

A General Nogood-Learning Framework for Pseudo-Boolean Multi-Valued SAT

Paper ID: 140

Abstract

We formulate a general framework for pseudo-Boolean multi-valued nogood-learning, generalizing conflict analysis performed by modern SAT solvers and its recent extension for disjunctions of multi-valued variables. This framework can handle more general constraints as well as different domain representations, such as interval domains which are commonly used for bounds consistency in constraint programming (CP), and even set variables. Our empirical evaluation shows that our solver, built upon this framework, works robustly across a number of challenging domains.

Introduction

In (Jain, O’Mahony, and Sellmann 2010) a new no-good learning approach for multi-valued satisfaction (SAT) problems was presented which was shown to infer significantly stronger no-goods than those inferred by a mechanism that is based on a Boolean representation of a multi-valued problem. Like earlier methods, the learning approach is based on an implication graph where nodes represent domain events and edges are implications drawn by the clauses in the given problem. One of the novelties in (Jain, O’Mahony, and Sellmann 2010) was the sole focus on variable *inequations* to infer the minimal reasons for a failure.

In this paper, we investigate why the particular use of inequations results in stronger no-goods and we formulate a general framework for multi-valued nogood-learning that can handle more general constraints, and also different domain representations, such as interval domains, which are commonly used for bounds consistency in constraint programming (CP). This is an essential step towards an integration of pseudo-Boolean and multi-valued SAT.

SAT vs. CP

Although both are concerned with feasibility problems, state-of-the-art SAT and CP methods differ significantly in style and philosophy. In SAT, we use one fixed, simple input format, allow only one type of constraints, namely clauses, and variables that can take only two different values. To model a real-world problem, any structure that it may exhibit must be crushed into a number of loosely connected atomic parts: Boolean variables and clauses.

In CP, on the other hand, the input format is not fixed a priori. Practically all CP solvers support variables that can take more than just two values, and many support more

involved constraints, such as linear inequalities on two or more numeric variables, element constraints or even global constraints such as all-different, shortest path, knapsack, or grammar constraints.

Arguably, CP solvers ought to have a significant advantage over SAT solvers, given the additional benefit of being provided with structural information of the problem to be solved. Interestingly, CP solvers that are based on SAT reformulations are highly competitive nevertheless, such as *Sugar* (Tamura, Tanjo, and Banbara 2008), which won the CP global constraints competition in two consecutive years (CSP Competition 2008/2009). With important exceptions, it appears that approaches that exploit structure “just in case” tend to be more expensive while approaches that improve inference “just in time” are often more efficient.

SAT-X Hybrids

Consequently, hybrid approaches have been introduced, such as pseudo-Boolean solvers (e.g., (Dixon, Ginsberg, and Parkes 2004)), SAT modulo theories (Nieuwenhuis, Oliveras, and Tinelli 2006), and the award-winning lazy clause generation approach (Ohrimenko, Stuckey, and Codish 2007). In the adjacent field of integer programming, the SCIP solver (Achterberg 2004) computes nogoods based on the analysis of the linear program at the “conflict” and the analysis of a Boolean-SAT like inference graph. Another related contribution is the recently proposed multi-valued SAT solver CMVSAT-1 (Jain, O’Mahony, and Sellmann 2010) that strengthens nogood learning by directly incorporating the knowledge that each variable must take exactly one out of a number of values.

In this paper, we analyze the nogood learning component of CMVSAT-1 and generalize the approach by providing a framework for strong multi-valued nogood learning. We identify sufficient conditions for constraint propagators to support nogood learning effectively. Moreover, we show that the generalized framework is capable of handling various domain representations, such as interval domains, and associated propagators that enforce bounds consistency.

CMVSAT-1

We begin by reviewing briefly the nogood learning component of CMVSAT-1, in the context of multi-valued SAT.

Definition 1 (Multi-Valued Variables). A *multi-valued variable* X_i is a variable that takes values in a finite set D_i called

the domain of X_i .

Definition 2 (Multi-Valued Clauses). • Given a multi-valued variable X_i and a value v , we call the constraint $X_i = v$ a *variable equation* on X_i . A variable equation $X_i = v$ is called *satisfiable* iff $v \in D_i$.

