Relaxation search: a simple way of managing optional clauses

Fahiem Bacchus, Jessica Davies, Maria Tsimpoukelli, George Katsirelos

Relaxation Search: a Simple Way of Managing Optional Clauses

Fahiem Bacchus Jessica Davies Maria Tsimpoukelli
University of Toronto, Canada
[fbacchus|jdavies]@cs.toronto.edu
maria.tsimpoukelli@gmail.com

George Katsirelos
MIAT, INRA, Toulouse, France
george.katsirelos@toulouse.inra.fr

Abstract

A number of problems involve managing a set of optional clauses. For example, the soft clauses in a MAXSAT formula are optional—they can be falsified for a cost. Similarly, when computing a Minimum Correction Set for an unsatisfiable formula, all clauses are optional—some can be falsified in order to satisfy the remaining. In both of these cases the task is to find a subset of the optional clauses that achieves some criteria, and whose removal leaves a satisfiable formula. Relaxation search is a simple method of using a standard SAT solver to solve this task. Relaxation search is easy to implement, sometimes requiring only a simple modification of the variable selection heuristic in the SAT solver; it offers considerable flexibility and control over the order in which subsets of optional clauses are examined; and it automatically exploits clause learning to exchange information between the two phases of finding a suitable subset of optional clauses and checking if their removal yields satisfiability. We demonstrate how relaxation search can be used to solve MAXSAT and to compute Minimum Correction Sets. In both cases relaxation search is able to achieve state-of-the-art performance and solve some instances other solvers are not able to solve.

Introduction

Over-constrained (unsatisfiable) CNF formulas are often dealt with by regarding some of the clauses as optional. When faced with such formulas we are often interested in finding a subset of the optional clauses whose removal makes the formula satisfiable. MAXSAT and computing Minimal Correction Sets (MCS) are two problems that can be cast this way. In general, relaxation search can potentially be applied to any problem involving optional clauses.

Both MAXSAT and MCS are important problems with a range of practical applications. MAXSAT is the optimization version of satisfiability and many industrial applications can be naturally encoded in MAXSAT. As a result, MAXSAT solvers have been successfully applied in various areas including bioinformatics (Strickland, Barnes, and Sokol 2005; Graça et al. 2012), planning (Cooper et al. 2006; Zhang and Bacchus 2012), and scheduling (Vasquez and Hao 2001). Computing MCSes finds application in areas like diagnosis (Reiter 1987), computing minimal models (Ben-Eliyahu and Dechter 1996; Soh and Inoue 2010), and other general areas of formal verification.

In MAXSAT we are looking for a minimum cost set of optional clauses to remove, whereas in MCS we are looking for a minimal set of optional clauses to remove. Some of the techniques that have been proposed in the literature for solving these problems, e.g., (Davies and Bacchus 2011; Marques-Silva et al. 2013), involve a two phase search: first we search for an appropriate subset of optional clauses to remove, and then we search for a satisfying assignment of the remaining formula. The search for a satisfying assignment is normally done with a standard CDCL SAT solver (Marques-Silva, Lynce, and Malik 2009), while the search for a subset of optional clauses to remove may use a range of techniques. A key, however, to making this sort of two phase search work is to find mechanisms for using information gathered in each phase to guide the search in the other phase. For example, in (Davies and Bacchus 2011) the SAT solver returns unsatisfiable cores to guide the search for a subset of optional clauses to remove.

In this paper we propose a very simple approach for unifying these two search problems so that they can solved in a more tightly integrated manner. In particular, relaxation search is a simple way of using a SAT solver so that it can search for the right set of optional clauses to remove, and in the same search tree, also test the satisfiability of the remaining clauses. Relaxation search naturally achieves a tight integration between these two phases of search and is able to automatically exploit clause learning to transfer information between the two phases. Furthermore, it is easy to implement, in some cases requiring only a change to the SAT solver's variable selection heuristic. Our empirical results show that the paradigm of relaxation search can achieve state-of-the-art performance, both for solving MAXSAT and for computing MCSes.

Relaxation Search

Let $\mathcal{F}$ be a formula expressed in CNF that is partitioned into a set $\mathcal{O}$ of optional clauses and a set $\mathcal{M}$ of mandatory clauses: $\mathcal{F} = \mathcal{O} \cup \mathcal{M}$.

Definition 1 (Relaxation Search Problem) *Let $\mathcal{R}$ be a relation between a set of clauses and a CNF formula F. The* ***relaxation search problem*** *is to find a subset S of optional* *clauses, $S \subseteq \mathcal{O}$, such that $\mathcal{F} - S$ is satisfiable and $\mathcal{R}(S, F)$ is true.*

For computing MCSes, $\mathcal{R}$ expresses a condition of minimality: $\mathcal{R}(S, \mathcal{F})$ if and only if for any set $S' \subseteq \mathcal{F}$ we have that $S' \subset S$ implies that $\mathcal{F} - S'$ is unsatisfiable. That is, S is a minimal set of clauses whose removal makes $\mathcal{F}$ satisfiable. For solving MAXSAT, we associate a weight $wt(c)$ with every clause c in $\mathcal{O}$, and take the weight of any set of clauses S, $wt(S)$, to be the sum of the weights of its clauses: $wt(S) = \sum_{c \in S} wt(c)$. The condition $\mathcal{R}(S, \mathcal{F})$ then becomes a condition of minimum weight required to achieve satisfiability: $\mathcal{R}(S, \mathcal{F})$ if and only if for any set $S' \subseteq \mathcal{O}$ we have that $wt(S') < wt(S)$ implies that $\mathcal{F} - S'$ is unsatisfiable.

