Structural Relaxations by Variable Renaming and their Compilation for Solving MinCostSAT

Miquel Ramírez[1] and Hector Geffner[2]

[1] Universitat Pompeu Fabra
Passeig de Circumvalació 8
08003 Barcelona Spain
miquel.ramirez@upf.edu
[2] ICREA & Universitat Pompeu Fabra
Passeig de Circumvalació 8
08003 Barcelona Spain
hector.geffner@upf.edu

Abstract. Searching for optimal solutions to a problem using lower bounds obtained from a relaxation is a common idea in Heuristic Search and Planning. In SAT and CSPs, however, explicit relaxations are seldom used. In this work, we consider the use of explicit relaxations for solving MinCostSAT, the problem of finding a minimum cost satisfying assignment. We start with the observation that while a number of SAT and CSP tasks have a complexity that is exponential in the treewidth, such models can be relaxed into weaker models of bounded treewidth by a simple form of variable renaming. The relaxed models can then be compiled in polynomial time and space, so that their solutions can be used effectively for pruning the search in the original problem. We have implemented a MinCostSAT solver using this idea on top of two off-the-shelf tools, a d-DNNF compiler that deals with the relaxation, and a SAT solver that deals with the search. The results over the entire suite of 559 problems from the 2006 Weighted Max-SAT Competition are encouraging: SR(w), the new solver, solves 56% of the problems when the bound on the relaxation treewidth is set to $w = 8$, while Toolbar, the winner, solves 73% of the problems, Lazy, the runner up, 55%, and MinCostChaff, a recent MinCostSAT solver, 26%. The relation between the proposed relaxation method and existing structural solution methods such as cutset decomposition and derivatives such as mini-buckets is also discussed.

1 Introduction

The idea of searching for optimal solutions to a problem using lower bounds obtained from a relaxation has been common in both Heuristic Search and Planning. In Pattern Databases, for example, certain state variables are abstracted away from the problem, and the resulting weaker problem is solved exhaustively. A table stored in memory saves the distance to the goal from any state of the relaxation which provides then a lower bound in the original problem for any state

where the value of the variables that have not been abstracted away coincides [1]. The best results for the Rubik's Cube, for example, have been obtained in this way [2]. Likewise, some of the best domain-independent planners use distance estimators obtained from an explicit relaxation where 'deletes' are dropped out of the problem [3].

In SAT and CSPs, the general idea of solving a problem by first solving a relaxation is implicit in many methods, even if explicit relaxations are seldom used. For example, node-consistency can be thought as a relaxation where non-unary constraints are excluded, and similarly, arc-consistency can be thought as iterating over relaxed problems that contain a single constraint [4].

In this work, we consider the use of explicit relaxations and their use in MinCostSAT: the problem of obtaining a minimum cost satisfying assignment of a CNF formula, where the cost of an assignment adds up the cost of the literals that are true [5]. We start with the observation that while a number of tasks over graphical models such as SAT and CSPs have a complexity that is exponential in the treewidth of the underlying interaction graph [6], such models can be relaxed into weaker models of bounded treewidth by a suitable form of variable renaming, where a variable that appears in many factors (clauses, constraints, etc.) is replaced by many fresh new variables that appear in few. Provided that the relaxed model has a bounded treewidth, the relaxation can be compiled in polynomial time and space, so that its solutions, obtained from the compiled representation without search can be used to prune the search in the original problem, very much as it is done with Pattern Databases in Heuristic Search.

Using these ideas, we have implemented a MinCostSAT solver that we call SR(w), on top of two off-the-shelf tools: the d-DNNF compiler due to Darwiche [7] and the MiniSAT 2.0 solver due to Sörensson & Eén [8]. Given a MinCostSAT problem over a CNF theory T, SR(w) maps T into the 'renamed' relaxation T^- which is compiled into d-DNNF [9], a form akin to OBDDs that supports a number of queries and transformations in polynomial time and in particular, MinCostSAT (called 'preferred models' in [10]). The compilation is exponential in the treewidth of the relaxation T^- that we control and call the *target treewidth* w. The optimal solution to the original theory T is obtained by performing a DPLL-style branch-and-bound search over T implemented on top of MiniSAT exploiting both the lower bounds obtained from the compilation of T^- and capabilities such Unit Propagation and Clause Learning. The resulting MinCostSAT solver, SR(w), is then evaluated empirically over the entire suite of problems from the 2006 Weighted Max-SAT competition [11].

2 MinCostSAT

MinCostSAT is the problem of obtaining a minimum cost satisfying truth assignment of a CNF propositional formula. If the formula is denoted by T, the cost $c(s)$ of a truth assignment s over T is defined as follows:

$$c(s) \stackrel{\text{def}}{=} \begin{cases} \sum_{l:s \models l} c(l) & \text{if } s \models T \\ \infty & \text{otherwise} \end{cases} \quad (1)$$

where l stands for literals and $c(l)$ stands for the cost of literal l. The optimal cost $c^*(T)$ of T is the cost of the best (minimum cost) assignment s. In some formulations, costs are defined only on *positive literals*, with the costs of *negative literals* assumed to be zero [12]. Non-negative costs on negative literals $\neg x$ can then be captured by introducing new positive literals x' set to $x' \equiv \neg x$. In this work we do not need this assumption and accommodate costs on both positive and negative literals. Without loss of generality, however, we assume that all costs are *non-negative*; indeed, a positive increment can always be added to the cost of any pair of complementary literals x and $\neg x$ without changing the solutions so that none has negative cost and at most one has a positive cost.