- Given a set of multi-valued variables $X = \{X_1, \dots, X_n\}$ and a set $T \subseteq X$, a *clause* over T is a disjunction of variable equations on variables in X .

Definition 3 ((Partial) Assignments and Feasibility). Given a set of multi-valued variables $X = \{X_1, \dots, X_n\}$, denote with D the union of all domains D_1, \dots, D_n . Furthermore, denote with S, T subsets of X .

- A function $\alpha : S \rightarrow D$ is called an *assignment* of variables in S . α is called *partial* iff $|S| < n$, and *complete* otherwise. If $|S| = 1$, α is called a *variable assignment*.
- An assignment α is called *admissible* iff $\alpha(X_i) \in D_i$ for all $X_i \in S$.
- An assignment α of variables in S is called *feasible* with respect to a clause over T iff $T \setminus S \neq \emptyset$ or if there exists a variable equation $X_i = \alpha(X_i)$ in the clause.
- Given a clause c , a complete, admissible, and feasible assignment is called a *solution* for c .

Definition 4 (Multi-Valued SAT Problem). Given a set X of multi-valued variables and a set of clauses over subsets of X , the *multi-valued SAT problem (MV-SAT)* is to decide whether there exists an assignment that is a solution for all clauses.

Multi-Valued Nogood Learning

Boolean SAT nogood learning is commonly based on a so-called implication graph. Analogously, the conflict analysis presented in (Jain, O’Mahony, and Sellmann 2010) is also based on a graph. Curiously, this graph was set up to only represent the implications that result in variable *inequations*. That is to say, even when propagation could show that some variable *must* take a certain value in its domain, the information added to the graph is that it cannot take any of the remaining values.

Note that this is not merely a matter of choice like representing a binary CP constraint as positive list of allowed tuples or as negative list of disallowed tuples. The reason why the approach in (Jain, O’Mahony, and Sellmann 2010) is based on variable inequations *only* is that this is the *most atomic information* that is communicated between constraints. This way, when a reason for a conflict is sought, it can be traced back to the *minimal domain reductions* that are needed to further reduce the domains further. To make the discussion less abstract consider the following example.

Example 1. Consider a constraint satisfaction problem with five variables and five constraints. The variables have domains $D_1 = D_2 = D_4 = \{1, 2, 3\}$, $D_3 = \{1, \dots, 7\}$, $D_5 = \{1, 2\}$. The constraints are $X_1 \neq X_4$, $X_2 \neq X_3$, $X_3 + 9 \geq 5X_4$, $X_3 + 4 \geq 5X_5$, and $X_4 \neq X_5 + 1$. □

We can model this problem as multi-valued SAT formula as follows: We need to add multiple clauses for each CSP constraint. Among others, for constraints $X_1 \neq X_4$, $X_2 \neq X_3$, and $X_4 \neq X_5 + 1$ we add $(X_1 = 2 \parallel X_1 = 3 \parallel X_4 = 2 \parallel X_4 = 3)$, $(X_2 = 1 \parallel X_2 = 3 \parallel X_3 = 1 \parallel X_3 =$

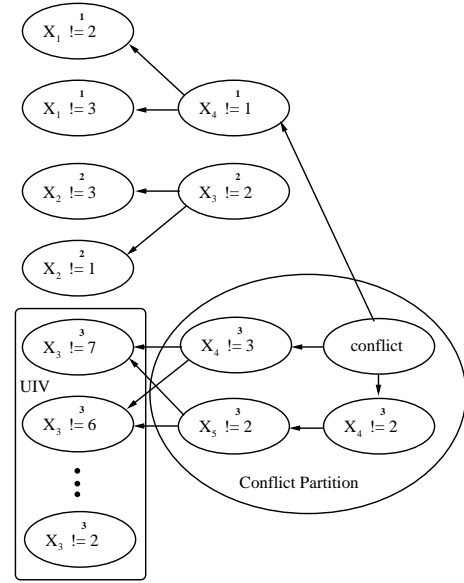


Figure 1: MV-SAT Implication Graph.

