Hence we can apply clause learning in many cases where an equivalent MAXSAT-resolution would be impractical. Furthermore, clauses learnt in one part of the search tree or during a preprocessing phase can be reused throughout the rest of the search tree. As pointed out above, learnt clauses cannot be soundly added to a MAXSAT

theory. So to use these learnt clauses soundly we exploit a known relationship between conflicts and hitting sets [16]. This allows us to formulate a hitting set problem from the learnt clauses whose minimal solution provides a lower bound for the B&B search.

To exploit this connection we address two additional problems. First we develop techniques for learning clauses that are more likely to increase the minimal hitting set solution. Second, since computing a minimal hitting set is itself an NP-Complete problem, we develop some tractable techniques for computing lower bounds on the exact solution. We develop two heuristics for this purpose, one of which improves on the heuristic given in [16]. We also can formulate the minimal hitting set problem as an integer program and use linear programming to provide a lower bound approximation. We show how this link to standard OR techniques yields an interesting hybrid approach to the MAXSAT problem.

From our experimental results, we show that this novel approach to computing lower bounds demonstrates potential, in particular it can often quickly prune the search space. On several MAXSAT benchmarks, we show that enough inference can be made to terminate search at the root, while other state-of-the-art solvers, such as MiniMaxSat [7] require search. Our approach does not currently achieve state-of-the-art performance, although we are pursuing a number of ideas aimed at improving its performance. Furthermore, much recent research has focused on ways of improving existing bounding techniques, e.g., by exploiting cyclic structures [11]. Our approach offers a completely new way of producing improved bounds through finding specialized ways of learning new clauses. Finally, we believe that the techniques we present for computing bounds on minimal hitting sets and our empirical results assessing the trade offs between these techniques may also be of independent interest for other applications that rely on computing hitting sets.

In the sequel we first provide some brief background, followed in Section 3 by our results on the connection between solving MAXSAT and the weighted minimal hitting set problem. Section 4 describes proposed methods of approximating the min-weight hitting set, while in Section 5 we consider how to generate learnt clauses to employ in our bounds. We present preliminary experimental results in Section 6 that encourage future work in this direction, as discussed in the conclusion.

2 Background

For simplicity we will formulate the MAXSAT problem in terms of computing the minimal value of an objective function, as opposed to a truth assignment. We have as input a set of propositional variables $x_1, \ldots, x_n$ and a set of clauses $c_1, \ldots, c_m$ over these variables each with an associated weight $wt(c_i)$. Let Π be the set of 2^n truth assignments to the variables x_i. For $\pi \in \Pi$ the function $\pi(c)$ has value 1 if π **falsifies** the clause c and 0 if it satisfies it. The MAXSAT problem is to compute the minimum sum of weights of the falsified clauses over all truth assignments to the variables: $\min_{\pi \in \Pi} \left(\sum_{i=1}^{m} wt(c_i)\pi(c_i) \right)$. Thus the weight of the

satisfied clauses is being maximized. Hard clauses are modeled as clauses c with $wt(c) = \infty$ (or any sufficiently large number). In the case of unweighted MAXSAT, we set $wt(c) = 1$ for all soft clauses c.

Solution methods: It is possible to approximate the optimal solution using methods like local search. But here we are mainly interested in exact solution methods, of which there are two main approaches taken in the literature.

In the first approach, e.g., [13], MAXSAT is solved by solving a sequence of SAT problems. Each run of the SAT solver checks to see if the current theory is UNSAT. If so, the theory is modified so that one (and only one) of the subset of clauses that caused the UNSAT result can be "turned off." This is accomplished by adding new "turn-off" variables to the culprit clauses along with extra clauses constraining the turn-off variables so that only one can be true. Then the SAT solver is run again on the new problem. If the SAT solver returns SAT after k such runs we know that k is the minimum number of clauses that must be falsified when satisfying the remaining clauses.

Although this approach works well on some problems it quickly becomes inefficient when more and more clauses have to be turned off, resulting in SAT problems which are larger and more difficult to solve. In particular, in the ith run many more clauses have been added to handle the turn-off variables, and the solver must search over many combinations of i clauses to turn off. Thus the approach is less effective on highly constrained problems. A more serious limitation however lies in its extension to weighted MAXSAT problems [1,14]. On such problems we must step through all possible values the MAXSAT solution can take, i.e., all distinct sums of clause weights. This can produce an explosion in the number of separate SAT solving episodes required.

The second approach involves a depth-first B&B search, which we also use in our work. Typically, a local search method is used to first compute an approximate solution, whose weight is taken as an initial upper bound on the optimal solution. During search, variables are assigned and any hard clauses (or clauses whose weight exceeds the upper bound) are used to perform unit propagation. After unit propagation various methods are used to compute a lower bound on the weight of clauses that must be falsified below the current node. This lower bound includes the weight of all clauses already falsified by the current assignment. In addition, inference is performed on the not-yet falsified (or satisfied) clauses to determine additions to the lower bound. For example, if two clauses have been reduced to units (x) and $(\neg x)$ then at every node below, one of these clauses will be falsified. Hence, $\min(wt((x)), wt((\neg x))$ can be added to the lower bound. More powerful inference methods are used to handle more complex cases [7,11]. However, as mentioned above, all of these inferences must be undone on backtrack.

