Exploiting the Power of MIP Solvers in MAXSAT

Jessica Davies and Fahiem Bacchus

Department of Computer Science, University of Toronto,
Toronto, Ontario, Canada, M5S 3H5
{jdavies,fbacchus}@cs.toronto.edu

Abstract. MAXSAT is an optimization version of satisfiability. Since many practical problems involve optimization, there are a wide range of potential applications for effective MAXSAT solvers. In this paper we present an extensive empirical evaluation of a number of MAXSAT solvers. In addition to traditional MAXSAT solvers, we also evaluate the use of a state-of-the-art Mixed Integer Program (MIP) solver, CPLEX, for solving MAXSAT. MIP solvers are the most popular technology for solving optimization problems and are also theoretically more powerful than SAT solvers. In fact, we show that CPLEX is quite effective on a range of MAXSAT instances. Based on these observations we extend a previously developed hybrid approach for solving MAXSAT, that utilizes both a SAT solver and a MIP solver. Our extensions aim to take better advantage of the power of the MIP solver. The resulting improved hybrid solver is shown to be quite effective.

1 Introduction

MAXSAT is an optimization version of satisfiability (SAT). Both problems deal with propositional formulas expressed in CNF. The goal of SAT is to find a setting of the propositional variables that satisfies all clauses. MAXSAT, on the other hand, tries to find a setting of the variables that maximizes the number of satisfied clauses.

MAXSAT is complete for the class FP^{NP} (the set of function problems computable in polynomial time using an NP oracle). FP^{NP} includes many practical optimization problems, and by completeness all of them can be compactly encoded into MAXSAT. Hence, MAXSAT solvers that are effective on a wide range of inputs would be able to solve a variety of practical problems through the simple device of encoding into MAXSAT. This is already the case with SAT, where many real-world problems in NP can be effectively solved by encoding into SAT and applying current SAT solvers. Work on developing widely applicable MAXSAT solvers is still ongoing, and this paper aims to make a contribution to this effort.

Many important industrial applications involve solving optimization problems, and many powerful solution techniques for such problems have been developed. Problems with Boolean or integer variables (like MAXSAT) are most often solved using sophisticated Mixed Integer Program (MIP) solvers. MIP solvers solve problems expressed as a set of linear inequalities and a linear objective function,

a representation that is more expressive than CNF. One common technique they employ is to utilize linear programming algorithms to solve the linear relaxation derived by allowing the integer variables to take on non-integral values. Cutting plane computations are then used to drive the linear relaxation towards integral solutions. The technique of cutting planes is theoretically more powerful than resolution [5], and thus these solvers potentially have access to more powerful inference methods than standard SAT solvers. In contrast, current MAXSAT solvers have almost exclusively used resolution-based SAT technology.

In this paper we perform an extensive empirical evaluation of a number of previous solvers. Our evaluation uses many more problem instances than any previously reported study, in part because we are interested in widely applicable MAXSAT solvers. We also evaluate the performance of a state-of-the-art MIP solver, IBM's CPLEX system, on these instances. Our evaluation, reported on in Sec. 3, provides a number of interesting insights. For example, we show that CPLEX is a very effective solver for MAXSAT. We also show that in the current state-of-the-art, the notion of a single best algorithmic approach for solving MAXSAT is suspect, as is the notion of a single best solver. Our experiments do however indicate that the solvers tested can be divided into two subsets with one subset arguably dominating the other in terms of performance. However, within the high performance subset no single solver dominates.

This variance in performance among the different solvers across the problem instances indicates that each of these solvers embeds ideas that are effective on some problems. Hence, one possible direction for future research is to investigate ways of combining these ideas to uncover new algorithmic insights.[1] In previous work we had developed such a hybrid approach, MAXHS [7], that utilized a MIP solver, CPLEX, along with a SAT solver, MINISAT [8]. Each solver was given a subset of the MAXSAT problem, and information was communicated between the solvers so as to solve the combined problem. The strong performance of CPLEX in our experiments lead us to investigate ways of taking better advantage of the MIP solver. In particular, in the second part of the paper, reported on in Sec. 4, we develop a number of techniques for increasing the amount and effectiveness of information supplied to CPLEX, thus allowing it to make stronger inferences. Our new techniques yield a considerable performance improvement to the MAXHS solver (Sec. 6), and as shown in Sec. 3 the resulting improved MAXHS+ solver is clearly placed in the set of top performing MAXSAT solvers. We conclude the paper with some ideas for further work.

2 Background

In this paper we address **weighted partial MAXSAT** problems (WPMS). This is the most general type of MAXSAT problem and it includes as special cases all of the other types of MAXSAT problems studied in the literature. (All of the

[1] Our empirical evaluation shows that many instances remain unsolvable by any solver. Hence, although portfolio approaches could yield useful performance improvements, significant advances will also require new algorithmic ideas.

solvers we experiment with can solve all of these special cases as well as general WPMS problems). WPMS problems are CNF formulas in which some clauses are classified as being hard while others are classified as being soft. Any solution must satisfy all of the hard clauses, but can falsify the soft clauses. However, each soft clause has a weight and a truth assignment will incur a penalty or cost equal to the clause weight if it falsifies that clause.

More formally, a MAXSAT problem $\mathcal{F}$ is specified by a CNF formula in which each clause has an associated weight.[2] Let $wt(c)$ denote the weight of clause c. We require that $wt(c) > 0$ for every clause.[3] If $wt(c) = \infty$ we say that c is a **hard** clause, otherwise $wt(c) < \infty$ and c is a **soft** clause. We use $hard(\mathcal{F})$ to indicate the hard clauses of $\mathcal{F}$ and $soft(\mathcal{F})$ to denote the soft clauses. Note that $\mathcal{F} = hard(\mathcal{F}) \cup soft(\mathcal{F})$.

We define the function *cost* as follows: (a) if H is a set of clauses then $cost(H)$ is the sum of the weights of the clauses in H ($cost(H) = \sum_{c \in H} wt(c)$); and (b) if π is a truth assignment to the variables of $\mathcal{F}$ then $cost(\pi)$ is the sum of the weights of the clauses falsified by π ($\sum_{\{c \mid \pi \not\models c\}} wt(c)$).