$3 \dots 7)$, and $(X_4 = 1 \parallel X_4 = 3 \parallel X_5 = 2)$. Constraints $X_3 + 9 \geq 5X_4$ and $X_3 + 4 \geq 5X_5$ are enforced by the clauses $(X_4 = 1 \parallel X_4 = 2 \parallel X_3 = 6, 7)$ and $(X_5 = 1 \parallel X_3 = 6, 7)$. In Figure 1 we show the implication graph that emerges when we first branch by committing $X_1 \leftarrow 1$, then $X_2 \leftarrow 2$, and finally $X_3 \leftarrow 1$, which leads to a conflict.

An important novelty introduced in (Jain, O’Mahony, and Sellmann 2010) is the notion of a unit implication variable:

Definition 5 (Unit Implication Variable). Given a multi-valued SAT problem and an implication graph G , assume that, if all nodes associated with variable X are removed from G , there exists no path from the conflict node to any branch node on the lowest branch level. Then, we call X a *unit implication variable (UIV)*.

In contrast to Boolean SAT, in multi-valued SAT a clause is unit as soon as all remaining variables equations regard the same variable. Consequently, a nogood computed by analyzing the implication graph is no longer defined by a *cutpoint* in the graph, but by a *cutset* of nodes that all regard the same variable. As long as we compute nogoods based on such cutsets, we can be sure that the corresponding nogood is unit after backjumping.

In our example, X_3 is a UIV. Based on its associated cutset, we can compute a conflict partition and set the nogood as the negation of the conjunction of all variable inequations which have a direct parent in the conflict partition. In our example, we thus find the multi-valued clause $(X_4 = 1 \parallel X_3 = 6 \parallel X_3 = 7)$. After backjumping to the highest depth level after the conflict level in our learned clause (in our case level 1), this clause is unit and prunes the domain of X_3 to $D_3 = \{6, 7\}$. That is, equipped with this nogood the solver does not need to branch on $X_3 \leftarrow 2$, $X_3 \leftarrow 3$, $X_3 \leftarrow 4$, and $X_3 \leftarrow 5$ as all of these settings would fail for the very same reason as $X_3 \leftarrow 1$ did. Note that the same strength of inference could not have been achieved if we had just noted $(X_3 = 1)$ as a sufficient reason for $(X_4 \neq 3)$ and $(X_5 \neq 2)$.

A Generalized Framework

We now generalize this approach and identify sufficient conditions under which strong nogoods can still be learned effectively. In multi-valued SAT, the only constraints allowed are multi-valued clauses. Moreover, nogood learning requires that branching decisions regard just one variable, and propagation is limited to unit propagation only. By generalizing the nogood learning component from (Jain, O’Mahony, and Sellmann 2010) we liberate ourselves from the above mentioned restrictions and are able to handle more general constraints as well as different variable domain representations. This is an essential step for the design of solvers that can handle pseudo-Boolean multi-valued SAT problems.

A Formal View of Constraint Programming

A canonical view in CP is to view the set of constraints as split in two classes. The first class of constraints are *primitive constraints*. Originally primitive constraints were introduced as constraints that can be directly expressed by the respective domain representation. However, from a formal point of view it is actually often convenient to use primitive constraints in the opposite way, namely to express what domain representation is used.

For example, let us consider all unary constraints as primitive. Obviously, any unary constraint can be expressed as a truth table. Consequently, the constraint can be directly encoded in the domain of the variable if the latter is represented as a list of allowed values. Alternatively, we can represent the current domain of a variable by such a primitive constraint. To give another example, assume that we consider unary lower or equal and greater or equal constraints as primitive (whereby implicitly we assume that an ordering on the values is given). This set of primitive constraints is equivalent to a domain representation that maintains an interval of values for each variable only, as is usually the case for approaches that maintain bounds consistency.

The other class of constraints is usually referred to as *secondary constraints* and it consists of all constraints that are supported which are not primitive. Now, the *propagation* of a secondary constraint C_s can be viewed as the process of entailment of new primitive constraints C_p from the conjunction of the existing primitive constraints and the secondary constraint that is propagated.

$$\bigwedge_{C_i \in \text{PrimStore}} C_i \wedge C_s \vdash C_p \quad (1)$$

Note that these inferred constraints need to be removed from the constraint store upon backtracking.

Nogoods as Disjunctions of Negated Primitives

Let us view the nogood learning approach from (Jain, O’Mahony, and Sellmann 2010) based on the above formal view of constraint programming. All *unary inequations* are primitive, and multi-valued clauses, i.e., *disjunctions of variable equalities* are the only secondary constraints.