3 Learnt Clauses—The Hitting Set Connection

We have already pointed out that learnt clauses cannot be soundly added to a MAXSAT theory (except if these clauses are inferred solely from the hard clauses of the theory). Nevertheless, they can be utilized in a MAXSAT solver.

First consider resolution proofs from a CNF MAXSAT theory Φ. Each proof p of a clause c_k is a sequence of clauses $c_1, \ldots, c_k$, where each c_i is either an input clause from Φ or the result of resolving two previous clauses in the sequence. For any proof p let $ic(p)$ be the set of input clauses used in p, i.e., $p \cap \Phi$. For any set of proofs P let $ic(P)$ be the **set of sets** of input clauses arising from these proofs $\{ic(p)|p \in P\}$.

Initially we will be interested in **refutations**. These are proofs of the empty clause, i.e., $c_k = ()$. Since resolution is sound, any truth assignment that satisfies the input clauses of a refutation must also satisfy $()$. However, no truth assignment can satisfy $()$, hence no truth assignment can satisfy all of these clauses. That is, the input clauses of each refutation form a conflict set [16]. Since we are considering conflicts detectable by a complete inference technique (resolution) we can prove a more general connection between conflicts and solutions to MAXSAT than that given in [16].

For any MAXSAT CNF problem Φ let $minval(\Phi)$ be the solution of Φ, i.e., the minimum achievable weight of falsified clauses; and let $\mathcal{R}_{all}(\Phi)$ be the set of **all refutations** that can be computed from Φ. A hitting set for any **set of sets** of clauses S is a set of clauses $\mathcal{H}$ such that for each $s \in S$, $\mathcal{H} \cap s \neq \emptyset$. The weight of a hitting set $\mathcal{H}$, $val(\mathcal{H})$ is the sum of the weights of the clauses in $\mathcal{H}$, and we let $minwt_{HS}(S)$ denote the weight of a minimum weight hitting set of S. Hence, $minwt_{HS}(ic(\mathcal{R}_{all}(\Phi)))$ is the weight of a minimum weight hitting set of $ic(\mathcal{R}_{all}(\Phi))$.

Theorem 1. *For any* MAXSAT *theory* Φ, $minval(\Phi) = minwt_{HS}(ic(\mathcal{R}_{all}(\Phi)))$.

Proof. Let $\mathcal{H}$ be a minimum weight hitting set of $ic(\mathcal{R}_{all}(\Phi))$.

We cannot prove the empty clause from $\Phi - \mathcal{H}$, otherwise there would be a refutation r in $\mathcal{R}_{all}(\Phi)$ such that $ic(r) \subseteq (\Phi - \mathcal{H})$. But then $\mathcal{H}$ would not hit $ic(r)$ and $\mathcal{H}$ would not be a hitting set of $ic(\mathcal{R}_{all}(\Phi))$. Thus we conclude that $minval(\Phi) \leq val(\mathcal{H})$. Say that $minval(\Phi) < val(\mathcal{H})$. Then there must exist a set of input clauses H with $val(H) < val(\mathcal{H})$ and such that $\Phi - H$ is satisfiable. But since $\mathcal{H}$ is a minimum weight hitting set, H cannot be a hitting set of $ic(\mathcal{R}_{all}(\Phi))$. Hence, there exists a refutation $r \in \mathcal{R}_{all}(\Phi)$ such that $H \cap ic(r) = \emptyset$. That is, r is a refutation provable from $\Phi - H$. This is a contradiction and we conclude that $minval(\Phi) = val(\mathcal{H})$. □

Theorem 1 says that the technique of finding hitting sets can solve the MAXSAT problem: it is a complete method. However, as stated it is also quite impractical. For one there are an exponential number of possible refutations of Φ.[1] Consider the case where instead of having access to $\mathcal{R}_{all}(\Phi)$ we have an incomplete collection of refutations $R \subset \mathcal{R}_{all}(\Phi)$. In this case a minimum hitting set of $ic(R)$ provides a lower bound on $minval(\Phi)$.

Proposition 1. *If* R *and* R' *are two sets of proofs with* $R \subset R'$, *then* $minwt_{HS}(ic(R)) \leq minwt_{HS}(ic(R'))$.

[1] The NP-Hardness of computing a minimum hitting set is also a problem with the direct application of Theorem 1. We address this issue in the next section.

This holds since every hitting set of $ic(R')$ must be a hitting set of $ic(R)$. Thus we have that $minwt_{HS}(ic(R)) \leq minwt_{HS}(ic(\mathcal{R}_{all}(\Phi))) = minval(\Phi)$.