MinCostSAT is a (boolean) special case of the class of constraint optimization problems called *Valued CSPs* (VCSPs)[13]. Another special case of VCSPs closely related to MinCostSAT is Weighted MAX-SAT, where the cost of an assignment reflects the costs of the clauses that are violated by it [14]. The transformation of one into the other is simple and involves the addition of a linear number of variables or clauses, and they both subsume other variations like MAX-SAT, (Weighted) MIN-ONE, (Weighted) MAX-ONE, and (Weighted) Partial MAX-SAT [5, 15].

3 Relaxing Graphical Models by Renaming Variables

Current state-of-the-art MinCostSAT and Weighted MAX-SAT perform a branch and bound search, pruning the space by various forms of constraint propagation that take into account both the constraints and their weights [16–19]. In this work, we approach the search in MinCostSAT from a slightly different perspective. *Rather than pruning values by local consistency methods, we compute explicit lower bounds by solving optimally a global relaxation.* This relaxation is defined structurally in terms of the *constraint* or *interaction* graph that represents the interactions among the variables in the problem [6].

Relaxations by Variable Renaming

We start with the observation that any graphical model can be relaxed into a weaker model of bounded treewidth by a simple form of variable renaming, where a variable that appears in many factors is replaced by many fresh new variables that appear in few. For example, a graphical model over variables $x_1, \ldots, x_n$ where there is a factor for each variable pair, has an interaction graph that is a full clique and a treewidth equal to $n-1$. Yet, if a variable x_i is replaced by a different 'alias' variable x_i^j in each of the factors where it appears, a relaxed model is obtained with treewidth $n-2$. Actually, if the same is done for all the variables $x_1, \ldots, x_n$, the treewidth of the relaxed model is reduced to 1.

If T is the propositional MinCostSAT theory, we will refer by T^- to the relaxed theory obtained by replacing some or all occurrences of some variables x_i in T by a set of new variables x_i^j that we call 'aliases'. There is a lot of freedom in the choice of the variables x_i in T to rename, in the number x_i^j of aliases to

introduce for each renamed variable x_i, and in the scope of these aliases (i.e., the set of clauses in T where x_i is replaced by x_i^j). A *renaming scheme* defines these three aspects.

For simplicity, we consider only schemes where the scopes of the clauses including the aliases are disjoint (no occurrence of a renamed x_i is replaced by more than one alias), and span all the clauses where x_i appears (so that renamed variables x_i do not appear in the relaxation T^-). Such schemes can be described by defining the set of variables x_i in T to be renamed (relaxed) along with the alias x_i^j to be used in place of x_i in each of the clauses where x_i appears. For example, given a theory T with four clauses:

$$C_1 : x_1 \vee x_2 \ , \ C_2 : \neg x_1 \vee x_3 \ , \ C_3 : x_2 \vee \neg x_3 \ , \ C_4 : x_1 \vee x_4$$

an admissible renaming can be obtained by replacing x_2 by the alias x_2^1 in clause C_1 and by the alias x_2^2 in C_3, resulting in the relaxation T^-:

$$C_1' : x_1 \vee x_2^1 \ , \ C_2 : \neg x_1 \vee x_3 \ , \ C_3' : x_2^2 \vee \neg x_3 \ , \ C_4 : x_1 \vee x_4$$

The interaction graphs of both of these theories are displayed in Figure 1, from which it is possible to see that T has treewidth 2, while T^- has treewidth 1.

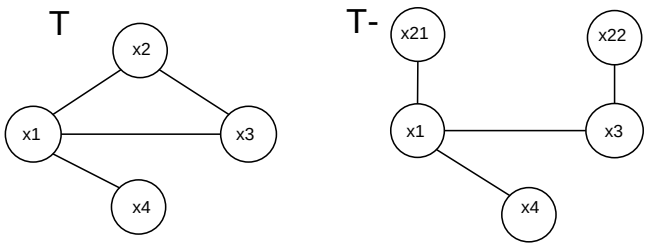

Fig. 1. The interactions graphs of T and T^- where variable x_2 in T is renamed into the aliases x_2^1 and x_2^2. The corresponding treewidths are 2 and 1

We say that variable x_i is *fully renamed* when the different occurrences of x_i in T are replaced by different aliases x_i^j in T^-.[3] When the variables are renamed fully, the only aspect that a renaming scheme must define is the set of variables to rename. For this we make use of the notion of w-cutsets: these are sets of variables x_i in T whose instantiation ensures that the (induced) treewidth of the resulting theory is bounded by w. This notion, introduced in [20], has been proposed as an alternative way for combining structural and search methods: basically, one can choose an small target treewidth w, search then exhaustively over the

[3] It is assumed that no variable in T appears twice in a given clause. If not, multiple occurrences can be collapsed into one when they all have the same sign, and else the clause is a tautology and can be deleted.

possible instantiations v of the w-cutset C_w, solving the resulting theories T_v by structural methods with a complexity exponential only on w. For a bounded w, the complexity of this method is dominated by the exhaustive search over the possible instantiations over the w-cutset variables and hence is exponential in $|C_w|$.

The method that we propose makes use of the w-cutset notion too but by *using renaming rather than instantiation for simplifying T into a tractable form*, does not require an exhaustive search over the w-cutset variables, except in the worst case.