Nodes in the implication graph represent primitive constraints, the outgoing arcs link a node to a set of primitive constraints that, when conjoined with some secondary constraint, allow the entailment of the corresponding primitive

constraint. A conflict is reached when a subset of nodes in the graph represents a set of primitive constraints that contradict one another. Since the only primitive constraints are variable inequations, conflict sets are necessarily sets of unary constraints regarding the same variable that together exclude all potential values.

When such a set of conflicting nodes is found, any cutset whose removal from the graph disconnects the conflict nodes and the nodes directly introduced through branching defines a valid redundant constraint. Namely, for the primitive constraints in the cutset it has been found that they contradict each other in the given constraint system as they entail the conflict. Consequently, we conclude that the conjunction of these primitive constraints must be false or, equivalently, that the *disjunction of the negation of these primitive constraints must be true*. Therefore, in this framework nogoods take the form of disjunctions of negated primitive constraints.

Valid Cutsets in Implication Graph

While any cutset results in a constraint that is redundant to the given system of constraints, in order to make progress we search for specific cutsets only. Namely, we are interested in those redundant constraints *that allow us to entail new primitive constraints efficiently upon backtracking*. For traditional SAT and also in multi-valued SAT this means that we try to learn a clause that will be *unit* upon backjumping so that unit propagation can be effective.

To generalize this notion, we require that the constraint reasoning system provides an efficient approach for finding cutsets, whereby the latter consist in negated primitive constraints whose disjunction can be expressed efficiently as a conjunction of primitive constraints. We call such cutsets *valid*. In particular, we require that primitive constraints added upon branching are valid cutsets.

Efficient Nogood Learning: Sufficient Conditions

To summarize, assume that the following conditions are met by our constraint reasoning system:

1. The system distinguishes between primitive and secondary constraints. Propagation of secondary constraints is the entailment of new primitive constraints.
2. Apart from adding new primitive constraints, secondary constraint propagators can efficiently provide a set of primitive constraints that is sufficient for entailing the new constraints.
3. Conflicting sets of primitive constraints can be detected efficiently. Disjunctions of negations of primitive constraints are supported as secondary constraints.
4. Negated primitive constraints can be expressed as a set of primitive constraints. Certain disjunctions of negated primitive constraints (valid cutsets) can efficiently be computed and represented as a set of primitive constraints.
5. Branching is executed as the addition of one or more primitive constraints. If a conjunction of primitive constraints is added, then the negation of of this conjunction is a valid cutset in the sense of (4).

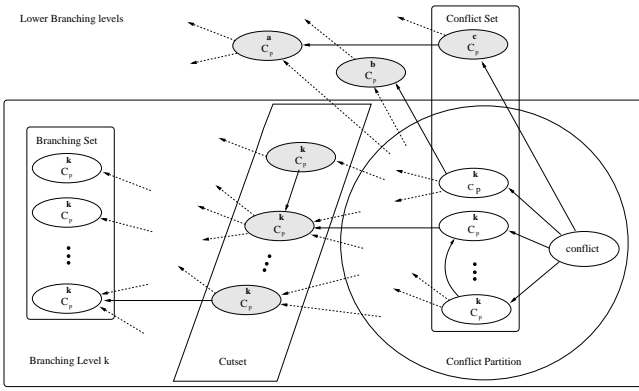


Figure 2: Abstract Implication Graph. We omit those parts of the graph that cannot be reached from the conflict node.

If these conditions are satisfied, we can efficiently learn strong nogoods in the following way (see Figure 2): We maintain an implication graph where nodes represent primitive constraints added by branching (5) or entailed by propagators (1). Nodes are associated with the branching level on which they were added or entailed. We add arcs from entailed primitive constraints to those primitive constraints that are needed for the entailment (2). When a conflict is reached (3), we compute a valid cutset on the last branching level that separates the conflict nodes (3) from the branching nodes (5). A nogood is inferred as the negation of the conjunction of these cutset nodes and nodes on lower branching levels that are adjacent to nodes in the conflict partition induced by the cutset. Note that the validity of the cutsets allows us, upon backjumping, to propagate the remaining disjunction of negated primitive constraints by expressing it as a set of primitive constraints.