Relaxation Search can be used to solve Relaxation Search Problems. It employs the standard technique of adding selector (or blocking) variables to the optional clauses. We define two different ways of using selector variables with optional clauses.

Let $\mathcal{O}^b$ be a new set of clauses formed by adding a new and different positive literal b_i to every clause $c_i \in \mathcal{O}$: $\mathcal{O}^b = \{(c_i \vee b_i) | c_i \in \mathcal{O}\}$. For $(c_i \vee b_i) \in \mathcal{O}^b$ let $eq((c_i \vee b_i))$ be the set of binary clauses $\{(\neg \ell_j, \neg b_i) | \ell_j \in c_i\}$. The clause $(c_i \vee b_i)$ enforces the condition $\neg c_i \rightarrow b_i$, while the addition of eq enforces the condition $\neg c_i \leftrightarrow b_i$. We define

$$\begin{aligned} \mathcal{F}^b &= \mathcal{M} \cup \mathcal{O}^b \\ \mathcal{F}^b_{eq} &= \mathcal{M} \cup \mathcal{O}^b \cup \textstyle\bigcup_{(c_i \vee b_i) \in \mathcal{O}^b} eq((c_i \vee b_i)) \end{aligned}$$

Both $\mathcal{F}^b$ and $\mathcal{F}^b_{eq}$ are relaxed versions of $\mathcal{F}$. If the mandatory clauses of $\mathcal{F}$ are satisfiable, then both $\mathcal{F}^b$ and $\mathcal{F}^b_{eq}$ are satisfiable even if $\mathcal{F}$ is not. $\mathcal{F}^b$ allows arbitrary subsets of $\mathcal{O}$ to be "removed" from the formula: if b_i is set to true then optional clause c_i is satisfied and no longer constrains the formula. $\mathcal{F}^b_{eq}$ encodes a tighter link between the optional clauses and the blocking variables. In particular, for any model π of $\mathcal{M}$ there is a unique model of $\mathcal{F}^b_{eq}$ obtained by setting each b_i to the truth value of $\neg c_i$ under π.

In $\mathcal{F}^b$ or $\mathcal{F}^b_{eq}$ any setting of the b variables removes or sets the truth value of the optional clauses. Let $\mathcal{E}|_\rho$ be the reduction of a CNF formula $\mathcal{E}$ by a set of literals ρ. $\mathcal{E}|_\rho$ is standardly defined to be the removal from $\mathcal{E}$ of all clauses that contain some literal of ρ and the removal from the remaining clauses of all literals ℓ such that $\neg\ell \in \rho$.

If ρ includes either b or $\neg b$ for every b variable and no other literals, then $\mathcal{F}^b|_\rho \subseteq \mathcal{F}$. All mandatory clauses are preserved, and for all $b_i \in \rho$ the clause $(c_i \vee b_i)$ is removed while for $\neg b_i \in \rho$ the clause $(c_i \vee b_i)$ is reduced back to c_i ($c_i \in \mathcal{F}$). Hence, $\mathcal{F}^b|_\rho$ is a relaxation of $\mathcal{F}$, i.e., it is a simpler less constrained formula. We also say that $\mathcal{F}^b_{eq}|_\rho$ is a relaxation, although in this case it is not necessarily a less constrained formula as $b_i \in \rho$ requires that all of the literals in c_i be falsified. Nevertheless, in both cases, by reducing $\mathcal{F}^b$ or $\mathcal{F}^b_{eq}$ via some setting of the b variables we generate a new formula that is satisfiable if and only if a particular subset of the soft clauses can be satisfied (and for $\mathcal{F}^b_{eq}$, while the remaining are falsified).

Definition 2 (Relaxation Search) *Given an over–constrained problem $\mathcal{F}$ containing optional clauses, **relaxation search** is the simple idea of solving $\mathcal{F}^b$ or $\mathcal{F}^b_{eq}$ with a standard CDCL SAT solver using the additional restriction that the solver **always chooses to branch on a b variable** if one remains unassigned.*

With relaxation search the SAT solver explores a search tree in which at the top of the search tree all decisions will be b variables, and every path in the top of the search tree selects a particular subset of $\mathcal{O}$ to relax, while the subtree below tests if that relaxation rendered $\mathcal{F}$ satisfiable. Let π be a satisfying assignment of $\mathcal{F}^b$ or $\mathcal{F}^b_{eq}$, and let $relax(\pi)$ be the set of optional clauses that have been relaxed by π, i.e., $relax(\pi) = \{c_i | (c_i \vee b_i) \in \mathcal{O}^b \text{ and } \pi \models b_i\}$.