Now we consider how these ideas could be utilized inside of a B&B search. At each node n of the search the original problem has been reduced by instantiating some set of variables. In addition, by employing clause learning techniques (described below) the search has been able to augment the input clauses Φ with an additional set of learnt clauses $\mathcal{L}$. For each $\ell \in \mathcal{L}$ the search has also computed the set of input clauses $ic(\ell)$ used in the proof of ℓ. We note that if c is an input clause ($c \in \Phi$) then a proof of c is simply c itself. Thus for convenience for $c \in \Phi$ we use $ic(c)$ to denote the set $\{c\}$.

At node n we wish to determine if $minval(\Phi|_n)$ (the input formula reduced by the literals made true at node n) has a value that is so high that it precludes finding a better solution under n. Theorem 1 says that $minval(\Phi|_n) = minwt_{HS}(ic(\mathcal{R}_{all}(\Phi|_n)))$ and we can lower bound this value using any subset of $ic(\mathcal{R}_{all}(\Phi|_n))$. Unfortunately we do not know any non-empty subset. The search has never visited n before, and thus has not derived any refutations from $\Phi|_n$.

Instead we will consider clauses that we know to be falsified at node n. Let Π_n be the set of all truth assignments (to the variables in Φ) that agree with the assignments made at node n. Let $minval(n) = \min_{\pi \in \Pi_n} \left(\sum_{i=1}^{m} wt(c_i)\pi(c_i) \right)$; this is the minimum weight of falsified clauses achievable by any truth assignment that lies below node n. We can backtrack immediately from n if we know any lower bound LB such that $LB \leq minval(n)$ and $LB \geq UB$ where UB is the value of the current best known solution.

Let $\mathcal{P}_{all}(\Phi, n)$ be the set of all proofs from Φ that derive a clause falsified by the assignments made at node n. That is, $p \in \mathcal{P}_{all}(\Phi, n)$ means that p derives, from the clauses of Φ, a clause all of whose literals are falsified at node n.

Theorem 2. $minval(n) = minwt_{HS}(ic(\mathcal{P}_{all}(\Phi, n)))$.

This means that if we had access to all proofs of clauses falsified at node n, we could find $minval(n)$ by way of a minimum hitting set problem. We do not have access to $\mathcal{P}_{all}(\Phi, n)$, but as noted above for every clause c of $\Phi \cup \mathcal{L}$ that has been falsified at n we know $ic(c)$, the input clauses used in the proof of c. Hence, $FC = \{ic(c) \mid c \text{ is falsified at } n\} \subset ic(\mathcal{P}_{all}(\Phi, n))$ and Prop. 1 tells us that $minwt_{HS}(FC) \leq minval(n)$. By computing $minwt_{HS}(FC)$ the search can use this value to potentially backtrack away from n.

Proof. The proof of this theorem is more complicated than the proof of Theorem 1, so we provide only a sketch. First, let $\Phi|_n$ be Φ reduced by the assignments made at n, i.e., all satisfied clauses and falsified literals are removed from Φ. It can be shown that $minval(\Phi|_n) = minval(n)$ by observing that for every clause of Φ falsified at n, $\Phi|_n$ contains an empty clause of equal weight. Theorem 1 then shows that $minwt_{HS}(ic(\mathcal{R}_{all}(\Phi|_n))) = minval(n)$. Now all that remains to be done is to prove that $minwt_{HS}(ic(\mathcal{R}_{all}(\Phi|_n))) = minwt_{HS}(ic(\mathcal{P}_{all}(\Phi, n)))$. First, we show that all proofs in $\mathcal{R}_{all}(\Phi|_n)$ can be converted to equivalent proofs in $\mathcal{P}_{all}(\Phi, n)$ by adding back all of the falsified literals to the clauses of the proof. This shows that any hitting set of $ic(\mathcal{P}_{all}(\Phi, n))$ can generate a hitting set of

$ic(\mathcal{R}_{all}(\Phi|_n))$ of no greater weight, and thus that $minwt_{HS}(ic(\mathcal{R}_{all}(\Phi|_n))) \leq minwt_{HS}(ic(\mathcal{P}_{all}(\Phi, n)))$. The other direction is more complex, but we use the same argument as Theorem 1 to show that if $minwt_{HS}(ic(\mathcal{R}_{all}(\Phi|_n))) < minwt_{HS}(ic(\mathcal{P}_{all}(\Phi, n)))$ there must be a hitting set H of $ic(\mathcal{R}_{all}(\Phi|_n))$ which when converted to a set of clauses of Φ (by adding back the falsified literals) cannot be a hitting set of $ic(\mathcal{P}_{all}(\Phi, n))$. Hence, there is a proof $p \in \mathcal{P}_{all}(\Phi, n)$ not covered by H. With a more complex transformation, p can then be converted into a refutation in $\mathcal{R}_{all}(\Phi|_n)$ by removing all satisfied clauses and falsified literals, and then fixing all of the now broken resolution steps. (For example, the literal being resolved on might have been removed from one of the clauses, or one of the clauses might have been satisfied). The conversion of p is not hit by H contradicting that H exists. □