A **solution** to the MAXSAT problem $\mathcal{F}$ is a truth assignment π to the variables of $\mathcal{F}$ with minimum cost that satisfies all of the clauses in $hard(\mathcal{F})$. We let $mincost(\mathcal{F})$ denote the cost of a solution to $\mathcal{F}$. If $hard(\mathcal{F})$ is UNSAT then $\mathcal{F}$ has no solution. Testing for this case is simply a SAT problem, hence from here on we will **assume that** $hard(\mathcal{F})$ **is satisfiable**.

A **core** κ for a MAXSAT formula $\mathcal{F}$ is a subset of $soft(\mathcal{F})$ such that $\kappa \cup hard(\mathcal{F})$ is unsatisfiable. That is, all truth assignments falsify at least one clause of $\kappa \cup hard(\mathcal{F})$. Since every solution satisfies $hard(\mathcal{F})$, every solution must falsify at least one clause in κ.

A common technique in MAXSAT solving is to add a unique **blocking variable** to each soft clause. Assigning *true* to a clause's blocking variable (b-variable) immediately satisfies the clause. This allows the solver to "turn off" or relax various soft clauses as it tries to solve the MAXSAT problem.

Definition 1. *If $\mathcal{F}$ is a* MAXSAT *problem, then its* ***b-variable relaxation*** *is a* SAT *problem $\mathcal{F}^b = \{(c_i \vee b_i) : c_i \in soft(\mathcal{F})\} \cup hard(\mathcal{F})$ where all clause weights are removed. The b-variable b_i appears in the relaxed clause $(c_i \vee b_i)$ and* ***no where else*** *in $\mathcal{F}^b$.*

Each truth assignment π to the variables of $\mathcal{F}^b$ has a cost $bcost(\pi)$: if $\pi \not\models \mathcal{F}^b$ then $bcost(\pi) = \infty$, otherwise $bcost(\pi) = \sum_{b_i : \pi \models b_i} wt(c_i)$. The minimum *bcost* satisfying assignments for $\mathcal{F}^b$ correspond to solutions of $\mathcal{F}$.

Proposition 1. *$mincost(\mathcal{F}) = \min_\pi bcost(\pi)$, where the minimum is taken over all truth assignments π to the variables of $\mathcal{F}^b$. Furthermore, if π achieves a minimum value of $bcost(\pi)$, then π restricted to the variables of $\mathcal{F}$ is a solution for $\mathcal{F}$.*

[2] Only integer clause weights are used in our experiments since most MAXSAT solvers require this restriction.

[3] Clauses with weight zero can be removed from $\mathcal{F}$ without impacting the solution. Clauses with negative weight yield a different problem from MAXSAT.

The observation behind the proposition is that for π to achieve a minimum value of $bcost(\pi)$ it must set b_i to *false* whenever it satisfies the soft clause c_i.

MIP Encoding: It is simple to encode a MAXSAT instance $\mathcal{F}$ as a MIP[4]. First, the clauses of the relaxed formula $\mathcal{F}^b$ are encoded as linear inequalities, using the standard method where a clause c is converted to the linear inequality $\sum_{j:p_j \in c} p_j + \sum_{i:\neg p_i \in c}(1 - p_i) \geq 1$. For example, the clause $(x \vee y \vee \neg z \vee b_1)$ becomes the linear inequality $x + y + (1 - z) + b_1 \geq 1$. Second, the objective function is to minimize $\sum_i wt(c_i) \times b_i$. The MIP thus tries to set the propositional variables so as to satisfy all clauses of $\mathcal{F}^b$ with minimum $bcost$.

Assumption Reasoning: The SAT solver MINISAT provides an **assumption** interface to test whether a given set of literals can be extended to a satisfying assignment. MINISAT can take as input a set of assumptions $\mathcal{A}$, specified as a set of literals, along with a CNF formula $\mathcal{F}$ and then determine if $\mathcal{F} \wedge \mathcal{A}$ is satisfiable. It will return a satisfying truth assignment for $\mathcal{F} \wedge \mathcal{A}$ if one exists (this truth assignment necessarily extends $\mathcal{A}$). Otherwise it will report UNSAT and return a learnt clause c which is a disjunction of negated literals of $\mathcal{A}$. This clause has the property that $\neg c$ specifies a subset of $\mathcal{A}$ such that $\mathcal{F} \wedge \neg c$ is unsatisfiable. This means $\mathcal{F} \models c$.

2.1 Existing MAXSAT Solvers

There have been two main approaches to building MAXSAT solvers. The first approach is to perform Branch and Bound search where a lower bound is computed by exploiting the logical structure of the CNF input, e.g., [9, 14]. The second approach is to solve the MAXSAT problem as a sequence of SAT problems.

In previous work these SAT problems typically encode the decision problem: "$mincost(\mathcal{F}) = k$?". This encoding is based on adding blocking variables to the soft clauses, and then translating linear inequality constraints over the blocking variables to CNF.[5] Starting with $k = 0$, if the answer from the SAT solver is "no" (i.e., the formula is unsatisfiable), the next lowest possible value for k, k^+, is computed from information extracted from the core returned by the SAT solver. Then the decision problem $mincost(\mathcal{F}) = k^+$ is encoded as the next SAT problem to be solved. The previously computed cores are also exploited in this decision problem by requiring that at least one clause from every previously extracted core is falsified. Many variations on this concept have been recently proposed [2, 10, 1]. The main disadvantage of these approaches is that as the SAT instances that need to be solved become larger and harder as the k gets larger.

MAXHS: The MAXHS solver attempts to reduce the burden placed on the SAT solver by also employing a MIP solver (CPLEX) [7]. MAXHS's algorithm also involves solving a sequence of SAT problems, however, these SAT problems are always subsets of the original MAXSAT formula $\mathcal{F}$ and are thus usually easy for

[4] The origins of this encoding are not clear. However, it is well known.