Framework Application Example

The abstract procedure above now allows us to generalize nogood learning. The framework is general enough to handle general constraints other than clauses (provided that condition (2) is met) as well as multi-valued variables. In fact, the system can even handle *set variables* whose domains (subsets of the power set over some universe of elements) can be represented as ranges, for example by using length-lexicographic bounds (Gervet and Van Hentenryck 2006).

Consider for example a constraint reasoning system where primitive constraints are unary lower or equal and greater or equal constraints on some variables (we call these *range variables*) and inequations for the other variables (we refer to these variables as *domain variables*). According to condition (3) we need to support disjunctions of lower or equal and greater or equal constraints on range variables and equalities on domain variables in our system. Let us assume that, on top of these secondary constraints, the system also supports linear inequalities. For both types of constraints it is trivial to satisfy condition (2). If we branch by splitting domains for range variables and by adding inequations for domain variables (5), all conditions are satisfied and we can use our nogood learning approach.

To illustrate how nogood learning in this exemplary system works, consider the following constraint satisfac-

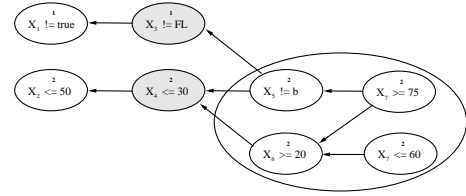


Figure 3: Implication Graph for Inequalities as Primitives.

tion problem. Assume we have seven variables $X_1 \in \{\text{true}, \text{false}\}$, $X_3 \in \{\text{NY}, \text{TX}, \text{FL}, \text{CA}\}$, $X_5 \in \{r, g, b\}$, and $X_2, X_4, X_6, X_7 \in \{1, \dots, 100\}$ and six constraints: $(X_1 = \text{true} \parallel X_3 = \text{NY} \parallel X_3 = \text{CA})$, $(X_2 \geq 60 \parallel X_4 \leq 30)$, $(X_3 = \text{FL} \parallel X_4 \geq 60 \parallel X_5 = r \parallel X_5 = g)$, $X_4 + X_6 \geq 50$, $(X_5 = b \parallel X_6 \leq 10 \parallel X_7 \geq 75)$, and $X_6 + X_7 \leq 80$.

Assume further that we first branch by adding the primitive constraint $X_1 \neq \text{true}$ (see Figure 3). From here we entail by the first secondary constraint that $X_3 \neq \text{FL}$ (and $X_3 \neq \text{TX}$ but we will not use this information). Say we branch on X_2 next and add $X_2 \leq 50$. By the second constraint we entail $X_4 \leq 30$ and, by the third and fourth constraint, we entail $X_5 \neq b$ and $X_6 \geq 20$. The fifth and sixth constraint then entail $X_7 \geq 75$ and $X_7 \leq 60$, two primitive constraints which obviously contradict each other. Analyzing the implication graph, we find that $X_4 \leq 30$ is a non-dominated cutset (in the sense of (Jain, O’Mahony, and Sellmann 2010)) on the lowest branching level, and we learn the nogood $(X_3 = \text{FL} \parallel X_4 \geq 31)$. Important for us is that this cutset is valid in the sense of condition (4) (the negation of $X_4 \leq 30$ can obviously be expressed as a primitive constraint).

We would like to point out two issues: First, note that, for this particular system, not any set of primitive constraints that regards the same variable results in a valid cutset. For example, assume that constraints $(X \leq 10)$ and $(X \geq 5)$ defined a cutset in some inference graph. Their negated conjunction $(X \leq 4 \parallel X \geq 11)$ cannot be expressed as conjunction or primitive constraints and it is therefore not valid. Secondly, in general, a valid cutset need not consist of primitive constraints that regard the same variable only. For example, it is conceivable that a constraint reasoning system treats inequalities over two variables $(X \leq Y)$ as primitive constraints. A disjunction of negated primitive constraints is valid (aka “unit” in SAT) iff it can be expressed as conjunction of primitive constraints. Consequently, the fact that a cutset consists in primitive constraints over the same variable is, in general, by itself neither necessary nor sufficient for validity.

Post-Processing the Implication Graph

As in conventional Boolean SAT, we can strengthen the learned nogoods by computing a minimal (not necessarily minimum) conjunction of negated literals in our nogood (note that these negated literals are primitive constraints!) for which propagation still results in a conflict (Een and Sörensson 2005). However, we can do even more when we assume that secondary constraints provide the corresponding functionality.