In summary, the results of this section show that learnt clauses can be exploited in MAXSAT to produce lower bounds. In particular, at any node n of the search tree some set of clauses will be falsified. These could be either input or learnt clauses. By keeping track of the input clauses used to derive each clause a hitting set problem can be set up. The minimum weight hitting set provides a lower bound on the best value that can be achieved below node n, $minval(n)$, which can be used by the B&B bounding procedure. Furthermore, it can be the case that some learnt clauses are falsified at node n even though no input clause is violated. Hence, clause learning can allow us to construct richer hitting set problems that can yield better bounds.

Two problems remain. First we cannot necessarily compute a minimum weight hitting set as this is an NP-Complete problem in itself. In the next section we present some ways of computing lower bounds on the minimum weight hitting set, which in turn act as lower bounds on $minval(n)$. Second, we present some ways that clauses can be learnt both prior to search and during search so as to increase the effectiveness of the computed lower bounds.

4 Lower Bounding the Minimal Hitting Set

During clause learning we remember for each learnt clause the set of input clauses used in its derivation. This increases the space required to store learnt clauses, but does not impose additional computational overheads during search. In particular, we only need to access this derivation set when a clause has been falsified.

During search, we use standard watched literal techniques to detect when clauses become false. The collection of falsified clauses at each node of the search tree form a hitting set problem. Let FC represent this collection. For each $f \in FC$ let $ic(f)$ be the set of input clauses used to derive it, and $ic(FC) = \bigcup_{f \in FC} ic(f)$. As noted above, if f is an input clause, i.e., $f \in \Phi$, we let $ic(f) = \{f\}$. The hitting set problem is to select a minimum weight set of input clauses that touches $ic(f)$ for each $f \in FC$. Due to the difficulty of computing this, our aim is to compute a lower bound on the weight of a minimal weight hitting set.

Simplification: Before trying to solve the problem we first apply two quite effective simplification rules [20].

1. If $f_1, f_2 \in FC$ with $ic(f_1) \subseteq ic(f_2)$ remove f_2 from FC. Any hitting set that hits f_1 will necessarily hit f_2.
2. If $wt(c_1) \leq wt(c_2)$ and $\{f | c_1 \in ic(f)\} \supseteq \{f | c_2 \in ic(f)\}$ remove c_2 from all sets $ic(f)$, $f \in FC$. We can substitute c_1 for c_2 in any hitting set without increasing the hitting set's weight.

Note that these two rules define a propagation scheme. That is, one application of these rules can enable additional applications. Simplification continues until neither of these rules can be applied again.

In addition to these two rules, which are applicable for any hitting set, we can also utilize an additional simplification rule specific to our problem.

3. Remove all input clauses c that are satisfied at the current node from $ic(f)$, $f \in FC$.

This last rule is justified by the semantics of our hitting sets. Each falsified clause implies that at least one of the input clauses used to derive it must be falsified by all truth assignments extending the current node. Thus, any clause that is already satisfied is no longer a candidate for falsification.

One thing that is useful to notice is that simplification often generates a collection of disjoint hitting set problems. For example, say that for falsified clause f we have $ic(f) = \{c\}$, perhaps because simplification removed all other members of $ic(f)$. Rule 1 then implies that all other falsified clauses f' with $c \in ic(f')$ will be removed from the hitting set problem. That is, an isolated hitting set problem consisting only of $f = \{c\}$ will be created. These disjoint problems can be solved independently and the answers added.

Heuristics: We have developed two heuristics for lower bounding the minimum weight hitting set.

```
H1(LB)
1. LB = 0
2. while FC ≠ ∅
3.     choose f ∈ FC
4.     LB += min_{c∈ic(f)} wt(c)
5.     F = {f' ∈ FC | ic(f) ∩ ic(f') ≠ ∅}
6.     FC = FC − F
```

This heuristic first chooses some falsified clause, f, and adds its minimum weight input clause to the lower bound. Then it removes f and all clauses that share an input clause with f from the set of falsified clauses. It repeats this loop until no more falsified clauses remain.

The intuition behind this heuristic is simple: we can hit f by selecting its min-weight input clause. But we could have selected any other input clause from $ic(f)$. Hence, the most we could have done is also hit all other falsified clauses connected to f via a member of $ic(f)$. The heuristic conservatively estimates that we did in fact hit all of these clauses with f's min-weight clause. Note

that the heuristic can yield different values dependent on which f is chosen. In our implementation, we use different selection schemes for f in the weighted and unweighted cases. For weighted, we always select the f with the minimum weight input clause of maximum weight, while for unweighted, we select the f whose set F in line 5 is of minimum cardinality. However, other natural selection schemes also exist.