[5] Some solvers, notably WBO [15], reason with these linear constraints directly instead of converting them to CNF.

the SAT solver to refute. MAXHS uses the assumptions mechanism of MINISAT to test subsets of $\mathcal{F}$ and derive cores. MINISAT is given the formula $\mathcal{F}^b$ and some setting of the b-variables as the assumptions. If MINISAT returns UNSAT, a clause $c = (b_{i_1} \vee \cdots \vee b_{i_k})$ such that $\mathcal{F}^b \models c$ will also be returned. Note that c will only contain positive b-variables since the b-variables only appear positively in $\mathcal{F}^b$ and thus no clause involving negative b-variables is entailed by $\mathcal{F}^b$. It is easy to see that the clause c corresponds to a core of $\mathcal{F}$.

Proposition 2. *If $\mathcal{F}^b \models (b_{i_1} \vee \cdots \vee b_{i_k})$ for some set of b-variables $\{b_{i_j}\}|_{j=1}^{k}$, then $\kappa = \{c_{i_1}, ..., c_{i_k}\}$ is a core of $\mathcal{F}$. We call $(b_{i_1} \vee \cdots \vee b_{i_k})$ a* ***core constraint****.*

Starting with an initial set of core constraints in the MIP model (Sec. 5.2), CPLEX is used to find a solution to them that minimizes the cost of the *true* b-variables. The CPLEX solution (a setting of the b-variables) is then given to MINISAT as the next set of assumptions. If MINISAT finds a satisfying solution π^b then π (its restriction to the variables of $\mathcal{F}$) is an optimal solution for $\mathcal{F}$. Otherwise MINISAT will return another core constraint that is added to the CPLEX model and the cycle is repeated.

The problem that CPLEX solves at each iteration can be interpreted as a hitting set problem: find a minimum cost collection of soft clauses sufficient to block all of the refutations (cores) that have been derived from $\mathcal{F}$ so far. In fact, the MAXHS approach is closely related to the implicit hitting set (IHS) problem as described in [12, 6]. In IHS problems one is trying to compute a minimum cost hitting set without knowing ahead of time the collection of sets that need to be hit. Instead, one is provided with an oracle that when given the current candidate hitting set, either declares the candidate to be a correct hitting set or returns a new un-hit set from the implicit collection. In the MAXHS algorithm, the cores of $\mathcal{F}$ form the collection of sets to be hit, CPLEX computes candidate hitting sets, and the SAT assumption test acts as the oracle deciding if the current candidate hitting set is correct, returning a new un-hit core if it is not. However, the SAT assumption test may take exponential time, while the oracle in IHS is assumed to run in polynomial time.

The disadvantage of the MAXHS approach is that sometimes a large number of iterations have to be performed during which CPLEX returns different hitting sets. Each of these hitting sets must be ruled out by another core, which increases the size of the MIP model.

3 Empirical Evaluation of Current MAXSAT Solvers

We performed an empirical study of nine existing MAXSAT solvers: CPLEX (version 12.2), WPM1 (with the latest 2012 improvements [1]), WPM2 (version 2 [2]), BINCD [10], WBO [15], MINIMAXSAT [9], SAT4J [4], AKMAXSAT [13], and MAXHS-Orig [7]. All of these solvers are able to solve MAXSAT in its most general form, i.e., weighted partial MAXSAT, and thus have the widest range of applicability. Our study included recently developed solvers utilizing a sequence of SAT approach (BINCD, WPM1, WPM2 and MAXHS-Orig), some older solvers (SAT4J and

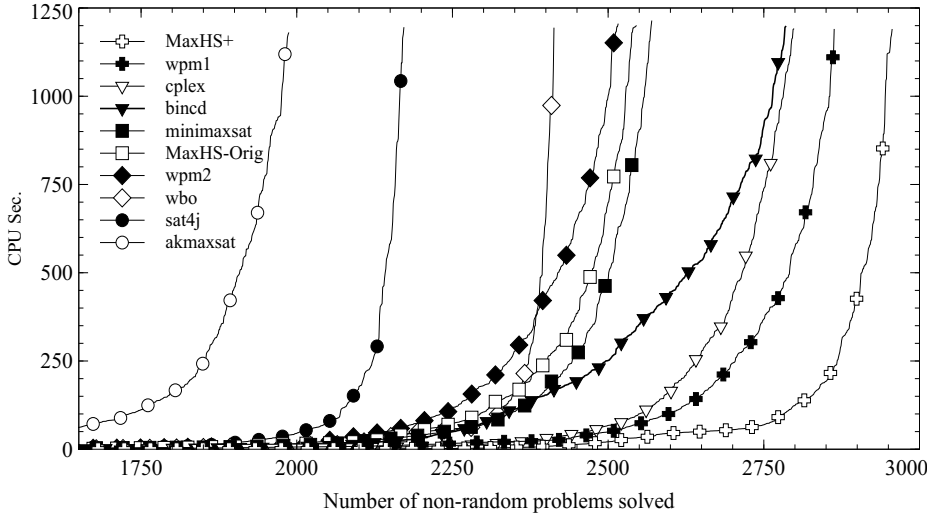

Fig. 1. Performance of solvers on all non-random problems

WBO), and two prominent Branch and Bound based solvers (AKMAXSAT and MINIMAXSAT). Also included was the MIP solver CPLEX using the encoding of MAXSAT specified in Sec. 2, and our original hybrid solver, MAXHS-Orig. We also compared against our newly developed solver MAXHS+. MAXHS+ extends the original MAXHS-Orig using the best overall combination of our newly developed techniques described in Sec. 4.

We obtained **all** problems from the previous seven MAXSAT evaluations [3]. We first discarded all instances in the Random category. After removing duplicate problems (as many as we could find) we ended up with 4502 problems divided up into 58 families. We then removed 17 of these families that in our judgement had little practical application. These included random problems, graph problems on random graphs, e.g., the maxcut, maxclique, "frb" and "kb-tree" families, and pure math problems, e.g., the Ramsey and spin glass problems.