The theory T^- obtained by *renaming fully* all the variables x_i in a w-cutset of T, can be processed indeed in time that is exponential in w, in the same way as the theory T_v that is obtained from T by *instantiating* such variables. Yet, while in the latter case the complexity bound follows from the (induced) treewidth of its interaction graph, in the former, it does not. This is because treewidth is a rough structural notion affected by the *size of the clauses,* a size that can be reduced by *instantiating* variables in a clause but not by *renaming* such variables. Finer structural measures like *hypertree width* [21] that take into account the size and scope of the constraints, can be used instead for characterizing the complexity bounds that follow from renaming. We will take however a simpler route and refer to the *simplified (induced) treewidth* of a theory T as the (induced) treewidth of its *simplified interaction graph*: this is the interaction graph of T *with the nodes representing the variables that occur in a single clause of T removed*.

Theorem 1. *Let T be a CNF formula and let S be a w-cutset for T. Then the relaxation T^- obtained by fully renaming each variable x_i in S has a* simplified treewidth *bounded by w.*

A simplified treewidth bounded by w ensures a complexity exponential in w over the typical queries in graphical models provided that variables that appear in a single factor can be eliminated in time that does not grow with the size of the factor. This is certainly true in our context for variables that appear in a single clause, as individual clauses can be compiled into d-DNNF (see below) in linear-time [9]. The alias variables x_i^j that result from fully renaming a variable x_i in T fall into this class. Note also that *simplified treewidth* of w implies *treewidth* of w when no factor has a size greater than w.

For simplicity, when no confusion arises, we will use the term treewidth to refer to simplified treewidth, and refer to the relaxation method that maps the theory T into T^- according to Theorem 1, as *w-cutset renaming*. This is a polynomial and fast operation. The theory T^- above, whose interaction graph is shown in Figure 1 is a w-cutset renaming relaxation of T, with $w = 1$ for the w-cutset $S = \{x_2\}$.

In order to compute a w-cutset, the *GWC* algorithm presented in [22] can be used. The *GWC* algorithm greedily and incrementally builds a minimal set of variables S that ensures $|C_i/S| \leq w$ over the maximal cliques C_i in the min-fill tree-decomposition of the model. In our solver, we just substitute the *min-fill*

ordering by *minimum induced width* (MIW) [6] which scales up better for large theories at a small cost in size of the cutsets.

In the experiments, we consider also an alternative relaxation method that we call *w-mini-bucket renaming* as it is based on the mini-bucket approximation scheme [23]. The w-mini-bucket renaming scheme runs the mini-buckets procedure *symbolically*. That is, propagating only the scopes of the primitive and induced functions but not the functions themselves. Each scope keeps track also of the set of primitive functions (clauses) it was derived from. If $x_1, \cdots, x_i, \cdots, x_k, \cdots, x_n$ is the ordering in which the variables are processed, then when processing the bucket of variable x_i, if the number of variables $x_k \neq x_i, k \geq i$ appearing on that bucket is above the w bound, the scopes in the bucket are partitioned into sets $s_1, \ldots, s_m$ none of which has more than w variables $x_k, k > i$. The variable x_i is then renamed into x_i^j in all the clauses that are associated with the scopes in s_j.

Like w-cutset renaming, w-mini-bucket renaming yields relaxations T^- with treewidth bounded by w, but unlike the former it does not use full variable renaming, as an alias may end up appearing in different clauses. As we will see, the experimental results for the two methods tend to be similar, with an slight edge for w-cutset renaming.

4 Compiling the Relaxation into d-DNNF

If the relaxation T^- has bounded (simplified) treewidth, then all the MinCostSAT sub-problems problems $T^- \cup s^-$ arising in the search where s^- is a set of T^- literals, can be solved from scratch in polynomial time and space by variable elimination [24]. A more efficient solution, which is more elegant too, can be obtained however by compiling the formula in a suitable way so that the solutions to these sub-problems can be efficiently 'retrieved' from a compiled representation. For formulas expressed in CNF, Darwiche's CNF to d-DNNF compiler [7] allows us to do exactly that in time and space exponential in the formula treewidth [9].

A formula T in d-DNNF is a rooted DAG (Directed Acyclic Graph) whose leaves are the positive and negative literals associated with the variables in T along with the constants **true** and **false**, and whose internal nodes stand for conjunctions or disjunctions (AND and OR nodes, respectively). A d-DNNF formula is thus in Negated Normal Form (NNF) as it contains only the connectives for conjunctions, disjunctions, and negations, and negation occurs only in literals [9]. The d-DNNF representation enforces two additional constraints, decomposability (no variable shared among sub-formulas represented by the children of an AND node) and determinism (no model shared among sub-formulas represented by the children of an OR node), that enable a large number of otherwise intractable queries and transformations to be done in time which is linear in the size of the DAG representation [25]. For example, the procedure for computing the cost of a formula T in d-DNNF can be expressed in term of the value of the

function $c^*(n)$ computed bottom up over the nodes n of the DAG as follows [10]:

$$c^*(n) = \begin{cases} 0 & \text{if } n = \textbf{true} \\ \infty & \text{if } n = \textbf{false} \\ c(L) & \text{if } n = L \text{ where } L \text{ is a literal different than } \textbf{true} \text{ and } \textbf{false} \\ \sum_i c^*(n_i) & \text{if } n \text{ is an AND node with children } n_i \\ \min_i c^*(n_i) & \text{if } n \text{ is an OR node with children } n_i \end{cases} \tag{2}$$

If the DAG represents the compilation of T^- then the cost $c^*(T^-)$ is given by the value $c^*(n)$ of the root node, while the cost $c^*(T^- \cup s^-)$ of the theory T^- extended with a set of literals s^- is obtained from the same bottom-up recursion but setting the costs $c(L)$ of the literals L whose negation is in s^- to ∞. Finally, while the costs $c^*(T^- \cup s^-)$ are obtained in a single bottom-up pass over the DAG, a minimum cost model M of T can be obtained by a subsequent top-down pass, collecting the literals in the leaves that can be reached from the root node, following all of the children of every reached AND node, and a single best child (min. cost) of every reached OR node [10].