This heuristic inherently takes advantage of any disjoint sub-problems. In particular, if the hitting set problem has been broken up into k disjoint subproblems, `H1` will return an LB that is no worse than the sum of the weights of the minimum weight input clause present in each subproblem.

For the second heuristic, for input clause c let $nbrs(c) = \{f | f \in FC \wedge c \in ic(f)\}$ and $deg(c) = |nbrs(c)|$, that is the number of falsified clauses it hits.

```
H2(LB)
1. LB = 0
2. foreach disjoint subproblem FC_P
3.    lb = 0, n = |FC_P|
4.    while n > 0
5.       c = c ∈ ic(FC_P) that minimizes wt(c)/deg(c)
6.       if deg(c) ≥ n OR unweighted clauses
7.          lb += wt(c)
8.       else lb += n × wt(c)/deg(c)
9.       n -= deg(c)
10.      remove c from ic(f) for all f ∈ FC_P
11.   LB += lb
```

This heuristic generalizes one given in [16]. First it takes advantage of the possible disjointness of the hitting set problem that might arise after simplification, and second it handles the case of weighted input clauses. It operates on each disjoint subproblem by selecting input clauses with lowest weight over degree. These clauses hit the most sets on a minimum cost per set basis. We select enough such minimum average cost input clauses until the sum of their degrees equals or exceeds the total number of sets to hit. However, in the case of weighted clauses, we are only allowed to count part of the weight of the last clause selected (line 8).

Proposition 2. *Both `H1` and `H2` return a lower bound on the weight of the minimum weight hitting set.*

This proposition is easy to see for `H1` and for `H2` in the unweighted case. The weighted case requires a bit more insight, but ultimately it also is not too difficult.

These two heuristics are incomparable. That is, on some problems `H1` provides a better bound than `H2` and vice versa on other problems.

For example, let $FC = \{f_1, f_2, f_3\}, ic(FC) = \{i_1, i_2, i_3\}$ (all unweighted), where $nbrs(i_1) = \{f_1, f_2\}$, $nbrs(i_2) = \{f_1, f_3\}$, $nbrs(i_3) = \{f_2, f_3\}$. Then `H1` can only pick a single element from FC for a LB of 1 while `H2` can pick any two elements from $ic(FC)$ for a LB of 2.

On the other hand, let $FC = \{f_1, f_2, f_3, f_4, f_5, f_6\}, ic(FC) = \{i_1, i_2, i_3\}$ (all unweighted), where $nbrs(i_1) = \{f_1, f_2, f_3\}$, $nbrs(i_2) = \{f_3, f_4, f_5\}$, and $nbrs(i_3) = \{f_5, f_6\}$. Then `H1` can pick $\{f_1, f_4, f_6\}$ for a LB of 3 while `H2` is forced to pick $\{i_1, i_2\}$ for a LB of 2.

Unfortunately both heuristics can yield arbitrarily bad approximations.

Theorem 3. *For hitting set instance $\mathcal{S}$, let `H1`($\mathcal{S}$) and `H2`($\mathcal{S}$) be the lower bounds computed by the heuristics and $minwt_{HS}(\mathcal{S})$ be the weight of the minimum weight hitting set for $\mathcal{S}$. Then for any $\epsilon > 0$, there exists $\mathcal{S}, \mathcal{S}'$ such that `H1`($\mathcal{S}$)/$minwt_{HS}(\mathcal{S})$ $\leq \epsilon$ and `H2`($\mathcal{S}'$)/$minwt_{HS}(\mathcal{S}') \leq \epsilon$*

4.1 Integer Programming Connection

Recall from Section 3 that the quality of the lower bound on the minimum hitting set dictates how soon we can backtrack from a non-optimal partial assignment. Furthermore, by recomputing the lower bound at each node during backtrack[2], it also impacts how far we can backtrack (as long as $LB \geq UB$). In light of this and Theorem 3, it may be desirable to use more powerful techniques to compute the exact value of the minimum hitting set at strategic points during search. One way to do this is to encode the hitting set instance as an integer program and solve it using a Mixed Integer Program (MIP) solver, such as CPLEX. A standard encoding [19] of a hitting set instance with $FC = \{f_1, ..., f_n\}$, $ic(FC) = \{i_1, ..., i_m\}$, where w_k is the weight of i_k, maps FC to constraints and $ic(FC)$ to boolean variables as follows:

$$\text{minimize: } \sum_{k=1}^{m} w_k \cdot x_k \qquad \text{where :}$$

$$\text{for } 1 \leq j \leq n \qquad \sum_{i_k \in ic(f_j)} x_k \geq 1$$

$$\text{for } 1 \leq k \leq m \qquad x_k \in \{0, 1\}$$

Because Integer Programming is itself NP-complete, the backtracking gains of an exact solution to the minimum hitting set problem may be outweighed by the overhead required to compute it, particularly for large instances. As a result, it can be more practical to instead solve the linear programming relaxation, whose solution is a valid lower bound on the minimum hitting set. To balance the trade off between the quality of the minimum hitting set approximation and the computational cost required, we employed the following strategy: first heuristics `H1` and `H2` are computed and their maximum used as initial lower bound LB. If $LB < UB$ but $LB/UB \geq \alpha$, for some tuned parameter α, then the linear program is solved. Finally, if this is still insufficient to exceed UB and if the size of the hitting set problem is less than some other tuned parameter β, the integer program is solved.

