Boosting weighted CSP resolution with BDDs

Student: Miquel Palahí,
Miquel Bofill, Josep Suy, and Mateu Villaret

Departament d'Informàtica i Matemàtica Aplicada
Universitat de Girona, Spain
{`mpalahi,mbofill,suy,villaret`}`@ima.udg.edu`

Abstract. We present a new approach for solving Weighted Constraint Satisfaction Problems (WCSP). The method is based on encoding the violation cost of soft constraints as a pseudo-Boolean objective function, and successively calling a decision procedure bounding the maximum allowable cost. The novelty of our approach consists in building a Binary Decision Diagram (BDD) for the objective function, using state-of-the-art generalized arc-consistent CNF encodings for it and, especially, maximising the reuse of the BDDs for the objective function between the successive calls to the decision procedure. The method has been incorporated into the WCSP solving system WSimply, based on reformulation into SMT, with preliminary encouraging results.

1 Introduction

A Constraint Satisfaction Problem (CSP) is a decision problem where the objective is to determine whether an assignment of values to a set of variables exists which satisfies a given set of constraints. It is usually to find CSPs where, additionally to determine if there exists a solution for the problem, the possible solution has to minimize or maximize some objective function. These kind of CSP are known as Constraint Optimization Problems (COP).

Occasionally, some real-world CSP instances have no solution. In such situations we can relax the CSP by allowing the violation of a subset of the constraints, and try to maximize the number of satisfied constraints. This CSP variant is known as Maximum CSP (MaxCSP) [8]. Furthermore, there can exist preferences over which constraints to violate. A convenient way of expressing these preferences is by giving a weight to each constraint, denoting its violation cost. The constraints that can be violated (the ones with a non-infinite weight) are usually called *soft*, while those constraints that must be satisfied are called *hard*. Then, the objective is to find an assignment which satisfies all hard constraints and minimizes the aggregated cost of the violated soft constraints [10]. These problems are known as Weighted CSP (WCSP) [9] or, alternatively, as Cost Function Networks (CFN) [6].

`WSimply` [2, 3] is a language and system for solving intensionally represented WCSPs by reformulation into Satisfiability Modulo Theories (SMT) [11], namely, into SAT Modulo Linear Integer Arithmetic. `WSimply` takes profit from the expressiveness of the SMT language and the performance of current SMT solvers.

However, SMT solvers are decision procedures, designed to check satisfiability of logical formulae with respect to background theories and they scarcely support optimization. For this reason, optimization is implemented in `WSimply` by means of successive calls to the decision procedure either by means of binary search or by means of core based algorithms (WPM1 [4], etc.).

In this paper we introduce a new optimization approach for WCSP, based on representing an objective function (generated from the violation cost of the soft constraints) as a BDD. This allows us to encode the objective function as a pure propositional formula, following the generalized arc-consistent encodings proposed in [1]. This way, we tighten the link between optimization and the logical structure of the problem, with the hope of benefiting from crucial capabilities of the underlying solver, such as conflict driven learning.

An interesting aspect of our approach is the reutilization of BDDs in successive calls to the decision procedure. Changing the bounds of the objective function implies building a new BDD, but some parts can be easily reused. This allows not only to improve the performance of the BDD construction algorithm, but also to keep a number of learned clauses from the solver.

Some preliminary but encouraging results of the implementation of this approach on `WSimply` are given.

2 Preliminars

A *Binary Decision Diagram (BDD)* is a data structure that is used to represent a Boolean function. It can be represented as a rooted, directed, acyclic graph, where each node represents a Boolean variable x which has two child nodes, whose edges represent the *true* and the *false* assignment of x. From now on, we will use the true child (false child) to refer the child node linked by the *true* (*false*) edge. The terminal nodes are called 0-terminal and 1-terminal, representing the satisfiability of the formula. A BDD is called *ordered* if different variables appear in the same order on all paths from the root. A BDD is said to be *reduced* if the following two rules have been applied to its graph:

- Merge any isomorphic subgraphs.
- Eliminate any node whose two children are isomorphic.

The advantage of a *Reduced Ordered Binary Decision Diagram* (ROBDD) is that it is canonical (unique) for a particular function and variable order.