Then π will be a solution to the relaxation search problem if $\mathcal{R}(relax(\pi), \mathcal{F})$ holds, i.e., if the relation we are interested in satisfying, $\mathcal{R}$, holds for the subset of $\mathcal{O}$ relaxed by π and $\mathcal{F}$. In particular, in addition to $\mathcal{R}(relax(\pi), \mathcal{F})$, π also demonstrates that $\mathcal{F} - relax(\pi)$ is satisfiable, (π satisfies $\mathcal{M}$ and every clause $c_i \in \mathcal{O}$ that it doesn't relax).

The decision to use $\mathcal{F}^b$ or $\mathcal{F}^b_{eq}$ depends on the relation $\mathcal{R}$. With $\mathcal{F}^b_{eq}$ we have that every satisfying assignment π falsifies every clause in $relax(\pi)$. Relaxation search can only find sets $relax(\pi)$ for satisfying assignments π. So $\mathcal{F}^b_{eq}$ might restrict our search for a solution or even make it impossible to find one if, e.g., $\mathcal{R}(S, \mathcal{F})$ implies that the clauses in S cannot be simultaneously falsified. However, in many cases, including MAXSAT and MCS, $\mathcal{R}(S, \mathcal{F})$ embodies a minimality condition, i.e., $\mathcal{R}(S, \mathcal{F})$ and $\mathcal{F} - S$ satisfiable implies that S cannot be made any smaller and still render $\mathcal{F}$ satisfiable. In these cases $\mathcal{F}^b_{eq}$ can be used and its extra constraints can make the search more efficient. In particular, in these cases if π satisfies $\mathcal{F}^b$ and $\mathcal{R}(relax(\pi), \mathcal{F})$ then π must falsify all clauses in $relax(\pi)$ (otherwise $relax(\pi)$ could be made smaller and still render $\mathcal{F}$ satisfiable) and so π must also satisfy $\mathcal{F}^b_{eq}$.

Minimal Correction Sets

Now we show how appropriate control over the b-variable decisions allows relaxation search to compute MCSes.

Definition 3 *Given an unsatisfiable formula $\mathcal{F}$ expressed in CNF, a **correction subset** is a set of clauses $C \subseteq \mathcal{F}$ such that $\mathcal{F} - C$ is satisfiable. A correction subset C is minimal (a MCS) if for any proper subset C' of C, $\mathcal{F} - C'$ remains unsatisfiable.*

In other words an MCS "corrects" $\mathcal{F}$ making it satisfiable, and it is a minimal correction in that removing anything less than the full MCS fails to make $\mathcal{F}$ satisfiable.

A useful fact about MCSes is the following.

Proposition 1 *If C is an MCS of $\mathcal{F}$ and π is a truth assignment satisfying the clauses $\mathcal{F} - C$ ($\pi \models (\mathcal{F} - C)$) then π must falsify all clauses in C: $\forall c \in C . \pi \not\models c$.*

Proof: By contradiction, suppose that π satisfies some clause in C: $\exists c \in C . \pi \models c$ then $\pi \models (\mathcal{F} - C) \cup \{c\}$ and thus C is not an MCS. ◀

As discussed above this shows that for computing MCSes we can use $\mathcal{F}^b_{eq}$.

Relaxation Search for MCS

Relaxation search is very easily controlled so that it can find an MCS. All that is needed is the additional condition that all b-variable decisions must be set to *false*.

Proposition 2 *Let π be a satisfying assignment for $\mathcal{F}^b_{eq}$ generated by a CDCL SAT solver (Marques-Silva, Lynce, and Malik 2009) under the restriction that during its operation each search decision sets some b-variable to false until no unassigned b-variables remain (no restriction is placed on the remaining ordinary variable decisions or on the ordering among the b-variables). Then $relax(\pi)$ is an MCS of $\mathcal{F}$. Furthermore, if U is an MCS of $\mathcal{F}$ then there exists a satisfying assignment π that can be generated by a CDCL solver under the above restriction such that $relax(\pi) = U$.*

Proof: First we show that $relax(\pi)$ is an MCS. $\mathcal{F} - relax(\pi)$ is satisfiable: π must satisfy all mandatory clauses and, since π falsifies the b-variable of every optional clause not in $relax(\pi)$, π must make some other literal in each of these clauses true, i.e., π must satisfy all optional clauses not in $relax(\pi)$. Further, consider any clause $c_i \in relax(\pi)$. Since decided b-variables are always *false*, b_i must be forced by unit propagation at some decision level k and thus has a clausal reason r. All decisions at level k or higher are false b-variables (no decision can be an ordinary variable while there is an unset b-variable). Therefore all literals in r that are not positive b-variables can be resolved away as the negation of these literals must have been forced. By a sequence of resolution steps we can convert r to $r' = (b_i, \vee B)$ where B is a set of positive b-variables. This clause proves that the set of optional clauses corresponding to the b-variables in it (along with the mandatory clauses) is unsatisfiable (i.e., at least one of these optional clauses must be relaxed). Since the clauses corresponding to B are already in $\mathcal{F} - relax(\pi)$, we have that $(\mathcal{F} - relax(\pi)) \cup \{c_i\}$ is unsatisfiable.