[2] Note that it may not be sufficient to simply reuse the lower bound that was computed the first time the node was reached, as additional learnt clauses may have been added to the hitting set instance since then.

5 Learning Clauses

It remains to consider how to generate the learnt clauses; we consider two methods. The first is to perform a preprocessing step to learn clauses during a relaxed DPLL search where soft as well as hard clauses are involved in learning [8]. A drawback to this approach is that the learnts may not be relevant to the subsequent B&B search. For one thing, all soft unit clauses are satisfied during relaxed DPLL but may have to be violated to find the optimum. As observed by Kroc et al., we found that some MAXSAT problems are too easy to refute even for relaxed DPLL, and in many cases few or no learnts can be generated this way. Therefore, we developed techniques to generate learnts during the B&B of a MAXSAT solver.

Soft Unit Propagation: At every node of the B&B search, unit propagation is applied over the hard (with respect to the current UB) clauses to soundly reduce the theory under the current prefix. If a hard conflict is found, we learn a hard clause and backtrack as normal to the asserting level. Otherwise, as in [7], we initialize a phase of *soft unit propagation* (SUP) with all soft clauses that are unit or falsified under the current prefix. Soft propagation involves unit propagating soft clauses as well as hard, so any literals set during the SUP phase must be undone before continuing search with another decision. However, if a clause is falsified during SUP, this conflict can be analyzed using standard techniques to produce a learnt with the desirable property of being falsified under the current prefix, *before* SUP. Thus it may contribute to increasing the hitting set lower bound at the current node. Therefore, the learnts produced are immediately relevant to the search, in contrast to those we obtained from the preprocessing step. The learnt clause, together with its set of input clauses, is saved upon backtrack and can be use in future search whenever it is falsified, to contribute to the hitting set lower bound. The soft learnts also participate in future SUP phases, so that soft learnts can be learned from others, contributing to the power of these clauses to prune the search.

SUP continues to propagate and learn until no more new falsified clauses are found. Thus many soft learnts can be produced at each node of the search, and we update the lower bound after SUP is finished. However, it could be beneficial to limit the amount of SUP learning performed at every node, or update the lower bound with new learnts as soon as we expect to be able to exceed the upper bound. The trade off between the strength of lower bound we can produce and time spent learning will be a focus of future investigation.

Turning off clauses: SUP learning, if implemented as described above, can produce duplicate learnts. Note that in our context, a learnt is only considered a duplicate of another if their sets of input clauses are also the same, since otherwise each can contribute to the lower bound under different circumstances depending on the structure of the hitting set problem. In order to prevent duplicate learnts from creating overhead, we prevent them from being learnt in the first place. This is achieved by "turning off" for SUP, one of the input clauses of each existing falsified learnt, for the duration of time it remains falsified. A

turned-off input clause can't be used to derive any more learnts, until it is turned back on. Of course, this policy may reject new learnts that aren't actually duplicates of an existing one. This is undesirable since it may reduce the strength of the lower bound. To limit the negative impact on heuristics `H1` and `H2`, we always choose to turn off the min-weight input clause, or the one with largest degree. We also consider turning off input clauses only at decision levels greater than k and relying instead on a general scheme of learnt database reduction to limit the number of learnts.

Similarly, when an input clause c is falsified by the current prefix, we turn off all of the learnts it has derived. This prevents us from learning clauses that we know can't currently contribute to the lower bound. To see this, note that any learnt we avoid deriving would have had c in its set of input clauses. Therefore, while c is falsified, it would be used by the hitting set heuristics to cover all such learnts, so the weight of the hitting set would not be increased by their presence.

Related lower bounds: If no clauses are turned off for SUP, and the exact minimum weight hitting set is calculated for the lower bound, this technique subsumes existing unit-propagation based bounds in the MAXSAT literature [12,7,21]. If we turn off clauses for SUP, our lower bound will be at least as good as the disjoint-inconsistent sub-formulas bound [12], since the turned off input clause from one inconsistent sub-formula does not need to be used to refute the others.