In order to ensure that the relaxation T^- compiles in time and space exponential in the target (simplified) treewidth w set for the relaxation, we must instruct the d-DNNF compiler to use an elimination ordering over the variables in T^- (in the form of a decomposition tree [26]) related to the variable ordering used for obtaining T^- from T. For this, if T^- was obtained by w-cutset renaming, we just place the alias variables x_i^j first in the ordering, while if T^- was obtained by w-mini-bucket renaming, the aliases x_i^j are introduced in place of the renamed variables x_i in the order in which they were 'eliminated' (symbolically) during the relaxation. These elimination orders ensure that the compiler processes the relaxed theories T^- in time and space exponential in the target width w in the worst case.

5 The Cost of Renamed Variables

If there are costs $c(x_i) > 0$ or $c(\neg x_i) > 0$ associated to variables x_i in T that have been renamed in T^-, we need to decide how to allocate these costs to their aliases x_i^j. One possibility is to ignore such costs by setting the costs of all alias literals to zero. This ensures that the compilation of T^- yields lower bounds for T, but they are not as informed as they could be. Setting the costs of the all the alias literals x_i^j and $\neg x_i^j$ to the cost of the corresponding literals x_i and $\neg x_i$ can lead to overestimation and hence does not necessarily produce lower bounds. Actually, a simple option that uses the costs over renamed variables x_i and yields lower bounds is to transfer these costs to a particular, designated alias variable x_i^j, thus setting $c(x_i^j)$ and $c(\neg x_i^j)$ to $c(x_i)$ and $c(\neg x_i)$ respectively, leaving the cost of all other alias literals in zero. This is actually the choice that we have made in our solver where such designated alias variables are selected greedily using the idea of Maximal Independent Sets (MIS) [27, 12]. We have also tried an scheme where the cost of a renamed literal is split uniformly over it alias

literals, that also ensures admissibility (lower bounds), but found the results to be slightly inferior.

A MIS ω is a subset of clauses that is maximally independent in the sense that no two clauses in ω share a variable. We build a MIS greedily, starting from the set S of clauses in T^- that include some alias variable x_i^j. Then iteratively a clause C with maximum "average cost" defined in [12]:

$$\frac{\sum_{l_i \in C} c(l_i)}{|\{l_i \in C_j\}|} \tag{3}$$

is chosen and added to ω, while all clauses featuring an alias variable x_i^k coming from the same renamed variable x_i as an alias variable x_i^j in C are removed from S. This process continues until no more clauses are left in S. The alias literals x_i^j or $\neg x_i^j$ in a clause of ω get then all the weight from the renamed literals x_i and $\neg x_i$, ensuring that this cost allocation yields a lower bound over the relaxation and also that the alias literals that get these weights are more tightly constrained.

6 The Search

The search algorithm looks for the best model of T by looking for the best model of the relaxation T^- that satisfies the constraints $x_i^k = x_i^j$ for all the aliases x_i^k and x_i^j of the same renamed variables x_i in T. This is achieved by a branch-and-bound DPLL-style search over T whose state s is a set s of literals over the renamed variables x_i that represents the commitments made so far. Initially s is empty. Then in each step, the best model M of $T^- \cup s^-$ and its cost $c^*(T^- \cup s^-)$ are computed, where s^- is the set of alias literals that correspond to s: namely, all x_i^j (resp. $\neg x_i^j$) are in s^- if x_i (resp. $\neg x_i$) in s. The lower bound $LB(s)$ is set to $c^*(T^- \cup s^-)$. The search does not continue beneath s if either $LB(s)$ is not smaller than the current upper bound (a bound conflict), the boolean constraint propagation procedures derives an *empty clause* (a logical conflict), or if M has no discrepancies (a solution). Else, a variable x_i with a discrepancy in M is selected (an alias pair x_i^j and x_i^k such that $x_i^j \neq x_i^k$ in M) the state s is extended with either x_i or $\neg x_i$, and the process is iterated. By setting a small target width w, the relaxation procedure that yields T^- from T, ensures that the compilation of T^- can be done in polynomial time and space, and the compilation in turn ensures that the best model M of $T^- \cup s^-$ and its costs can be computed efficiently for any set of literals s^-.

This basic search procedure is implemented on top of MiniSAT 2.0 [8], thus relying on the efficient boolean constraint propagation provided by the *two-literal watching* rule [28] which is performed right after every commitment. We also use its (dynamic) VSIDS variable selection heuristic [28] but branch only on the variables x_i that have been renamed, choosing always first the value x_i or $\neg x_i$ with least cost $c(x_i)$ or $c(\neg x_i)$. For relaxations T^- obtained by w-cutset renaming this means that in the worst case, the size of the search space will

be exponential in the size of the w-cutset used. Within this space, however, the lower bounds $LB(s)$ obtained from the compiled relaxation and the pruning that they produce, will normally keep the search away from this worst case scenario, as the experimental results below show.