Second, if U is an MCS then we can run the CDCL solver selecting the b-variables associated with the optional clauses in $\mathcal{F} - U$ as decision variables and setting them to false (skipping to the next unassigned b-variable in $\mathcal{F} - U$ if some are forced by previous decisions). Since $\mathcal{F} - U$ is satisfiable, $\mathcal{F}^b_{eq}$ must be satisfiable under these decisions. Let π be the satisfying assignment found. By Prop. 1 π must falsify all clauses in U, hence $relax(\pi) \supseteq U$. Furthermore, $\mathcal{F} - U$ is satisfiable, thus π cannot assign any b-variable from the clauses of $\mathcal{F} - U$ to *true*. Hence $U \supseteq relax(\pi)$, and $relax(\pi) = U$. ◀

The proposition shows that relaxation search can compute MCSes by the simple technique of (a) generating $\mathcal{F}^b_{eq}$, (b) making a simple modification to the variable selection heuristic of an ordinary CDCL SAT solver, and (c) running the solver to find a satisfying solution from which an MCS can be extracted.

Finding All MCSes

In some applications it is useful to find all MCSes, e.g., this allows all minimal unsatisfiable subsets (MUSes) to be computed via hitting set duality (Liffiton and Sakallah 2008). Relaxation search is easily extended to compute all MCSes by adding blocking clauses.

Let π be a satisfying assignment found by the SAT solver as described in Proposition 2. We then add to the solver the blocking clause $\bar{\pi} = \bigvee_{b_i|\pi \models b_i} \neg b_i$. This clause states that we are not allowed to relax all of the clauses in $relax(\pi)$ (the just discovered MCS) again.

At the current leaf node of the search tree (where π has been found), $\bar{\pi}$ is a conflict clause that we can use to backtrack the search. Note that this will cause the solver to go back to some decision level where a b variable was set, thus the solver will only find one satisfying assignment under the current setting of the b variables.

Proposition 3 *If for every satisfying assignment π found by the SAT solver (operating under the restriction of Prop. 2) we output $relax(\pi)$, add the blocking clause $\bar{\pi}$, and then continue the search,*[1] *the solver will output all and only MCSes of $\mathcal{F}$ without repetition, stopping when it learns an empty clause.*

Proof: First, by Prop. 2 for every unenumerated MCS U there exists a satisfying assignment π of $\mathcal{F}^b_{eq}$ such that $relax(\pi) = U$. Since U relaxes a different set of optional clauses from every previously enumerated MCS, π must satisfy all previously added blocking clauses D. Thus, the proof of Prop. 2 continues to hold and π can still be generated by the solver even though its current formula is $\mathcal{F}^b_{eq} \cup D$.

Second, if π is a satisfying assignment found by the solver operating on the augmented formula $\mathcal{F}^b_{eq} \cup D$, then the proof in Prop. 2 that $relax(\pi)$ is an MCS continues to hold. This shows that only MCSes will be outputted. In particular, π continues to satisfy $\mathcal{F} - relax(\pi)$ by the same argument. Furthermore, the clause r' used to show that no other clause c_i can be removed from $relax(\pi)$ is generated by resolution steps that do not involve the clauses of D—the clauses of D only resolve against positive b-variables and r' is generated without any such resolution steps.

Finally, since $\pi \models D$, $relax(\pi)$ cannot be the same as any previously enumerated MCS. ◀

Previous MCS Algorithms

In (Marques-Silva et al. 2013) a number of MCS algorithms are presented, including new algorithms that advance the state-of-the-art. The two best performing algorithms are CLD and ELS. ELS is an enhanced version of basic linear search, where starting with S being the set of mandatory clauses, we test each optional clause c to see whether $S \cup \{c\}$ is satisfiable. If it is, we update S by adding c, and otherwise we move on to the next optional clause. CLD also works with a truth assignment π partitioning the formula $\mathcal{F}$ into the set S of clauses that are satisfied by π, and $\mathcal{F} - S$, the clauses falsified by π (S must contain the mandatory clauses of $\mathcal{F}$). It tests the satisfiability of $S \wedge \vee_{c \in (\mathcal{F}-S)} c$. Note that $\vee_{c \in (\mathcal{F}-S)} c$ is a large disjunction of the literals appearing in the clauses falsified by π. If this formula is unsatisfiable then there is no way to satisfy S and any additional clauses of $\mathcal{F} - S$, thus $\mathcal{F} - S$ is an MCS. Otherwise, if the formula is satisfiable, then we have found a new π' which satisfies a superset of S, and we can continue with the next iteration.

[1]The blocking clauses cannot be subsequently deleted.

Marques-Silva et al. (Marques-Silva et al. 2013) suggest three enhancements to these algorithms. The first enhancement is to first find and exploit disjoint cores, the second enhancement uses *backbone literals*, and the third enhancement is to use each satisfying model found to grow the currently known satisfiable subset S.