Consider the previous example. The final conflict is given

by the conjunction of $X_7 \geq 75$ and $X_7 \leq 60$. Sufficient for a conflict would have been if we had just inferred $X_7 \leq 74$, so let us replace the node $X_7 \leq 60$ with $X_7 \leq 74$. When the node $X_7 \leq 60$ was added to the implication graph, this happened as a result of propagating the sixth constraint $X_6 + X_7 \leq 80$. To achieve a node $X_7 \leq 74$ it would therefore have sufficed to have the primitive constraint $X_6 \geq 6$. Suppose the linear constraint can provide this minimal primitive constraint needed for inferring that $X_7 \leq 74$. Then, we need to check what other implications the node $X_6 \geq 20$ was needed for. We observe that, to infer $X_7 \geq 75$ by the fifth constraint, we needed at least that $X_6 \geq 11$. When replacing the node $X_6 \geq 20$ we must add the strongest precondition for all constraints that have added incoming arcs to this node. We therefore replace the node with $X_6 \geq 11$.

Working our way through the graph this way, we find next that $X_4 \leq 30$ can be replaced by the stronger of the nodes $X_4 \leq 59$ and $X_4 \leq 39$. As we have now reached a node that is part of the nogood, we stop further minimizations that would result from the weakening of $X_4 \leq 30$ to $X_4 \leq 39$. We therefore learn the *strengthened nogood* ($X_3 = \text{FL} \parallel X_4 \geq 40$).

Note that this post-processing of the implication graph has the potential to remove certain nodes entirely, for example when the initial domain bound of a range variable is enough to infer a bound on another that is sufficient to entail the conflict. Therefore, post-processing can also create new cutsets that dominate the ones in the non-processed implication graph. Consequently, post-processing should always take place *before* the cutset is computed and not (as we just did) after the nogood is learned.

Empirical Evaluation

We devised a constraint system, *CMVSAT-2*, based on the results in the previous section. In particular, the system provides for range and domain variables, whereby primitive constraints for range variables consists in lower or equal and greater or equal constraints. For domain variables, primitive constraints are inequalities. Secondary constraints that we provide so far are linear inequalities and disjunctions of equalities and range constraints, such as ($X_1 \in [5 \dots 60] \parallel X_2 \in [37 \dots 74] \parallel X_5 = b \parallel X_{10} \leq 15$). Note that the latter constraints include disjunctions of negated primitive constraints.

In order to test the effectiveness of our general framework for conflict driven nogood learning, we consider problems from three domains, each of which combines challenging combinatorial constraints with linear inequality constraints: ‘quasigroup with holes’ problem with costs (qwh-c), market split problem (msp), and a weighted version of N-queens problem (nqueens-w). These problem domains are discussed below. We compare our solver *CMVSAT-2* with the pure SAT solver *MiniSat 2.2.0* (Eén and Sörensson 2004), the CSP solver *Mistral* (Mis 2006), and the mixed integer programming (MIP) solver *SCIP 2.0.1* (Achterberg 2004). In order to generate problem instances in the various formats required by the above solvers, we used the tool *Numberjack 0.1.10-11-24* (Hebrard, OMahony, and OSullivan 2010) to create a generic model. This model was solved directly by *Mistral* as the primary

built-in solver for *Numberjack*, and was translated using *Numberjack* into “.cnf” and “.lp” formats for *MiniSat* and *SCIP*, respectively. All experiments were conducted on Intel Xeon CPU E5410 machines, 2.33GHz with 8 cores and 32GB of memory, running Ubuntu. We executed only one run per machine at a time to prevent unintended interference between runs. The time limit for all solvers was 10 minutes.