7 Learning from Bound Conflicts

The implementation of the search on top of a SAT solver benefits from the ability of to learn new clauses during search after *logical failures*. It is well known that *clause learning* is a key technique in current solvers that can reduce the search space quite drastically [29, 30]. The success of *learning from logical conflicts* in SAT, suggested to look at the problem of *learning from bound conflicts* as when the lower bound $LB(s)$ does not improve the current upper bound UB for a given state s; i.e., when $LB(s) \geq UB$. In the context of solvers whose inference is based on *unit propagation*, this problem has been approached in [31] and [12] where a set of literals that explains the bound conflict is obtained and negated. Our solver, however, does not use only unit resolution but also and mainly optimal inference over the compiled relaxation T^-. The trivial way to learn in such setting is by simply recording the clause $\neg s$ when $LB(s) \geq UB$. This, however, while enables us to preserve the conflict-directed implementation of MiniSAT, has not pruning effect. A much more effective alternative is to find the smallest possible subset s' in s that explains the bound conflict, i.e., a subset $s' \subseteq s$ such that $LB(s') \geq UB$. Interestingly, it is possible to use the compiled d-DNNF representation of T^- for computing such 'causes' for failure even if there is no guarantee that such causes are minimal. The idea is simple and requires only a single downward pass over the DAG representing the compilation of T^-.

Basically, starting from the root node of the DAG representing the compilation of T^- we perform a top-down scan, skipping some nodes, while collecting the literals l in the leaves that are reached, retaining from this set only the literals l in s. Of course, if no node is skipped in this scan, we will get back the set s itself. Yet as we will see there are three types of nodes n that can be skipped because the commitments beneath them in the graph, if any, are not relevant either to the cost of their parent node in the graph or the bound failure. For this, let $c^*(n)$ be the cost of node in n in the DAG when evaluated in the context of the commitments s^- and let $c_0^*(n)$ be the value of the same node when evaluated in the context with no commitments at all (i.e., with s^- assumed empty, as in the beginning of the search).

Clearly if $c^*(n) = c_0^*(n)$, it means that the commitments beneath n are not relevant to its cost given s^0. Likewise, if n is an OR-node, the commitments beneath the child n_1 are not relevant to the cost of n either if $c_0^*(n_1) > c^*(n)$. Last, if n is an AND-node, the commitments beneath the child n_1 can be ignored if $c_0^*(n_1) + c^*(n_2) \geq UB$ as the commitments beneath the other child n_2 suffice to explain the bound failure.

By performing and exhaustive scan of the DAG representing T^- after finding that $LB(s) \geq UB$, starting from the root node while skipping nodes as above, it

is then possible to get a reduced set of alias literals $t^- \subseteq s^-$ such that $LB(t) \geq UB$ as well. The *conflict clause* $\neg t$ is then fed to the 1st Unique Implication Point [29] heuristic implemented by MiniSAT 2.0, that derives the *blocking clause* and the decision level to backtrack to.

8 Empirical Evaluation

The MinCostSAT solver SR(w) accepts a MinCostSAT problem in the form of a cost function c and a CNF theory T. Using w-cutset renaming, it then maps T into a relaxation T^- which is compiled in d-DNNF using Darwiche's c2d compiler, and sets the cost of the renamed literals according to the MIS procedure described above. It then carries a DPLL-style branch-and-bound search on top of MiniSAT 2.0 that benefits from the lower bounds obtained from the relaxation, as well as from unit propagation and clause learning from both logical and bound conflicts.

The experiments below have all been run on a grid consisting of 76 nodes, each one being a dual-processor Xeon "Woodcrest" dual core computer, with a clock speed of 2.33 GHz and 8 Gb of RAM. Execution time was limited to 1,800 seconds.

Overall performance

Table 1 compares SR(w) against three state-of-the-art Weighted-MAX SAT and MinCostSAT solvers over the 559 problems of the 2006 Weighted SAT Competition [11]: the winner of the competition, *toolbar* [17], the runner-up, *Lazy* [32], and the recent MinCostSAT solver, *MinCostChaff* [12] all ran on the same platform. The problems were converted into MinCostSAT by adding a *slack* variable x into each non-unary clause and setting $c(x)$ to the weight of the clause. Unary clauses over a literal x ($\neg x$) were removed and their weight was assigned to $\neg x$ (x) resp. Last, hard clauses, represented with a very large weight in Weighted MAX-SAT, were modeled as hard clauses (crisp constraints).

The competition problems fall into different categories: *MinCostSAT*, where all clauses are hard constraints and literals are weighted, *Weighted Max-SAT*, where all clauses are soft and weights are associated to constraints, and *Weighted Partial Max-SAT* which have a mix of both hard and soft constraints. This is explicitly indicated in Table 1.

The value of the target treewidth w used in the SR(w) solver is $w = 8$. The performance for other values of w is analyzed below. From the results shown, it is clear that SR(w) does well in relation to state-of-the-art MinCostSAT and Weighted-MAX solvers, trailing only Toolbar, while showing an slight edge over Lazy, which does a lot better in these instances than MinCostChaff. The best relative performance of SR(w) is on the Weighted Partial Max-SAT problems, where in two domains (WCSP; dense tight and sparse tight) manages to solve 10 and 17 instances where the other solvers solve none.