Pseudo-Boolean (PB) constraints [5] are constraints of the form $a_1x_1 + \cdots + a_nx_n \ \# \ K$, where a_is and K are integer coefficients, the x_is are Boolean (0/1) variables, and the relation operator $\#$ belongs to $\{<, >, \leq, \geq, =\}$. We will assume that $\#$ is $\leq$ and the a_is and K are positive, since other cases can be easily reduced to this one.

There exist several BDD-based approaches which reformulate PB constraints into SAT clauses [7]. We focus on the recent work [1] that proposes a simple an efficient algorithm to construct ROBDD and a corresponding Generalized Arc Consistent (GAC) SAT encoding for monotonic Boolean functions.

The key point of that ROBDD construction algorithm are the PB intervals. Let C be a constraint of the form $a_1x_1 + \cdots + a_nx_n \ \le \ K$. The *interval of* C is the set of all integers M such that the constraint $a_1x_1 + \cdots + a_nx_n \ \le \ M$, seen as a Boolean function, is equivalent to C (i.e. that the corresponding Boolean functions have the same truth table). For instance, the interval of $2x_1 + 3x_2 + 4x_3 \le 7$ is $[7, 8]$. Since each node in a BDD represents a PB constraint, we can naturally overload intervals and refer also to intervals of nodes.

The algorithm from [1] is a dynamic programming algorithm that works with *layers*: for a given variable ordering of a PB constraint, say $x_1, x_2, \ldots, x_n$, a list of layers $\mathcal{L} = L_1, \ldots, L_{n+1}$ is created. For every $i \in \{1, 2, \ldots, n+1\}$ the layer L_i will contain pairs $([\beta, \gamma], \mathcal{B})$, where $\mathcal{B}$ is the ROBDD of the constraint $a_ix_i + \cdots + a_nx_n \ \le \ K$ for every $K \in \{\beta..\gamma\}$. The algorithm works as follows, an initial procedure initializes $\mathcal{L}$, with the pairs of the 0-terminal and 1-terminal nodes, and calls the construction of the ROBDD procedure with the following parameters: a PB constraint $a_ix_i + \cdots + a_nx_n \ \le \ K$, the list of layers $\mathcal{L}$ and an index i of the current layer. The first step consists in searching in layer L_i if there exists a ROBDD $\mathcal{B}$ for the current K, i.e, if a pair $([\beta, \gamma], \mathcal{B})$ with $K \in \{\beta..\gamma\}$ exists in L_i. If so, the existing pair is returned, otherwise the procedure is recursively called for the two descendants increasing the index layer to $i+1$ and updating the K of the true child to $K - a_i$. Once the two descendants' ROBDD are returned a new pair for the layer L_i is created. If the two returned ROBDDs are different, the $\mathcal{B}$ of the new pair will be a new ROBDD created from them, otherwise $\mathcal{B}$ will be the returned ROBDD of the children.

With the ROBDD constructed, we only have to encode it to SAT. As usual the encoding introduces an auxiliary variable for every node. Let v be a node with selector variable x and auxiliary variable n. Let f be the variable of its false child and t be the variable of its true child. We need two clauses per node:

$$f \rightarrow \bar{n} \qquad \bar{t} \wedge x \rightarrow \bar{n}$$

Then, to obtain a GAC encoding, we add a unit clause forcing the variable of the root node to be *true*.

Details of the BDD construction or the SAT encoding can be found in [1].

3 WCSP solving through ROBDDs

Since the objective of a WCSP consists in finding a solution where the sum of the weights of the violated constraints is minimal, what we do is to reformulate our WCSP into a COP having as objective function to be minimized this sum:

$$w_1b_1 + \cdots + w_nb_n \tag{1}$$

where w_i is the weight (violation cost) of the constraint c_i and b_i is 1 when constraint c_i is falsified otherwise 0. In other words, the new COP has all original WCSP hard constraints plus new hard constraints like:

$$c_i \leftrightarrow \overline{b_i}$$

for each original WCSP soft constraint c_i with weight w_i, and with the above mentioned sum (1) as objective function. We refer to all this constraints without the objective function as φ.

The objective function can be easily represented in SAT modulo linear integer arithmetic. However, in order to tighten the link between optimization and the logical structure of the problem we propose a new optimization method based on SAT encodings for PB constraints. With this encoding we hope to benefit from crucial capabilities of the underlying solver, such as conflict driven learning.