The remaining 3826 problems either fell into the "industrial" category or were problems that we felt had application to real problems. For example, MAXSAT has applications in automated planning [17], so we kept the crafted planning problems. Similarly, the "KnotPipatsrisawat" problems involve computing MPE (most probable explanation) which is heavily used in areas like computer vision. When in doubt we erred on the side of keeping the problems, as we are in general interested in applying MAXSAT solvers as widely as possible. It should be noted that our evaluation used many more non-random problems than any previously reported evaluation (including the prior MAXSAT evaluations).

Figure 1 shows a cactus plot of the solvers running on the 3826 non-random problem we kept. Our experiments were performed on 2.1 GHz AMD Opteron machines with 98GB RAM shared between 24 cores (about 4GB RAM per core).

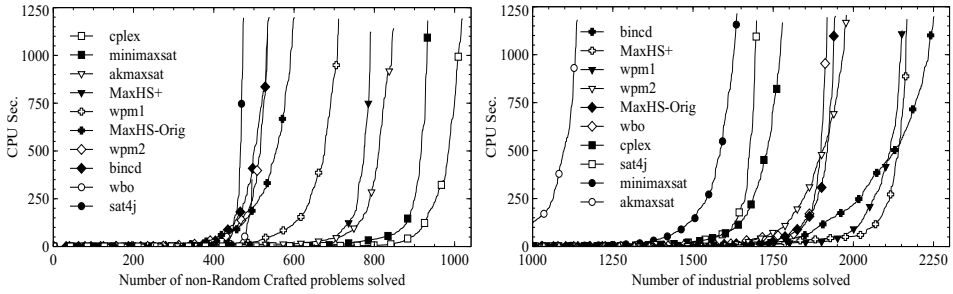

Fig. 2. Performance of solvers on Crafted and Industrial problems

Each problem was run under a 1200 sec. timeout and with a memory limit of 2.5GB. The data shows that our new solver MAXHS+ solved the most problems and that it significantly outperforms our previous solver MAXHS-Orig. The data also shows that BINCD, WPM1, and CPLEX have good performance in terms of the number of problems solved. The performance of CPLEX is particularly noteworthy. Although MIP solvers have been widely available for some time (before most MAXSAT solvers) very little has previously been reported about their performance on MAXSAT. Our data shows that CPLEX is a surprisingly good MAXSAT solver.

Figure 2 shows a break down between industrial and non-random crafted problems. It should be noted, however, that despite the labeling the non-random crafted problems also contain problems of practical (industrial) interest. The data shows that BINCD solves the most problems from the industrial class, with our new solver MAXHS+ and WPM1 having similar but not as good performance on these problems. On these problems, which tend to involve large CNF formulas, CPLEX does not perform as well. On the crafted problems the Branch and Bound solver MINIMAXSAT performed well, but interestingly CPLEX was best overall.

From this data we select MAXHS+, CPLEX, MINIMAXSAT, BINCD, and WPM1 as being in our class of top performers. These solvers dominate the others either on all problems, or on the industrial problems, or on the non-random crafted problems. Hence, we restrict our further attention to these solvers.

One bias in counting the number of problems solved is that the problem families are not equally sized. Hence, this metric will be skewed if a solver is good at solving a particular family and that family contains many problems. Table 1 shows for each of the top solvers and problem categories, the number of problems solved (from the cactus plots), the family score, and the number of families the solver was the best on. The family score for solver s is the sum over all families f of the percentage of problems in f that s solves (this attempts to normalize for the size of the family). There are 41 families in the category **All**, 22 in **Industrial**, and 19 in **Crafted**, so these are the maximum possible family scores. A solver s is best on a family f if it solves as many problems in f as any of the other top solvers. (There can be more than one solver best on a family.) Table 1 shows, e.g., that although BINCD solves more problems in the industrial

	All			Industrial			Crafted		
Solver	Solved	F-Score	F-Best	Solved	F-Score	F-Best	Solved	F-Score	F-Best
MAXHS+	**2956**	**26.75**	**20**	2165	13.98	**9**	791	12.77	**11**
WPM1	2863	25.92	13	2152	14.68	**9**	711	11.24	4
CPLEX	2798	25.69	17	1779	11.70	7	**1019**	**13.98**	10
BINCD	2785	25.38	12	**2251**	**14.97**	7	534	10.41	5
MINIMAXSAT	2570	22.80	13	1637	9.45	2	933	13.35	**11**

Table 1. The number of instances solved, Family Scores, and number of Families where each solver is best categorized by all, industrial, and crafted (non-random) problems.

category and has the highest family score, it is best on fewer families than WPM1 and MAXHS+. It also shows that although WPM1 solves almost as many crafted problems as MAXHS+ it is best on fewer families.

Finally, Table 2 shows that each of these top solvers has quite a diverse coverage. The table shows for each pair of top solvers s and s' how many problems s solves that s' fails to solve. This number is shown in the cell at row s and column s'. In fact the table contains two number in each cell. The first is the number of industrial problems s solves but s' doesn't while the second number is the number of crafted problems s solves but s' doesn't. This metric is influenced by the timeout, as s might have solved a problem in 1200 seconds that s' would have solved in 1210 seconds if it hadn't timed out. To avoid this issue, to count a problem p as solved by s and not solved by s' we require that s solves p in less than 600 seconds while s' fails to solve p.

The data shows, e.g., that on this metric CPLEX dominates MINIMAXSAT, solving more problems that MINIMAXSAT fails to solve in both the industrial and crafted categories. Similarly MAXHS+ dominates WPM1 on this metric, and BINCD dominates all other solvers on this metric for industrial problems. The main message from this data, however, is that all of these solvers dominates each other solver on some problems (typically a non-trivial number of problems).

	MAXHS+	WPM1	CPLEX	BINCD	MINIMAXSAT
MAXHS+		**191**/**208**	**420**/19	61/**264**	**547**/37
WPM1	171/121		**481**/27	89/**235**	**636**/120
CPLEX	69/**234**	128/**321**		52/**494**	**351**/**178**
BINCD	**124**/19	**169**/63	**483**/19		**544**/22
MINIMAXSAT	18/**169**	115/**342**	192/97	11/**410**	

Table 2. The entry (nI,nC) located at row i and column j shows the number of industrial problems nI (crafted problems nC) solved by solver i within 600 sec. that j fails to solve.