		SR(w)		toolbar-3.1		Lazy		MinCostChaff	
Set Name	N	S	T	S	T	S	T	S	T
Auctions (paths)	30	20	136.28	28	244.49	21	123.69	0	0
Auctions (sched.)	30	18	131.97	30	82.93	28	0.65	6	317
Auctions (regions)	30	30	171.07	30	3.39	30	41.39	13	173
Max-Clique (brock)	12	0	–	4	59.14	4	66.57	0	0
Max-Clique (c-fat)	7	3	23.92	7	10.99	7	0.07	1	973
Max-Clique (ham.)	6	2	23.92	5	67.03	5	119.02	3	627
Max-Clique (John.)	4	2	1.40	3	34.96	3	25.98	2	782
Max-Clique (Kell.)	2	1	16.63	1	20.67	1	27.50	0	0
Max-Clique (Mann)	4	1	0.06	2	48.63	1	0.18	1	1,472
Max-Clique (p_hat)	12	1	1,813.73	7	385.63	7	367.21	3	1,591
Max-Clique (san)	11	0	–	4	649.00	1	6.03	0	0
Max-Clique (sanr)	4	0	–	3	463.08	2	466.25	0	0
Max-One	45	40	130.65	45	129.93	30	357.29	1	1,623
WCSP (SPOT5)	21	9	16.24	5	81.03	3	533.96	2	277
QCP	25	6	242.99	11	133.25	7	255.79	25	43
MinCostSAT	243	133	54.7%	185	76.1%	150	61.7%	57	23.5%
WCSP (S-L)	20	20	40.22	18	344.46	10	351.52	17	252
WCSP (D-L)	20	10	722.98	16	446.24	13	629.06	3	339
WCSP (D-T)	30	10	546.82	0	0.00	0	0.00	0	0
WCSP (S-T)	20	17	413.89	0	0.00	0	0.00	0	0
WCSP (SPOT5)	21	9	16.24	4	83.02	3	655.94	8	1,189
Wt. Partial Max-SAT	111	66	59.5%	38	34.2%	26	23.4%	28	25.2%
WCSP (S-L)	20	13	6.93	20	3.33	18	299.49	15	285
WCSP (D-L)	20	11	656.23	20	17.51	20	260.34	1	970
WCSP (D-T)	30	27	270.02	30	33.44	0	0.00	0	0
WCSP (S-T)	20	13	60.22	20	161.54	0	0.00	0	0
Wt. Max-Cut (spin.)	5	2	0.43	3	49.71	2	0.14	2	820
Wt. Max-Cut (brock)	12	2	0.10	12	9.20	12	11.62	0	0
Wt. Max-Cut (c-fat)	7	4	15.44	7	7.35	7	15.50	1	881
Wt. Max-Cut (hamm.)	6	0	–	4	67.97	5	285.10	3	638
Wt. Max-Cut (john.)	4	1	12.10	3	54.79	3	47.45	2	731
Wt. Max-Cut (keller)	2	0	–	2	10.33	2	11.20	0	0
Wt. Max-Cut (mann)	4	2	78.20	4	1,034.70	4	640.47	1	1,492
Wt. Max-Cut (p-hat)	12	6	193.53	12	1,138.46	12	7.03	2	1,415
Wt. Max-Cut (san)	11	1	541.14	11	54.73	11	36.53	0	0
Wt. Max-Cut (sanr)	4	1	1,575.30	4	39.59	4	16.86	0	0
Wt. Ramsey	48	33	124.83	35	3.27	29	42.72	34	17
Wt. Max-SAT	205	116	56.6%	187	91.2%	129	62.9%	61	29.8%
Total	559	315	56.4%	410	73.3%	305	54.6%	146	26.1%

Table 1. Overall performance over the 2006 Weighted SAT Competition instances. S is the number of solved instances in each domain and T the average time needed to solve them in seconds. Domains are split into native *MinCostSAT*, *Weighted Partial Max-SAT*, and *Weighted Max-SAT*. Times for SR(w) include the relaxation, compilation and search, for the target treewidth fixed at $w = 8$ which results in the best coverage over these instances.

Domains	I	w=1	w=2	w=4	w=8	w=16	w=32	w=64	w=tw
Auction (paths)	30	5	10	12	20	25	24	7	29
Auction (sched.)	30	18	17	18	18	20	22	17	26
Auction (regions)	30	24	25	28	30	30	30	29	30
Max-Clique	62	10	10	11	10	10	6	4	11
Max-One	45	42	42	42	40	17	0	0	9
WCSP (SPOT5)	42	8	8	8	18	8	7	0	12
QCP	25	10	10	9	6	0	0	0	0
WCSP (S-L)	40	12	22	32	33	31	0	0	0
WCSP (D-L)	40	1	1	3	21	0	0	0	0
WCSP (D-T)	60	0	0	2	37	28	0	0	9
WCSP (S-T)	40	1	0	0	30	29	0	0	2
Wt. Max-Cut (spin.)	5	1	2	2	2	1	0	0	1
Wt. Max-Cut	62	2	7	7	17	10	0	0	7
Wt. Ramsey	48	25	25	26	33	15	0	0	10
Total Solved	559	159	179	200	315	224	89	57	146
% solved		28.4%	32.0%	35.8%	56.4%	40.1%	15.9%	10.2%	26.1%

Table 2. Impact of target treewidth parameter w in SR(w): Number of problems solved for each value of w. The last column with $w = tw$, means no relaxation at all so that $T^- = T$. In such a case, the problem is solved without search from the d-DNNF compilation of T, when the compilation is successful.