The translation to CNF format for *MiniSat* is similar to the one used by *Sugar* (Tamura, Tanjo, and Banbara 2008). It creates Boolean variables to capture relations of the form $X = v$ and $X \leq v$ for a variable X and a domain value v . The use of propositions for $X \leq v$ eliminates the quadratic blowup usually associated with the direct encoding of a CSP variable into a set of Boolean variables. It also allows implicitly for domain splitting as a branching strategy. Like *Sugar*, the size of CNF encodings is kept under control (where necessary) by using a “compact” encoding that breaks up long inequalities into a number of smaller ones consisting of only three variables each, by introducing auxiliary variables. E.g., $X_1 + X_2 + X_3 + X_4 + X_5 \leq 20$ is broken down into $Y_1 = X_1 + X_2$; $Y_2 = Y_1 + X_3$; $Y_2 + X_4 + X_5 \leq 20$ in our compact encoding. The market split problem brings out an interesting trade-off between encoding compactness and propagation efficiency, as we will shortly see.

Problem Domains and Results

Our main finding is that *CMVSAT-2* is a robust general method that works well across a variety of domains. Each of the other solvers we consider (SAT, CSP, and MIP solvers) may perform somewhat better than *CMVSAT-2* on one domain but they suffer from a significant disadvantage in other domains. The results, averaged over 100 instances in each domain considered, are summarized in Table 1. In all cases, we used *impact based variable selection* for *CMVSAT-2*. For *value selection*, selecting the value with the least impact worked well in general, although the “disco-novogogo” heuristic (Sellmann and Ansótegui 2006) was better on nqueens-w.

The *qwh-c* domain adds costs to the classic quasigroup with holes problem. An instance is defined by an $N \times N$ array A some of whose entries are filled with numbers from $\{1..N\}$ and the rest are “holes”. The task is to fill $A_{i,j}$ with a number $X_{i,j}$ from $\{1..N\}$ such that no number appears twice in any row or any column of A , and X matches the already filled entries of A . In addition, we assign uniform random costs $C_{i,j}$ from $\{1..10\}$ to each cell and define $\text{cost}(X) = \sum_{1 \leq i,j \leq N} C_{i,j} X_{i,j}$. Given A , C , and a cost threshold T , the problem is to find $X_{i,j}$ such that $\text{cost}(X) \leq T$. For our experiments, we used $N = 25$ with 40% filled entries in A (i.e., 375 holes) and a cost threshold of 50% of $\sum_{i,j} C_{i,j}$. In this regime, the unweighted problem is close to the known feasibility and hardness threshold, but still allows enough flexibility to have several solutions within the 50% cost requirement.

The results for the *qwh-c* domain are summarized in the first row of Table 1. We see that *CMVSAT-2* is able to solve all 100 instances, requiring an average of only 0.54 seconds. The CNF translation for these instances, even with the “compact” encoding, was too large to generate or to read using *MiniSat*, as the cost inequality involves 625 variables with

problem domain	CMVSAT			MiniSat			Mistral			SCIP		
	solved	runtime	nodes	solved	runtime	nodes	solved	runtime	nodes	solved	runtime	nodes
qwh-c	100	0.54s	233	0	MEM	MEM	36	$\geq 403.05s$	$\geq 2.9M$	99	$\geq 58.78s$	≥ 65
misp-3	100	1.16s	10,826	100	19.47s	5,361	100	0.02s	7,527	100	0.54s	4,732
nqueens-w	100	10.33s	2,642	100	0.07s	2,018	92	$\geq 125.51s$	$\geq 2.3M$	100	105.21s	110
all	300	4.01s	–	200	$\geq 206.51s$	–	228	$\geq 176.19s$	–	299	$\geq 54.84s$	–

Table 1: Summary of results with a 10 minute cutoff. The numbers reported are the average over 100 instances in each domain of the number of timeouts (TO), the runtime, and the number of search nodes processed by each solver. The last row, marked “all”, represents the total number of instances solved and the average runtime computed with a 10 minute cutoff.

coefficients and variable values ranging in $\{1..10\}$. SCIP can solve 99 instances but needs roughly a minute on average, while Mistral shows extreme variation in runtime and times out on 64 instances.

The **misp-3** domain represents the market split problem of order $M = 3$. In a nutshell, the problem is defined by set of M weight vectors of size $N = 10 \times (M - 1)$ each. The weights $W_{i,j}$ drawn randomly from $\{1..100\}$. The task is to find an assignment to N binary variables X_j such that for all $i \in \{1..M\}$, we have $\sum_{1 \leq j \leq N} W_{i,j} X_j = \lfloor (\sum_j W_{i,j}) / 2 \rfloor$. This domain is notoriously hard for constraint solvers. In our 100 test instances generated at random, 10 were satisfiable and 90 unsatisfiable.