While relaxation search does not exploit disjoint cores, there is some relationship between our relaxation search method and the second and third enhancements. The technique of backbone literals (Kilby et al. 2005) involves adding the negation of the clause c to the formula whenever c is added to the candidate MCS set. Adding $\neg c$ to the formula makes future iterations of the algorithms more efficient. In relaxation search, a clause c_i is considered for addition to the MCS when b_i is set to true. This can only happen if b_i is forced to true, and it can only be forced when every literal of c_i has already been forced to false. These forced literals improve the efficiency of the search to verify that the current b-variable settings yield an MCS.

Growing S by including all newly satisfied optional clauses also has some relationship to relaxation search when $\mathcal{F}^b_{eq}$ is used. In particular, whenever propagation satisfies an optional clause (by forcing one of its literals) the corresponding b-variable is set to false in $\mathcal{F}^b_{eq}$ (but not in $\mathcal{F}^b$). This corresponds to excluding this optional clause from the candidate MCS.

Relaxation search, however, does not operate in the same way as the algorithms described in (Marques-Silva et al. 2013). In particular, it can only force inclusion or exclusion from the MCS (i.e., force the b-variables) via propagation from the previous set of decisions. The other algorithms extract this information from complete calls to the SAT solver.

The main advantages of relaxation search over prior methods are mainly that it is conceptually simple and trivial to implement. Relaxation search also has the ability to learn clauses over the b-variables. These clauses impose constraints on what can and can't be included in the MCS and make searching for an MCS more efficient. It should be noted, however, that some of the alternate algorithms are also able to exploit clauses learnt from previous iterations (both CLD and ELS call the SAT solver on incrementally stronger formulas).

Empirical Results

We implemented the method for finding MCSes on top of MINISAT (Eén and Sörensson 2003). The solver can be run either to produce a single MCS or to enumerate all MCSes. We compared the performance of our solver on both types of tasks against all algorithms proposed in (Marques-Silva et al. 2013). We refer to relaxation search as RS and use BFD, BLS, CLD, EFD and ELS to refer to the algorithms from (Marques-Silva et al. 2013). We used the implementation of these algorithms provided by the authors of that paper, and invoked them using the recommended parameters. Additionally, we use VBS-1 to refer to the Virtual Best Solver when the choice is restricted to the algorithms from (Marques-Silva et al. 2013), and VBS-2 to denote the Virtual Best Solver that also includes RS.

For the comparison, we experimented with the set of 1343 instances used in (Marques-Silva et al. 2013), which consists of a mix of (Weighted Partial) MAXSAT and SAT instances. In the SAT instances, all clauses are considered optional, while in MAXSAT instances, only the soft clauses are optional. We ran all experiments on a cluster with 48-core AMD Opteron 6176 nodes at 2.3GHz, with 378GB of RAM under a timeout of 1800 seconds.

Generating one MCS. On the majority of instances we tested generating the first MCS was very easy for all of the algorithms. However, on 26 instances, none of the algorithms could compute a single MCS within the timeout. On the rest of the instances, RS was the most effective algorithm, as it was able to find an MCS for 20 instances for which none of the other algorithms could, and there were only 11 instances for which RS was unable to produce an MCS but at least one other algorithm could. We summarize these results in Table 1. We see that RS outperforms all other solvers, as well as VBS-1. This is confirmed by the cactus plot, shown in Figure 1, where we see that RS dominates even VBS-1. Note that the cactus plot is zoomed in to narrow range, which may hide the fact that most instances are easy for all algorithms.

	RS	BFD	BLS	CLD	EFD	ELS	VBS-1	VBS-2
RS	-	37	38	23	34	32	20	0
BFD	9	-	4	1	2	1	0	0
BLS	10	4	-	0	2	0	0	0
CLD	11	17	16	-	15	12	0	0
EFD	9	5	5	2	-	1	0	0
ELS	11	8	7	3	5	-	0	0
VBS-1	11	19	19	3	16	12	-	0
VBS-2	11	39	39	23	36	32	20	-

Table 1: Comparison on computing a single MCS. The cell at row i and column j indicates the number of instances for which solver i produced an MCS but solver j was unable to do so.

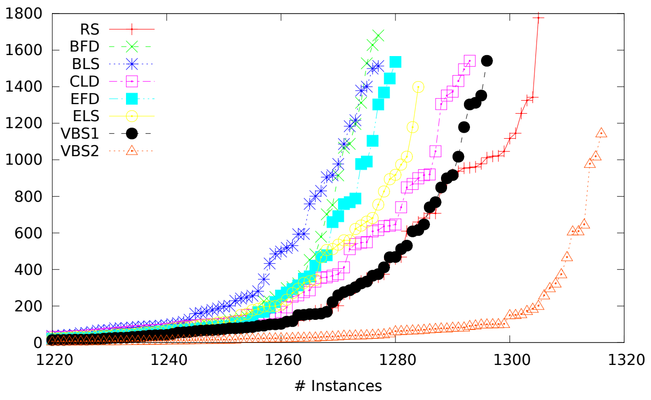

Figure 1: Cactus plot for generating one MCS. Note that the plot is zoomed into the range between 1220 and 1320 instances.