Impact of Target Width, Learning, and Renaming Schemes

Table 2 shows the number of problems solved by SR(w) for different bounds $w = 1, 2, 4, 8, 16, \ldots$ on the (simplified) treewidth of the relaxation. Interestingly, the best coverage is not obtained for the extreme values of w but for $w = 8$ and $w = 16$. The last column shows the results corresponding to the relaxation $T^- = T$. In such a case, the theories T, if they compile, are solved without search (no variables are renamed), so that column is testing the CNF to d-DNNF compiler, which does pretty well, solves by itself the same number of problems as MinCostChaff. Further details on some of the instances for various values of w are shown in Table 3 and Table 4, illustrating the typical trade-off between search and inference: for relaxations with larger w, the search visits less number of nodes, but the compilation is more expensive in terms of space and time, and the overhead per node in the search (which is linear in the size of the compilation) is greater. Indeed most of the failures for widths $w \leq 32$ are search timeouts.

Table 5 shows the combined impact of the treewidth bound w, the renaming scheme (w-cutset vs. w-mini-buckets), and learning (turned on and off) on two instances. As it can be seen, learning can help a lot, in particular, for small values of w, while the differences between the results obtained with w-cutset and w-mini-bucket renaming are not so clear cut.

Problem	# vars.	# Clauses	tw	w	# Nodes	Relax	Comp	Search	Total
8.wcsp.dir	20	21	6	4	2	0.001	0.010	0.000	0.011
1502.wcsp.dir	515	427	7	4	20	0.045	0.040	0.003	0.088
503.wcsp.dir	317	849	13	12	7	0.036	0.120	0.002	0.158
404.wcsp.dir	187	966	23	16	162	0.022	0.070	0.019	0.111
54.wcsp.dir	154	441	23	8	428	0.011	0.040	0.036	0.087
29.wcsp.dir	139	629	31	16	98	0.016	0.080	0.026	0.122
1504.wcsp.dir	1,577	6,312	43	32	339	0.853	16.380	38.971	56.204
505.wcsp.dir	552	3,296	46	45	3	0.182	19.230	0.132	19.544
408.wcsp.dir	392	3,013	55	32	1372	0.113	0.670	10.458	11.241
42.wcsp.dir	361	1,883	61	32	10,654	0.084	4.660	733.013	737.757

Table 3. A closer look at some instances from the SPOT5 domain: tw is the instance upper bound treewidth as estimated by the MIW ordering, w is the treewidth bound that produced the best total time for SR(w) over each instance, and *Relax*, *Comp*, and *Search* are the relaxation, compilation and search times all in seconds.

w	d-DNNF size	# Nodes	Relax.	Comp.	Search	Total
1	5,581	453,758	0.03	0.04	131.14	131.21
2	5,130	76,231	0.02	0.03	7.65	7.7
4	4,922	4,737	0.02	0.03	0.41	0.46
8	11,240	482	0.02	0.05	0.07	0.13
16	25,721	98	0.02	0.08	0.03	0.12
30	35,822	3	0.01	0.32	0	0.34

Table 4. A closer look at the performance for various relaxation treewidths w over the *29.wcsp.dir* instance from Table 3. *d-DNNF size* denotes the number of nodes in the d-DNNF DAG while *# nodes* refers to the number of nodes during search.

	w-mini-buckets renaming		w-cutset renaming	
w	Learning	No learning	Learning	No learning
1	21,373 (21.99)	98,461 (102.6)	14,399 (15.88)	Timeout
2	25,162 (28.3)	129,859 (143.63)	9,302 (10.1)	556,181 (546.47)
4	64,716 (76.61)	146,377 (168.29)	11,218 (18.71)	75,511 (105.81)
8	11,627 (60.63)	18,473 (96.17)	4,889 (40.08)	9,669 (67.25)
16	474 (196.96)	695 (300.37)	238 (186.59)	737 (533.52)

Table 5. Number of nodes in the search and total run-time (in parenthesis) over a random Max-3-SAT instance. w denotes the treewidth bound on the relaxation. MIW estimate was 29.

9 Discussion

We have presented an structural relaxation scheme based on renaming variables that maps a CNF theory T into a relaxation T^- with a (simplified) treewidth bounded by a parameter w. We have then used this relaxation for developing a MinCostSAT solver that obtains its lower bounds from a suitable compilation of the relaxation into d-DNNF. The observed performance of this solver, implemented on top of a state-of-the-art SAT solver that benefits also from efficient unit propagation and clause learning, suggests that the idea of compiling structural relaxations for guiding the search, is an idea worth exploring in further depth that may be applicable in other contexts too. This approach is closely related to other methods that aim to combine structural and search-methods and in particular to Dechter's mini-buckets. Indeed, if the relaxation is done by the scheme that mimics mini-buckets which we call w-mini-bucket renaming, we would be obtaining the same sort of structural lower bounds. On the other hand, by appealing to *explicit relaxations*, our approach is more general as it is not tied to a particular renaming scheme, can benefit from existing tools such as SAT solvers and d-DNNF compilers, and deals in a natural way with dynamic variable ordering. Just in time for the last version of this paper, we have learned about an independent but closely related relaxation scheme in the context of Bayesian Networks to appear at UAI-07 [33].