4 Exploiting CPLEX More Effectively

The MAXHS algorithm decomposes the MAXSAT problem into a series of SAT problems and hitting set problems. Neither the SAT solver nor CPLEX alone has enough information to solve the entire MAXSAT problem, since the SAT solver does not have any information about the clause weights, and the CPLEX model, which is only over b-variables, knows nothing about the original variables and clauses. The CPLEX model is also restricted to constraints of a specific form, i.e. core constraints which are clauses over positive b-variables. In the remainder of the paper we propose several techniques to overcome these limitations, in order to take better advantage of the MIP solver.

Many sound constraints exist over the soft clauses that do not take the form of core constraints, as illustrated by the following example.

Example 1. Let $\mathcal{F} = \{(x), (\neg x), (x \vee y), (\neg y), (\neg x \vee z), (\neg z \vee y)\}$ where each clause has weight 1. $\mathcal{F}^b$ is therefore the set of clauses $\{(b_1 \vee x), (b_2 \vee \neg x), (b_3 \vee x \vee y), (b_4 \vee \neg y), (b_5 \vee \neg x \vee z), (b_6 \vee \neg z \vee y)\}$. Suppose that the three cores $\kappa_1 = \{(x), (\neg x)\}$, $\kappa_2 = \{(\neg x), (x \vee y), (\neg y)\}$, and $\kappa_3 = \{(x \vee y), (\neg y), (\neg x \vee z), (\neg z \vee y)\}$ have been found. These cores correspond to the core constraints $\mathcal{K} = \{(b_1 \vee b_2), (b_2 \vee b_3 \vee b_4), (b_3 \vee b_4 \vee b_5 \vee b_6)\}$. We see that to satisfy these core constraints at least two b-variables in $\mathcal{F}^b$ must be set to *true*, and at least two soft clauses will be falsified by the MAXSAT solution. Given this lower bound, we can use a SAT solver to search over truth assignments that assign at most two b-variables *true*, looking for a cost-2 relaxation that satisfies $\mathcal{F}^b$. The search will benefit from the three core constraints, since they help to prune the search space. However, not all cost-2 relaxations that satisfy the core constraints need to be considered. For example, as soon as b_1 is assigned on any branch, $\neg b_2$ could be inferred because it is impossible to falsify both (x) and $(\neg x)$ at the same time. Therefore, b_1 and b_2 can not both belong to a minimum cost relaxation. Unfortunately, unit propagation in $\mathcal{F}^b \cup \mathcal{K}$ can not make this inference. Similarly, whenever $\neg b_2$ is assigned we obtain $\neg x$ and b_1 by unit propagation in $\mathcal{F}^b \cup \mathcal{K}$. However, we do not detect that $\neg b_5$ must hold as well since its soft clause is now satisfied. These two examples demonstrate that in addition to the core constraints $\mathcal{K}$, $\mathcal{F}$ also implies the constraints $(\neg b_1 \vee \neg b_2)$ and $(b_2 \vee \neg b_5)$. If these constraints could be discovered automatically, then the search over relaxations could be further constrained and potentially made more efficient.

In [7] a realizability condition was introduced. Realizability requires that there exists a truth assignment that falsifies all of the clauses in the hitting set and satisfies the hard clauses. This condition can be used to enforce some non-core constraints over the b-variables. However, it is insufficient to capture all constraints over the b-variables. For example, although the realizability condition would enforce the first non-core constraint in Example 1, $(\neg b_1 \vee \neg b_2)$, it would not capture the second, $(b_2 \vee \neg b_5)$. Therefore, we must look beyond the realizability condition for techniques to discover non-core constraints that the b-variables must satisfy.

4.1 *b*-variable Equivalences

Relaxing a soft clause in $\mathcal{F}^b$ is not equivalent to falsifying it in $\mathcal{F}$. Example 1 indicates that although the b-variables of $\mathcal{F}^b$ are intended to represent the soft clauses of $\mathcal{F}$ this correspondence is not strictly enforced by $\mathcal{F}^b$. That is, $\mathcal{F}^b$ admits models that unnecessarily set b-variables to *true* even when the corresponding soft clause is satisfied. This is the reason that the inference $\neg b_2 \rightarrow \neg b_5$ was missed in Example 1.

Proposition 1 shows, however, that minimum cost models of $\mathcal{F}^b$ do obey a stricter correspondence of equivalence between the b-variable settings and the soft clauses satisfied. Since MAXSAT solving involves searching for minimum cost models, a natural and simple modification to $\mathcal{F}^b$ is to force the b-variables to be equivalent to the negation of their corresponding soft clauses.

Definition 2. *Let $\mathcal{F}$ be a* MAXSAT *formula. Then*

$$\mathcal{F}^b_{eq} = \mathcal{F}^b \cup \bigcup_{c_i \in soft(\mathcal{F})} \{(\neg b_i \vee \neg \ell) : \ell \in c_i\}$$

is the relaxation of $\mathcal{F}$ with b-variable equivalences.

We define a correspondence between the truth assignments for $\mathcal{F}$ and the truth assignments for $\mathcal{F}^b_{eq}$.

Definition 3. *If π is a truth assignment to the variables of $\mathcal{F}$ we let π^b denote its corresponding truth assignment to the variables of $\mathcal{F}^b_{eq}$, where*

$$\pi^b = \pi \cup \{\neg b_i : \pi \models c_i, c_i \in soft(\mathcal{F})\} \cup \{b_i : \pi \not\models c_i, c_i \in soft(\mathcal{F})\}.$$

If π^b is a truth assignment to the variables of $\mathcal{F}^b_{eq}$ we let π denote its corresponding truth assignment to the variables of $\mathcal{F}$ where π is simply π^b restricted to the variables of $\mathcal{F}$.