Enumerating all MCSes. For the task of enumerating all MCSes, we first observe that the situation is reversed from generating a single MCS with respect to the hardness of the instances. Here, a relatively small percentage of instances solvable by any of the algorithms (given the time limit).

In this scenario, RS is in the middle individually, but it is not significantly outperformed by any single other algorithm. Furthermore, RS causes VBS-2 to improve over VBS-1, as RS manages to generate all MCSes for 14 instances for which none of the other solvers could.

	RS	BFD	BLS	CLD	EFD	ELS	VBS-1	VBS-2
RS	-	29	26	22	29	21	14	0
BFD	21	-	1	7	3	0	0	0
BLS	23	6	-	9	7	1	0	0
CLD	28	21	18	-	22	15	0	0
EFD	22	4	3	9	-	1	0	0
ELS	24	11	7	12	11	-	0	0
VBS-1	32	26	21	12	25	15	-	0
VBS-2	32	40	35	26	39	29	14	-

Table 2: Comparison on computing all MCSes. The cell at row i and column j indicates the number of instances in which solver i produced all MCSes but solver j was unable to do so.

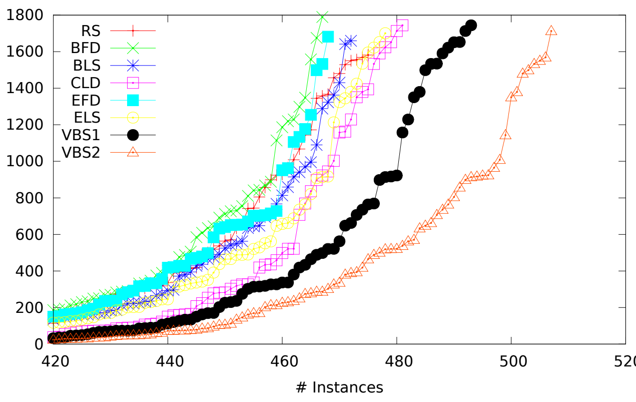

Figure 2: Cactus plot for generating all MCSes. Note that the plot is zoomed into the range between 420 and 520 instances.

Relaxation Search for MAXSAT

Relaxation search can also be used to solve MAXSAT, but it no longer suffices to find a single satisfying assignment as we did when solving MCS. To determine if the set $S \subseteq \mathcal{O}$ is an MCS we need only check the subsets S' of S to ensure that $\mathcal{F} - S'$ remains unsatisfiable. Furthermore, checking just the linear number of sets $S - c$ for all $c \in S$ suffices: if removing $S - c$ fails to achieve satisfiability then removing any subset of $S - c$ will also fail to achieve satisfiability. With MAXSAT, on the other hand, to verify that S is a minimum weight set of optional clauses whose removal achieves satisfiability, we must verify that *all* other subsets S' of lower weight (including S' unrelated to S) fail to achieve satisfiability. In the worst case, this requires checking an exponential number of alternate subsets S'. Hence, for solving MAXSAT, relaxation search must search through multiple satisfying assignments.

In MAXSAT the optional clauses are usually called *soft* clauses while the mandatory clauses are called *hard*. Associated with each soft clause c is a weight $wt(c)$, and the weight $wt(S)$ of a set of soft clauses is equal to the sum of the weights of the clauses in S. MAXSAT is the problem of finding a correction set of minimum weight.

Branch and Bound.

It is natural to extend relaxation search to handle optimization conditions $\mathcal{R}$, using Branch and Bound. First we let $wt(b_i)$, for any b-variable b_i, to be equal to the weight of its corresponding soft clause, $wt(c_i)$. If ρ is a path in the search tree then $wt(\rho)$ is simply the sum of the weights of its *true* b-variables, as each such variable corresponds to falsifying (in $\mathcal{F}^b_{eq}$) its corresponding soft clause.

A trivial lower bound on the cost of any correction set extending a path ρ is $wt(\rho)$. Given an upper bound on the weight of a minimum weight correction set, we can force a backtrack whenever the lower bound exceeds or meets the already known upper bound. Similarly, if we find a satisfying solution ρ we can update the upper bound to the new lower value $wt(\rho)$. At the completion of the search the last satisfying assignment fou ill be a solution to the MAXSAT problem.

Branch and Bound has been utilized for solving MAXSAT in a number of prior works (Heras, Larrosa, and Oliveras 2008; Li et al. 2009). The difference here is that relaxation search makes a commitment to which soft clauses to keep in the top part of each search path. This means that the bounds only need to be checked in the top part of the search tree (where b-variable decisions are made), whereas the bottom part of the search tree (where ordinary variables are decided on) can be searched using a standard CDCL search. Relaxation search has some potential advantages over previous Branch and Bound algorithms in that it can learn clauses over the b-variables which capture conditions on which sets of soft clauses can be simultaneously satisfied or falsified. Other branch and bound algorithms must deal with the soft clauses directly which limits the amount of learning they can do (Davies and Bacchus 2011).