6 Empirical Study

We implemented these ideas on top of the SAT solver Minisat [4], to investigate how they perform in practice. We incorporated the variable ordering heuristics used by MiniMaxSat, as well as the natural extension of probing (aka failed literal detection) to our clause learning framework [7]. We also use the Dominating Unit Clause rule to simplify the theory at each node [15]. Our experiments were conducted on a subset of 378 instances from the previous MaxSAT Evaluations [2]. The instances were selected by identifying benchmark families in which MiniMaxSat is unable to solve some instances, where our B&B solver can outperform MiniMaxSat on at least some problems. We then selected particular families in order to represent all types (MAXSAT, partial MAXSAT etc.) and some from each category (random, crafted, industrial). All experiments were conducted on a dual-core 2GHz AMD Opteron processor with 3GB of RAM, and all experiments were run with a 1200 second timeout.

`H1` vs. `H2`: Our first set of experiments investigates the performance of the two hitting set approximation heuristics, `H1` and `H2`, in the context of providing a lower bound during B&B search. We ran the B&B search with both heuristics enabled, and used the maximum of the two as the lower bound. We counted the number of times each of the two heuristics provided different bounds, and calculated the relative amount by which the winner was better. The results over the 226 instances that required at least 100 decisions to solve, are summarized in Table 1. The first column specifies the heuristic that was used, either `H1`

Table 1. Comparison of the `H1` and `H2` lower bounds during search. The 'Freq' column refers to the percentage of all lower bounds calculated for which the heuristic gave the larger bound, averaged over all instances. The 'Size' column gives the average factor by which the bound was larger. The other columns give the average number of decisions, average runtime, and number of instances solved.

LB Heuristic	Freq	Size	Decisions	Time (s)	Num Solved
`H1`	50	1.15	36280	49	115
`H2`	6	1.09	40115	50	115
max(`H1`,`H2`)	–	–	36192	49	117

Table 2. Comparison of the `H1` and CPLEX LP lower bounds during search. The average number of decisions and runtime (over instances both methods solved), and the number of instances solved is shown.

LB Heuristic	Decisions	Time (s)	Num Solved
CPLEX LP	25296	48	105
`H1`	35059	15	115

Table 3. Comparison of MiniMaxSat and our B&B solver using the CPLEX ILP lower bound

Year	Type	Name	Optimum	MiniMaxSat Decisions	Our Decisions	MiniMaxSat Time (s)	Our Time (s)
2007	wpms	8.wcsp.log	2	1	0	0.05	0.05
2007	wpms	norm-mps-v2-20-10-stein15	9	2191970	0	3.8	0.07
2007	wpms	norm-mps-v2-20-10-stein27	18	–	0	>1200	0.59
2007	wpms	norm-mps-v2-20-10-stein9	5	230	0	0.12	0.14
2008	pms	norm-fir01_area_delay	5	14	0	0.14	0.12
2008	pms	norm-fir02_area_partials	19	38	0	0.16	0.06
2008	pms	norm-fir04_area_partials	30	13	0	0.12	0.6
2008	wms	frb10-6-1	50	755	0	0.27	0.23
2008	wms	frb10-6-2	50	678	0	0.13	0.26
2008	wms	frb10-6-3	50	1302	0	0.13	0.23
2008	wms	frb10-6-4	50	580	0	0.29	0.3
2008	wms	frb15-9-1	120	387470	0	1.37	3.76
2008	wms	frb15-9-2	120	206845	0	2.29	4.6
2008	wms	frb15-9-4	120	199365	0	2.24	5.77
2008	wms	frb15-9-5	120	271024	0	1.27	5.59
2009	wpms	warehouse0.wcsp	328	46	0	0.11	0.12

or `H2` alone, or their maximum. The second column shows the percentage of all lower bounds calculated for which the one heuristic gave a larger bound than the other (averaged over all instances). Whenever one heuristic gave a strictly larger bound than the other, we measured the relative difference (e.g. `H1`/`H2` if `H1` was the larger); the third column reports this averaged over all instances. The average number of decisions and average runtime are shown in columns four and five, over the 112 instances that were solved by all three methods. The total number of instances solved using each method is included in the last column. These results encourage us to use both lower bounds and take the maximum, since they are both cheap to calculate and can solve more problems when combined.

`H1` vs. CPLEX LP: We also consider the trade off between using our `H1` heuristic, and solving the linear program for the hitting set problem using CPLEX.

We expected that the dynamic addition and removal of variables and constraints from the CPLEX model would limit the efficiency of this approach, and the results confirm that the added strength of the LP lower bound comes with the price of greater computational cost. We ran our B&B solver with the `H1` lower bound alone, and then with only the CPLEX LP lower bound. Ninety-seven instances were solved by both methods, and the results are shown in Table 2. We see that by using the LP lower bound, we make 28% fewer decisions on average. We found that the reduction in decisions can sometimes pay off in reduced runtime, for 37 instances. However in the majority of cases, the LP bound increases the runtime by an average of about 30%. In general, the extra computational cost does not pay off, since using the stronger LP bound solves 10 fewer problems. These results confirmed our expectations, but demonstrate that a hybrid approach, using the stronger bounds at judicious points during search to exceed the UB, is a well-justified direction for future work.