In this definition π^b is constructed so that it assigns each b-variable a truth value equivalent to the negation of the truth value π assigns to the corresponding soft clause. Thus π^b models the b-variable equivalences. Under this correspondence we obtain a 1-1 correspondence between the models of $\mathcal{F}^b_{eq}$ and the models of $hard(\mathcal{F})$.

Proposition 3. *$\pi \models hard(\mathcal{F})$ if and only if $\pi^b \models \mathcal{F}^b_{eq}$. Furthermore, if $\pi^b \models \mathcal{F}^b_{eq}$ then $cost(\pi) = bcost(\pi^b)$, and therefore π is a solution for the* MAXSAT *formula $\mathcal{F}$ if and only if π^b achieves minimum bcost over all satisfying truth assignments for $\mathcal{F}^b_{eq}$.*

This proposition shows that we can solve the MAXSAT problem $\mathcal{F}$ by searching for a *bcost* minimal satisfying assignment to $\mathcal{F}^b_{eq}$.

Algorithm 1: A MAXSAT algorithm that exploits non-core constraints

```
1  MAXSAT-solver (F)
2  K = ∅
3  obj = wt(c_i) * b_i + ... + wt(c_k) * b_k
4  while true do
5      A = Optimize(K,obj)
6      (sat?,κ) = AssumptionSatSolver(F^b_eq, A)
       // If SAT, κ contains the satisfying truth assignment.
       // If UNSAT, κ contains a clause over b-variables.
7      if sat? then
8          break // Exit While Loop, κ is a MAXSAT solution.
       // Add new constraint to the optimization problem,
9      K = K ∪ {κ}
       // and to the SAT formula for better performance
10     F^b_eq = F^b_eq ∪ {κ}
11 return (κ, cost(κ))
```

4.2 Non-Core Constraints in MAXHS

The extension to utilize non-core constraints in MAXHS is conceptually simple. We simply substitute the encoding $\mathcal{F}^b_{eq}$ for the weaker encoding $\mathcal{F}^b$. Now since in $\mathcal{F}^b_{eq}$ the b-variables are no longer pure, the SAT solver can return both core and non-core constraints. Each constraint is passed to CPLEX which operates as before. (A copy of the learnt constraint is also kept by the SAT solver). The resulting modified version of MAXHS is shown in Algorithm 1.

Initially, the set of b-variable constraints (clauses), $\mathcal{K}$, is empty (line 2). The objective function is defined on line 3 as the sum of the clause weights for b-variables that are assigned *true*. On line 5, an assignment to the b-variables, $\mathcal{A}$, is calculated that satisfies the current constraints $\mathcal{K}$ and minimizes the value of the objective function obj. This setting of the b-variables is passed as the set of assumptions to the SAT solver on line 6, along with the SAT instance $\mathcal{F}^b_{eq}$. If the SAT solver returns UNSAT, κ will be a clause over negated literals from $\mathcal{A}$. This constraint is added to $\mathcal{K}$ on line 9 and the process iterates until the SAT solver reports a solution.

Theorem 1. *Algorithm 1 returns a solution to the* MAXSAT *problem* $\mathcal{F}$.

Proof. First we observe if the κ returned by the SAT solver at line 6 is a clause then $\mathcal{F}^b_{eq} \models \kappa$ (as explained in Section 2). On the other hand, if κ is a satisfying assignment then $bcost(\kappa)$ is equal to the sum of the costs of the *true* b-variables in $\mathcal{A}$, the solution returned by the optimizer at line 5. This follows from the fact that κ extends $\mathcal{A}$ which has already set all of the b-variables. Let κ be the satisfying truth assignment causing the algorithm to terminate. All satisfying assignments of $\mathcal{F}^b_{eq}$ satisfy the constraints in $\mathcal{K}$ as each of these is entailed by $\mathcal{F}^b_{eq}$. Furthermore, $bcost(\kappa)$ is equal to the cost of an optimal solution to these

constraints, thus κ achieves minimal *bcost* over all satisfying truth assignments for $\mathcal{F}^b_{eq}$, and by Proposition 3 κ restricted to the variables of $\mathcal{F}$ is a MAXSAT solution for $\mathcal{F}$.

Second, we observe that each iteration except the final one adds a constraint to $\mathcal{K}$ that eliminates at least one more setting of the b-variables. Since there are only a finite number of different settings, the algorithm must eventually terminate. ■

The key difference with the original MAXHS algorithm is that the optimizer no longer deals with a pure hitting set problems as the constraints can now contain negative b-variables. This means that the paradigm of MAXHS changes from an implicit hitting set problem to something like a logic based Benders decomposition approach [11]. In particular, the optimization problem is being solved only over the b-variables while the SAT solver is being used to add additional constraints to the optimization model until its solution also satisfies the feasibility conditions. Although CPLEX is no longer solving a hitting set problem, we have found that it remains very effective in the presence of non-core constraints.

5 Other Improvements

In addition to the ability to learn non-core constraints, we propose two additional techniques that help to refine the constraints and exploit the strength of CPLEX.

5.1 Constraint Minimization

The first improvement is to more aggressively minimize the constraints before adding them to the CPLEX model. In general, shorter clausal constraints are stronger, so the quality of the constraints can be improved by using techniques to minimize their length. Therefore, we ensure that the constraints we add to CPLEX are minimal, in the sense that removing any literal from the clausal constraint leaves a clause that is no longer entailed by $\mathcal{F}^b_{eq}$. We use a simple destructive MUS algorithm, as described in [16], to achieve this. Empirically, we found that the minimization computation typically takes only a small percentage of the solver's runtime, so more sophisticated MUS algorithms are unlikely to yield a significant benefit in the current solver.

5.2 Disjoint Phase

Similar to the original MAXHS solver, we also use a disjoint core phase before Algorithm 1 begins to supply CPLEX with an initial set of core constraints. During this phase we run the SAT solver on $\mathcal{F}^b$ (rather than $\mathcal{F}^b_{eq}$) so that only core constraints are derived. Initially, the SAT solver is run under the assumption that all b-variables are *false*. This generates a core constraint (unless the MAXSAT problem has a solution of zero cost). After minimizing that constraint we run the SAT solver again under the assumption that all of the b-variables in the core

constraints found so far are *true* and all other b-variables are *false*. This has the effect of removing all soft clauses that have participated in cores from the theory. Hence, the next core must be over a disjoint set of soft clauses. This process is repeated setting more and more of the b-variables to *true*, until the SAT solver can no longer find a contradiction. The collection of cores found are all disjoint and the corresponding linear constraints are initially added to CPLEX.