However, better techniques for computing lower bounds would need to be developed and experimented with, in order to obtain an efficient MAXSAT solver based on relaxation search controlled by Branch and Bound.

Iterative Deepening Search.

Iterative deepening is a standard AI search technique that could be applied with relaxation search. Iterative deepening in this context corresponds to starting with a bound M that is known to be less than or equal to the true minimum cost. During relaxation search, only paths ρ with weight less than or equal to M are explored—as soon as the weight of a path becomes greater than M we backtrack. If a satisfying assignment is found it is a MAXSAT solution. Otherwise, we can increase M to the minimum weight over those paths the search backtracked from due to their weight exceeding M. (Note that other paths backtracked from because of a logical conflict need not be used to update M.)

Iterative deepening works in a manner somewhat similar to a number of popular MAXSAT algorithms like WPM1 and WPM2 (Ansótegui, Bonet, and Levy 2013). In these algorithms an upper bound is imposed on the set of b-variables that can be made true, and if the formula is still unsatisfiable this bound is increased.

A key to making iterative deepening relaxation search work efficiently, would be to find techniques to soundly keep some of the learnt clauses: only clauses learnt independently of the bound M would be sound to use in the next iteration after the bound is increased.

Logic Based Benders Relaxation Search.

One more approach to using relaxation search for MAXSAT is to use a logic based Benders approach (Hooker and Ottosson 2003). This is very similar to the approach used in the MAXHS system (Davies and Bacchus 2013), except that relaxation search is used in place of the SAT solver. We experimented with this approach and obtained good results.

This scheme utilizes a Mixed Integer Linear Programming Solver (MIP); we used IBM CPLEX. The MIP solver starts off with no constraints, a 0/1 variable corresponding to each of the b-variables, and an objective function minimizing the sum of the weights of the true b-variables. Each stage starts by asking the MIP solver for an optimal solution. Initially the optimal solution would be to set all b-variables to *false*. We then use this solution to guide relaxation search by making relaxation search set all of its b-variable decisions to the value given by the MIP solution. If this leads to a satisfying assignment then that assignment is a MAXSAT solution. Otherwise at some stage, perhaps after some search, relaxation search will force one or more of the b-variables to a value different from the MIP solution. From that forced b-variable, a clause over the b-variables can be learnt. This clause expresses a condition over the b-variables (a constraint on which soft clauses can and can't be falsified) that the MIP solver was not aware of (its returned solution violated this condition). By continuing the relaxation search we can collect some set of clauses over the b-variables that are violated by the MIP solution.

After a sufficient number of such clauses have been collected we can terminate relaxation search, give the learnt clauses as new constraints to the MIP solver, and ask it again to find an optimal solution. This new optimal solution is then used to initiate another stage of guided relaxation search, which once again tries to extend the provided setting of the b-variables to a satisfying solution.

Empirical Results

We implemented the above logic based Benders relaxation search approach for MAXSAT and tested it on all industrial and crafted instances from the 2013 MAXSAT Evaluation. We compared its performance against some successful sequence-of-SAT solvers from that evaluation: MAXHS (Davies and Bacchus 2013), WPM1, WPM2 (Ansótegui, Bonet, and Levy 2013) and BINCD (Heras, Morgado, and Marques-Silva 2011). We use VBS-1 to denote the virtual best solver of all previous algorithms and VBS-2 to denote the virtual best solver that also includes RS. We used the same hardware and resources limits as in our experiments computing MCSes.

The results show that RS is competitive with the state-of-the-art. As Table 3 shows, it is not dominated by any solver, nor does it dominate any other solver. In fact, for any pair of solvers, there is a relatively large number of instances which exactly one of the pair can solve within the timeout. That notwithstanding, the addition of RS causes VBS-2 to improve over VBS-1 by 7 instances. Moreover, as we can see in Table 4, it places near the top in total number of instances solved, a fact which is also reflected in the cactus plot in Figure 3.

Conclusion

In this paper we have presented the very simple yet flexible idea of relaxation search. Relaxation search uses two simple techniques–blocking variables and controlling a SAT solver's decisions–to combine what is usually two separate searches into one CDCL search tree. A trivial modification of the SAT solver's decisions yields an improvement in the state-of-the-art for computing minimal correction sets. We suggested a range of new approaches to solving MAXSAT, all based on utilizing relaxation search. We implemented one of these approaches, and obtained a near state-of-the-art MAXSAT solver. Our conclusion is that relaxation search offers a very flexible and easy to implement way of developing new algorithms for solving problems involving optional clauses.

	RS	MAXHS	BINCD	WPM1	WPM2	VBS-1	VBS-2
RS	-	96	472	272	267	7	0
MAXHS	116	-	443	344	264	0	0
BINCD	164	115	-	192	49	0	0
WPM1	142	194	370	-	148	0	0
WPM2	236	213	326	247	-	0	0
VBS-1	270	243	571	393	294	-	0
VBS-2	270	250	578	400	301	7	-

Table 3: Comparison on MAXSAT. The cell at row i and column j indicates the number of instances solved by solver i but not by solver j.