Solving without search: Finally, we present some results that show the promising potential of our framework to allow stronger yet practical inference. As mentioned at the beginning of this section, we implemented a probing phase at the root of the B&B search. For each literal, we force it to *true* and reduce the theory with this assignment using first hard unit propagation, followed by SUP. If a clause is falsified, we learn a unit or empty clause (associated with a set of input clauses) and move on to probe the next literal. On some instances, the set of clauses we learn during probing is strong enough that the lower bound will equal a tight upper bound provided by one run of ubcsat [18]. This occurs on 16 problems, presented in Table 3. Note that we only report the cases where MiniMaxSat couldn't solve the instance without search. Here, we have used CPLEX to solve the ILP model and generate the lower bound, since the size of the hitting set problem is sufficiently small.

7 Conclusions

We introduced an innovative approach for MAXSAT solving, with potential for practical impact based on generating bounds from unrestricted clause learning for MAXSAT. Although it may always be necessary to use a restricted version on real problems, we argue that this framework provides new insight into how strong lower bounds can be made practical, for example, by being smart about which soft clauses we learn, or by approximating the minimum hitting set well. In addition to these contributions, we present two heuristics for the weighted hitting set problem, and show that this approach can be used effectively in a novel context. Based on our preliminary implementation, we have discovered that the primary challenge in bringing this technique to the state-of-the-art in practical performance, will be to develop methods to learn the best clauses to prune the search tree. This is the topic of ongoing research.

Acknowledgment. This work was supported by the National Research Council of Canada.

References

1. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (weighted) partial maxsat through satisfiability testing. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 427–440. Springer, Heidelberg (2009)
2. Argelich, J., Li, C.M., Manyà, F., Planes, J.: The maxsat evaluations (2007–2009)
3. Bonet, M.L., Levy, J., Manyà, F.: A complete calculus for max-SAT. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 240–251. Springer, Heidelberg (2006)
4. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
5. Freuder, E., Wallace, R.: Partial constraint satisfaction. Artificial Intelligence (AI) 58(1-3), 21–70 (1992)
6. de Givry, S., Larrosa, J., Meseguer, P., Schiex, T.: Solving max-SAT as weighted csp. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 363–376. Springer, Heidelberg (2003)
7. Heras, F., Larrosa, J., Oliveras, A.: MinimaxSAT: An efficient weighted max-sat solver. Journal of Artificial Intelligence Research (JAIR) 31, 1–32 (2008)
8. Kroc, L., Sabharwal, A., Selman, B.: Relaxed dpll search for maxsat. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 447–452. Springer, Heidelberg (2009)
9. Larrosa, J., Heras, F.: Resolution in max-SAT and its relation to local consistency in weighted csps. In: Proceedings of the International Joint Conference on Artifical Intelligence (IJCAI). pp. 193–198 (2005)
10. Larrosa, J., Heras, F., de Givry, S.: A logical approach to efficient max-SAT solving. Artificial Intelligence (AI) 172(2-3), 204–233 (2008)
11. Li, C.M., Manyà, F., Mohamedou, N., Planes, J.: Exploiting cycle structures in max-SAT. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 467–480. Springer, Heidelberg (2009)
12. Li, C.M., Manyà, F., Planes, J.: New inference rules for max-SAT. Journal of Artificial Intelligence Research (JAIR) 30, 321–359 (2007)
13. Liffiton, M., Sakallah, K.: Generalizing core-guided max-SAT. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 481–494. Springer, Heidelberg (2009)
14. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted boolean optimization. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 495–508. Springer, Heidelberg (2009)
15. Niedermeier, R., Rossmanith, P.: New upper bounds for maximum satisfiability. Journal of Algorithms 36, 63–88 (2000)
16. Petit, T., Bessière, C., Régin, J.C.: A general conflict-set based framework for partial constraint satisfaction. In: 5th Workshop on Soft Constraints (Soft 2003), Kinsale, Ireland (2003)
17. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: Hard and easy problems. In: Proceedings of the International Joint Conference on Artifical Intelligence (IJCAI), pp. 631–639 (1995)
18. Tompkins, D., Hoos, H.: Ubcsat: An implementation and experimentation environment for sls algorithms for SAT and max-SAT. In: Hoos, H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 306–320. Springer, Heidelberg (2005)
19. Vazirani, V.: Approximation algorithms. Springer, Heidelberg (2001)
20. Weihe, K.: Covering trains by stations or the power of data reduction. In: Proceedings of Algorithms and Experiments (ALEX 1998), pp. 1–8 (1998)
21. Xing, Z., Zhang, W.: Maxsolver: An efficient exact algorithm for (weighted) maximum satisfiability. Artificial Intelligence (AI) 164, 47–80 (2005)