Acknowledgements

We thank the anonymous reviewers for useful comments, R. Dechter for an informative exchange on treewidth, and N. Sörenssón, D. Le Berre, and Z. Fu for answering various questions about MiniSAT and MinCostChaff. We also thank the authors of the tools, the c2d compiler and MiniSAT, used in our solver, and R. Isla from UPF for help with the cluster. H. Geffner is partially supported by grant TIN2006-15387-C03-03 from MEC, Spain.

References

1. Culberson, J.C., Schaeffer, J.: Pattern Databases. Comp. Int. **14** (1998)
2. Korf, R.E.: Finding Optimal Solutions to Rubik's Cube Using Pattern Patabases. In: Proc. AAAI-98. (1998)
3. Bonet, B., Geffner, H.: Planning as Heuristic Search. Art. Int. **129** (2001)
4. Mackworth, A.K.: Consistency in Networks of Relations. Art. Int. (8) (1977)
5. Li, X.Y.: Optimization Algorithms for the Minimum–Cost Satisfiability Problem. PhD thesis, North Carolina State University (2004)
6. Dechter, R.: Constraint Processing. Morgan Kauffman (2003)
7. Darwiche, A.: New Advances in Compiling CNF to Decomposable Negational Normal Form. In: Proc. ECAI'04. (2004)
8. Eén, N., Söressón, N.: MiniSAT: an Extensible SAT Solver. Technical report, Chalmers University (2005)

9. Darwiche, A.: Decomposable Negation Normal Form. Journal of the ACM **48**(4) (2001) 608–647
10. Marquis, P., Darwiche, A.: Compiling Propositional Weighted Bases. Artificial Intelligence **157**(1–2) (2004) 81–113
11. J. Argelich et al.: First Evaluation of Max-SAT solvers. Available at http://www.iiia.csic.es/ maxsat06 (2006)
12. Fu, Z., Malik, S.: Solving the Minimum Cost Satisfiability Problem using SAT Based Branch-and-Bound Search. In: Proc. of ICCAD'06. (2006)
13. Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaille, G., Fargier, H.: Semiring-Based CSPs and Valued CSPs: Frameworks, Properties and Comparison. Constraints **4**(3) (2004) 199–240
14. Papadimitrou, C.H.: Computational Complexity. Addison-Wesley (1994)
15. Giunchiglia, E., Maratea, M.: Solving Optimization Problems with DLL. In: Proc. of ECAI'06. (2006)
16. Larrosa, J., Heras, F.: Resolution in Max-SAT and its Relation to Local Consistency in Weighted CSPs. In: Proc. of IJCAI-05. (2005) 193–199
17. Larrosa, J., Heras, F.: New Inference Rules for Efficient Max-SAT Solving. In: Proc. AAAI'06. (2006)
18. Li, C.M., Manyá, F., Planes, J.: Detecting Disjoint Inconsistent Subformulas for Computing Lower Bounds for Max-SAT. In: Proc. AAAI'06. (2006)
19. de Givry, S., Larrosa, J., Messeguer, P., Schiex, T.: Solving Max-SAT as Weighted CSP. In: Proc. CP-03. (2003) 363–376
20. Dechter, R.: Enhancement Schemes for Constraint Processing: Backjumping, Learning and Cutset Decomposition. Artificial Intelligence **43**(3) (1990) 273–312
21. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. J. Comput. Syst. Sci. **64**(3) (2002) 579–627
22. Bidyuk, B., Dechter, R.: On Finding Minimal w-cutset. In: Proc. UAI-04. (2004)
23. Dechter, R.: Mini–Buckets: a General Scheme for Generating Approximations in Automated Reasoning. In: Proc. of IJCAI-97. (1997) 1297–1303
24. Dechter, R.: Bucket Elimination: a Unifying Framework for Reasoning. Artificial Intelligence **113**(1–2) (1999) 41–85
25. Darwiche, A., Marquis, P.: A Knowledge Compilation Map. Journal of AI Research (17) (2002) 229–264
26. Darwiche, A.: Recursive Conditioning. Artificial Intelligence **126**(1–2) (2001) 5–41
27. Coudert, O., Madre, J.C.: New Ideas for Solving Covering Problems. In: Proc. of DAC'95. (1995) 641–646
28. Moskewicz, M.W., Madigan, C.F., et al: Chaff: Engineeering an Efficient SAT Solver. In: Proc.of DAC-01. (2001)
29. Zhang, L., et al., C.F.M.: Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. In: Proc. of ICCAD'01. (2001)
30. Silva, J.P.M., Sakallah, K.A.: GRASP – A New Search Algorithm for Satisfiability. In: Proc. of ICCAD'96. (1996) 220–227
31. Manquinho, V., Marques-Silva, J.: Search Pruning Techniques in SAT-based Branch-and-Bound Algorithms for the Binate Covering Problem. IEEE Trans. on CAD and Integrated Circuit and Systems **21** (2002) 505–516
32. Alsinet, T., Manyá, F., Planes, J.: Improved Exact Solvers for Weighted Max-SAT. In: Proc. of the 8th SAT conference. (2005)
33. Choi, A., Chavira, M., Darwiche, A.: Node splitting: A scheme for generating upper bounds in bayesian networks. In: Proceedings UAI-07. (2007) To appear.