Category	RS	MAXHS	BINCD	WPM1	WPM2	VBS-1	VBS-2
MS crafted (167)	5	6	8	9	12	12	12
MS industrial (55)	14	5	22	21	17	23	25
PMS crafted (377)	298	300	238	178	248	306	308
PMS industrial (627)	384	446	445	377	473	550	552
WMS crafted (116)	28	33	16	10	18	35	35
WPMS crafted (340)	331	331	132	290	232	333	333
WPMS industrial (397)	295	254	186	340	324	359	360
total (2079)	1355	1375	1047	1225	1324	1618	1625

Table 4: Comparison on MAXSAT, showing the number of problems solved by different solvers by category (number of instances in category in brackets).

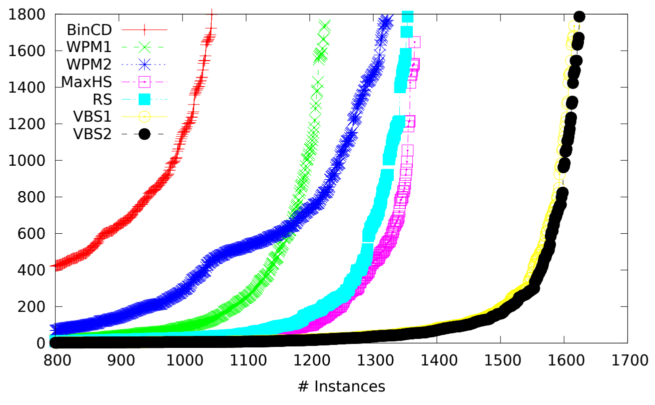

Figure 3: Cactus plot for MAXSAT. Note that the plot is zoomed into the range between 800 and 1700 instances.

References

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2013. Sat-based maxsat algorithms. *Artificial Intelligence* 196:77–105.

Ben-Eliyahu, R., and Dechter, R. 1996. On computing minimal models. *Ann. Math. Artif. Intell.* 18(1):3–27.

Cooper, M. C.; Cussat-Blanc, S.; de Roquemaurel, M.; and Regnier, P. 2006. Soft arc consistency applied to optimal planning. In *Principles and Practice of Constraint Programming (CP)*, 680–684.

Davies, J., and Bacchus, F. 2011. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Principles and Practice of Constraint Programming (CP)*, 225–239.

Davies, J., and Bacchus, F. 2013. Postponing optimization to speed up MAXSAT solving. In *Principles and Practice of Constraint Programming (CP)*, 247–262.

Eén, N., and Sörensson, N. 2003. An extensible sat-solver. In *Proceedings of Theory and Applications of Satisfiability Testing (SAT)*, 502–518.

Graça, A.; Lynce, I.; Marques-Silva, J. a.; and Oliveira, A. L. 2012. Efficient and accurate haplotype inference by combining parsimony and pedigree information. In *Proceedings of the 4th International Conference on Algebraic and Numeric Biology*, 38–56.

Heras, F.; Larrosa, J.; and Oliveras, A. 2008. MiniMaxSAT: An efficient weighted Max-SAT solver. *J. Artif. Intell. Res. (JAIR)* 31:1–32.

Heras, F.; Morgado, A.; and Marques-Silva, J. 2011. Core-guided binary search algorithms for maximum satisfiability. In *Proceedings of the AAAI National Conference (AAAI)*, 36–41.

Hooker, J. N., and Ottosson, G. 2003. Logic-based benders decomposition. *Mathematical Programming* 96(1):33–60.

Kilby, P.; Slaney, J. K.; Thiébaux, S.; and Walsh, T. 2005. Backbones and backdoors in satisfiability. In *Proceedings of the AAAI National Conference (AAAI)*, 1368–1373.

Li, C.; Manyà, F.; Mohamedou, N.; and Planes, J. 2009. Exploiting cycle structures in Max-SAT. In *Proceedings of Theory and Applications of Satisfiability Testing (SAT)*, 467–480.

Liffiton, M. H., and Sakallah, K. A. 2008. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning* 40(1):1–33.

Marques-Silva, J.; Heras, F.; Janota, M.; Previti, A.; and Belov, A. 2013. On computing minimal correction subsets. In *Proceedings of the International Joint Conference on Artifical Intelligence (IJCAI)*, 615–622.

Marques-Silva, J.; Lynce, I.; and Malik, S. 2009. Conflict-driven clause learning SAT solvers. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability*. IOS Press. 131–153.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32(1):57–95.

Soh, T., and Inoue, K. 2010. Identifying necessary reactions in metabolic pathways by minimal model generation. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 277–282.

Strickland, D. M.; Barnes, E.; and Sokol, J. S. 2005. Optimal protein structure alignment using maximum cliques. *Operations Research* 53(3):389–402.

Vasquez, M., and Hao, J.-K. 2001. A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Comp. Opt. and Appl.* 20(2):137–157.

Zhang, L., and Bacchus, F. 2012. Maxsat heuristics for cost optimal planning. In *Proceedings of the AAAI National Conference (AAAI)*, 1846–1852.