As we see from Table 1, CMVSAT-2 can solve all 100 instances in roughly 1 second on average. Mistral and SCIP perform better on this domain. However, MiniSat is slower than CMVSAT-2 on average by 20x using the non-compact encoding. Interestingly, while MiniSat is in general able to process many more nodes per second than CMVSAT-2, the size of the input files significantly limits its propagation engine despite techniques such as watched literals. Here, MiniSat takes roughly 20x longer to process roughly half as many nodes as CMVSAT-2, because of sheer encoding size. The compact CNF encoding, while significantly reducing the size, apparently hinders propagation as it caused 57 instances to time out.

Finally, the **nqueens-w** domain adds weights to the classic N-queens problem, where the task is to place N queens on an $N \times N$ chessboard such that no two queens attack each other. We assign uniform random weights $W_{i,j}$ to the cells and define the weight of a queen placement to be the sum of the weights of the cells the queens occupy. Given a weight threshold T , the task is to find a placement of queens with weight at least T . We use $N = 30$ and 70% of the sum of the weights of all cells as the threshold, where the problem almost always has solutions but is not easy to solve with CSP and MIP solvers.

As Table 1 shows, CMVSAT-2 can solve all 100 instances within an average of roughly 10 seconds each. MiniSat fares better here, while Mistral times out on 8 instances and SCIP is slower than CMVSAT-2 on average by 10x.

Overall, Table 1 demonstrates that our general framework is capable of good performance across a variety of domains, while other solvers do not generalize as well. The last line of Table 1, for example, shows that CMVSAT-2 was able to solve all 300 instances considered within an average of only 4 seconds, while MiniSat could solve only 200 in over 200 seconds on average, Mistral could solve only 228 in

over 176 seconds on average, and SCIP could solve 299 but needed over 54 seconds on average.

Conclusion

This work presented a generalized framework for performing SAT-style conflict driven nogood-learning in any setting where primitive and secondary constraints (and the corresponding consistency notions) supported by the system satisfy certain sufficient conditions. The framework combines the benefits of multi-valued variables and pseudo-Boolean solvers, and takes nogood-learning one step further. Unlike SAT modulo theories (SMT), this work deals with generalized constraints that are native to the solver, and performs nogood-learning directly on them. Our empirical results attest to the robustness of the proposed approach.

References

- Achterberg, T. 2004. SCIP - a framework to integrate Constraint and Mixed Integer Programming. Technical report.
- CSP Competition. 2008,2009. International CSP competition result pages.
- Dixon, H. E.; Ginsberg, M. L.; and Parkes, A. J. 2004. Generalizing Boolean satisfiability I: Background and survey of existing work. *JAIR* 21:193–243.
- Eén, N., and Sörensson, N. 2004. An Extensible SAT-solver. In *SAT-2005*, volume 2919 of *LNCS*. chapter 37, 333–336.
- Een, N., and Sörensson, N. 2005. MiniSat v1.13 - A SAT Solver with Conflict-Clause Minimization, System description for the SAT competition.
- Gervet, C., and Van Hentenryck, P. 2006. Length-lex ordering for set cps. In *AAAI-06*, 48–53.
- Hebrard, E.; OMahony, E.; and OSullivan, B. 2010. Constraint Programming and Combinatorial Optimisation in Numberjack. *CPAIOR-2010* 181–185.
- Jain, S.; O’Mahony, E.; and Sellmann, M. 2010. A complete multi-valued sat solver. In *CP-2010*, 281–296.
2006. Mistral web page.
- Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2006. Solving sat and sat modulo theories: From an abstract davis-putnam-logemann-loveland procedure to dpll(t). *J. ACM* 53(6):937–977.
- Ohrimenko, O.; Stuckey, P. J.; and Codish, M. 2007. Propagation = lazy clause generation. In *CP-07*. Springer-Verlag.
- Sellmann, M., and Ansótegui, C. 2006. Disco-Novo-GoGo: Integrating local search and complete search with restarts. In *AAAI-06*.
- Tamura, N.; Tanjo, T.; and Banbara, M. 2008. System description of a SAT-based CSP solver Sugar. *Proceedings of the Third International CSP Solver Competition* 71–75.