5.3 Seeding CPLEX with Constraints

Each call to CPLEX's solve routine incurs some overhead so it is desirable to reduce the number of calls to CPLEX. We propose to accomplish this by "seeding" the CPLEX model with a number of initially computed b-variable constraints. In this way each candidate solution (setting of the b-variables) returned by CPLEX is more informed about the constraints of the problem and thus more likely to be a true solution. We perform seeding after the disjoint core phase, but before the iterations of Algorithm 1 begin. We now describe several techniques to cheaply identify such additional b-variable constraints.

Eq-Seeding: In $\mathcal{F}^b_{eq}$, literals that appear in soft *unit* clauses of $\mathcal{F}$ are actually logically equivalent to their b-variables. To see this, recall that if $c_i = (x) \in \mathit{soft}(\mathcal{F})$ is a soft unit clause, then $\mathcal{F}^b_{eq}$ will contain clauses $(x \vee b_i)$ and $(\neg b_i \vee \neg x)$. These two clauses together imply that $b_i \equiv \neg x$. For a clause c of $\mathcal{F}^b$, if each variable in c has an equivalent b-variable (or is itself a b-variable), then we can derive a new constraint from c by replacing every original variable by its equivalent b-variable. This constraint is a clause over the b-variables that can be added to CPLEX.

Example 2. In Example 1, $b_1 \equiv \neg x$ due to the soft unit clause (x) and its relaxation by b_1. Similarly, $b_4 \equiv y$. Therefore, from the relaxed clause $(b_3 \vee x \vee y) \in \mathcal{F}^b$ we can obtain the b-variable constraint $(b_3 \vee \neg b_1 \vee b_4)$ by simply substituting the equivalent b-variable literals for the original literals.

Imp-Seeding: In $\mathcal{F}^b_{eq}$, each of the b-literals may imply other b-literals. We perform a trial unit propagation on each b-literal b_i in order to collect a set of implied b-literals $\mathit{imp}(b_i) = \{b_i^1, ..., b_i^k\}$. This represents a conjunction of k binary clauses $b_i \rightarrow b_i^j$ $(1 \leq j \leq k)$ over the b-variables. Although these k clauses could be individually added to CPLEX we can in fact encode their conjunction in a single linear constraint that can be given to CPLEX:

$$-k \times b_i + b_i^1 + \cdots + b_i^k \geq 0$$

Note that these are b-literals, so as is standard a negative literal b is encoded as $(1 - \mathit{var}(b))$ and a positive literal is encoded as $\mathit{var}(b)$ (Sec. 2). To understand this constraint note that if b_i is *true* (equal to 1) then all of the b_i^j variables must be 1 to make the sum non-negative.

Imp+Rev-Seeding: During the trial unit propagation of each b-literal b_i, we can also keep track of every original literal that is found to be implied by b_i in

order to obtain sets of reverse implications: $rev(x) \subseteq \{b_i : \mathcal{F}^b_{eq} \wedge b_i \models x\}$. Then, for each clause $c_i \in \mathcal{F}^b$, we check if each of its original literals $x \in c_i$ has a non-empty $rev(\neg x)$. If so, a b-literal $b_{\neg x} \in rev(\neg x)$ is chosen for each x and its negation $\neg b_{\neg x}$ is substituted for x in c_i. The result is a new clause containing only b-variables that can be added to CPLEX. It is easy to see that this clause is sound by considering the following example.

Example 3. Suppose that $(x \vee y \vee b_1) \in \mathcal{F}^b$ where x, y are original literals and b_1 is a blocking variable. Suppose that $b_{\neg x} \in rev(\neg x)$ and $b_{\neg y} \in rev(\neg y)$. This means that clauses $(\neg b_{\neg x} \vee \neg x)$ and $(\neg b_{\neg y} \vee \neg y)$ are implied by $\mathcal{F}^b_{eq}$. Therefore $(\neg b_{\neg x} \vee \neg b_{\neg y} \vee b_1)$, which can be obtained in two resolution steps, is also implied by $\mathcal{F}^b_{eq}$ and can be added to CPLEX.

Since the b-literal implications $imp(b_i)$ are also available, we add the Imp-Seeding constraints as well if we are computing the Rev-Seeding constraints. Note that if $b \equiv x$, as in Eq-Seeding, we obtain at least as many seeded constraints as would be obtained by Eq-Seeding. If $rev(\neg x)$ contains more than one b-literal, we could choose any one of them to form the new clause. We simply use an equivalent b-literal if one exists, and otherwise we choose the first b-literal that was found to imply $\neg x$. In future work we could investigate different ways of choosing the member of $rev(\neg x)$, or methods for using them all.

6 Empirical Evaluation of Proposed MAXHS Improvements

In this section we examine the empirical behaviour of the improvements to the MAXHS algorithm proposed above. Our experiments with the original MAXHS algorithm showed that it spent most of its time in the MIP solver and relatively little in the SAT solver. In the improved versions of MAXHS, the MIP solver still dominates the CPU time. However, seeding and our other techniques provide better information to CPLEX, which means that fewer calls are required to converge on a solution.

We ran a number of different versions of MAXHS on the problems described in Sec. 3. Figure 3 shows a cactus plot of their performance running on all non-random problems. The data shows a number of things. First, adding core minimization (+min on the plot) yields a significant performance gain compared to the original MAXHS solver. When we add to this version the ability to generate non-core constraints via the $\mathcal{F}^b_{eq}$ relaxation (+noncore), there is another jump in the number of problems solved.

If we seed CPLEX with extra constraints (Eq, Imp and Imp+Rev Seeding) in addition to the previous two techniques, performance improves again. However, there is relatively little to choose in overall performance between the different types of CPLEX seeding we developed. When we looked at the time taken to solve various problems we did find that on some instances the more extensive seeding (Imp+Rev) yields a factor of 10 improvement in solving time. However, on some

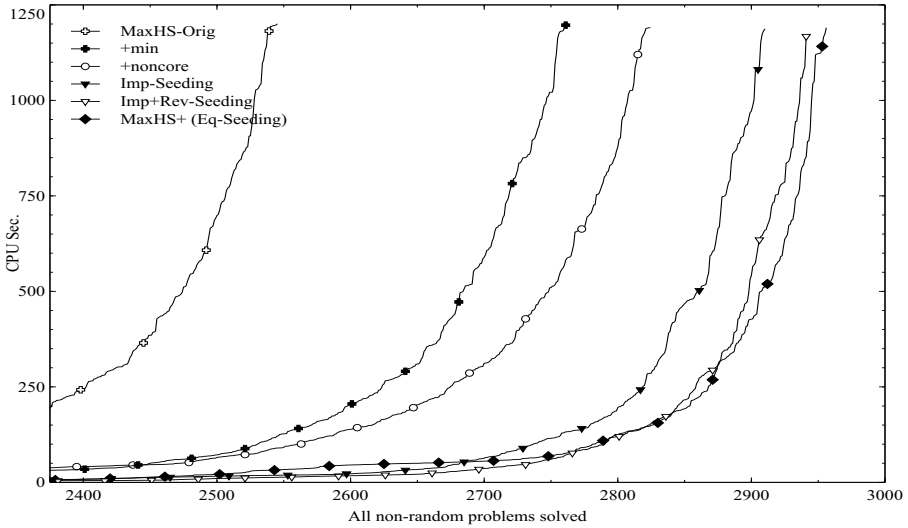

Fig. 3. Performance of MAXHS variants on all non-random problems

problems such seeding adds a very large number of additional constraints to CPLEX, without much improvement in the solution candidates produced. These extra constraints sometimes produce an increase in CPLEX's runtime sufficient to cause a time-out. Future work will require examining particular families or problem instances to obtain understanding of the trade-offs involved with our different levels of seeding.

MAXHS+ The seeding method with a slight advantage in terms of overall number of problems solved is Eq-Seeding. We referred to the configuration that uses Eq-Seeding (as well as non-core constraints, minimization, etc.) as MAXHS+ throughout the paper.

7 Conclusion

We made two main contributions in this paper. First we have reported on the results of an extensive evaluation of current MAXSAT solvers. These results provide a number of insights into current solvers. Second, inspired by one of these insights, that the MIP solver CPLEX is more effective than expected, we developed a number of new techniques aimed at enhancing our previously developed hybrid MAXSAT solver MAXHS. These techniques were mainly aimed at improving the information given to CPLEX so as to better exploit it.

In future work we aim to find out if some of the techniques used in other solvers, e.g., the binary search used in BINCD and the weight stratification method used in WPM1, which help them solve a number of distinct problem instances, can be exploited in our framework. We also plan to investigate the trade-offs we observed with the different types of seeding in more detail.

References

1. Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving sat-based weighted maxsat solvers. In: Principles and Practice of Constraint Programming (CP). pp. 86–101 (2012)
2. Ansótegui, C., Bonet, M.L., Levy, J.: A new algorithm for weighted partial maxsat. In: Proceedings of the AAAI National Conference (AAAI). pp. 3–8 (2010)
3. Argelich, J., Li, C.M., Manyà, F., Planes, J.: The maxsat evaluations (2007–2011), `http://www.maxsat.udl.cat`
4. Berre, D.L., Parrain, A.: The sat4j library, release 2.2. JSAT 7(2-3), 59–6 (2010)
5. Buss, S.R.: Lectures on proof theory. Tech. rep., `http://www.math.ucsd.edu/~sbuss/ResearchWeb/Barbados95Notes/reporte.pdf` (1996)
6. Chandrasekaran, K., Karp, R., Moreno-Centeno, E., Vempala, S.: Algorithms for implicit hitting set problems. In: Proceedings of the Symposium on Discrete Algorithms (SODA). pp. 614–629 (2011)
7. Davies, J., Bacchus, F.: Solving maxsat by solving a sequence of simpler sat instances. In: Principles and Practice of Constraint Programming (CP). pp. 225–239 (2011)
8. Eén, N., Sörensson, N.: An extensible sat-solver. In: Proceedings of Theory and Applications of Satisfiability Testing (SAT). pp. 502–518 (2003)
9. Heras, F., Larrosa, J., Oliveras, A.: Minimaxsat: An efficient weighted max-sat solver. Journal of Artificial Intelligence Research (JAIR) 31, 1–32 (2008)
10. Heras, F., Morgado, A., Marques-Silva, J.: Core-guided binary search algorithms for maximum satisfiability. In: Proceedings of the AAAI National Conference (AAAI). pp. 36–41 (2011)
11. Hooker, J.N.: Planning and scheduling by logic-based benders decomposition. Operations Research 55(3), 588–602 (2007)
12. Karp, R.M.: Implicit hitting set problems and multi-genome alignment. In: Combinatorial Pattern Matching. Lecture Notes in Computer Science, vol. 6129, p. 151 (2010)
13. Kügel, A.: Improved exact solver for the weighted Max-SAT problem. In: Workshop on the Pragmatics of SAT (2010)
14. Li, C.M., Manyà, F., Mohamedou, N.O., Planes, J.: Resolution-based lower bounds in maxsat. Constraints 15(4), 456–484 (2010)
15. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted boolean optimization. In: Proceedings of Theory and Applications of Satisfiability Testing (SAT). pp. 495–508 (2009)
16. Silva, J.P.M., Lynce, I.: On improving mus extraction algorithms. In: Proceedings of Theory and Applications of Satisfiability Testing (SAT). pp. 159–173 (2011)
17. Zhang, L., Bacchus, F.: Maxsat heuristics for cost optimal planning. In: Proceedings of the AAAI National Conference (AAAI) (2012)