3.1 Optimization algorithm

The main idea of the new method is to take the maximum advantage of the ROBDD reuse that is done in the algorithm explained in the previous section. Note that when the ROBDD construction procedure finishes, $\mathcal{L}$ will contain all the ROBDD nodes with their corresponding intervals. Since the objective function of a WCSP will always have the same variable ordering and coefficients (we only change the K), $\mathcal{L}$ can be shared during all the optimization process. As we have seen, in most of the cases the existing ROBDD nodes will be reused, specially at the deepest layers.

In Figure 1 there is an example showing how the ROBDD nodes can be reused from one iteration to another. The first diagram of the figure corresponds to the ROBDD of the constraint $2x_1 + 3x_2 + 4x_3 \leq 7$ and the second one corresponds to the ROBDD of the constraint $2x_1 + 3x_2 + 4x_3 \leq 5$, where only one new node needs to be created because almost all the existing nodes of the first ROBDD can be reused.

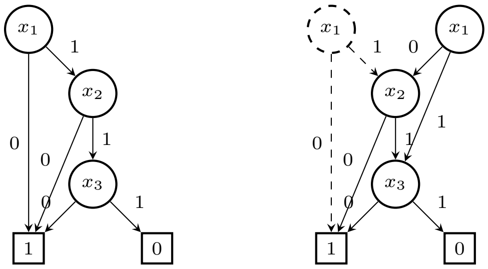

Fig. 1. ROBDDs for $2x_1 + 3x_2 + 4x_3 \leq 7$ (left) and $2x_1 + 3x_2 + 4x_3 \leq 5$ (right).

The optimization algorithm is a binary search algorithm that uses the SAT encoding of the above mentioned ROBDD to constraint the objective function during the search.

First of all, we initialize the list of layers $\mathcal{L}$ using the objective function (1). After that we start the binary search, where in the first step we call the ROBDD construction method, described in the previous section, with the corresponding K of the iteration and $\mathcal{L}$ as an input/output parameter. This way, we will have in $\mathcal{L}$ all the computed ROBDDs for the following iterations of the search, significantly reducing the construction time. The procedure returns the ROBDD $\mathcal{B}$ representing the objective function for the specific K of the iteration. In the next step we generate the SAT clauses from the new nodes of $\mathcal{B}$ as explained previously, and insert them into φ. Then we insert the variable of the root node of $\mathcal{B}$ into φ as a unit clause to make the encoding GAC. At this point we call the SMT solver to check the satisfiability of φ. If φ is satisfiable we can keep all the learned clauses, otherwise we only have to remove the unitary clause with the root node variable. This way we will only remove the learned clauses related to this clause. In the last step we update the bounds and repeat these steps until the optimal solution is found.

4 Benchmarks and further work

We test the performance of the new solving method with a softened version of the well-known balanced academic curriculum problem (BACP), the Soft BACP (SBACP) [2, 3].

In SBACP the number of periods have been reduced until all the instances are unsatisfiable due to the prerequisites chain, and then the prerequisites are considered soft constraints. We use the five variants of the SBACP presented in [2, 3] for this preliminary performance study.

First of all, we compare the performance of the new solving method (noted as BDD) with the previously best solving method of `WSimply` (which uses a linear integer arithmetic constraint and is noted as LIA). Table 2 contains the aggregated solving time of the 28 instances for each variant. Clearly the new method outperforms the old one, specially in the first variant where it is almost 9 times faster.

	sbacp	*sbacp_h1*	*sbacp_h2*	*sbacp_h2_ml2*	*sbacp_h2_ml3*
LIA	112.27	8.46	36.70	570.86	886.76
BDD	13.61	8.47	13.08	132.57	150.85

Fig. 2. Aggregated times for the 5 set of instances of the SBACP variants.

In Figure 3 we present two plots showing (left) the average of the percentage of reused nodes per iteration for the hardest SBACP variant (*sbacp_h2_ml3*), (right) the average of the aggregated time (in percentage) per iteration for the same instances. With respect to the reused nodes plot, notice that in iteration 5 the percentage of reused nodes begins to increase significantly, and from iteration 9 almost 100% of the nodes are reused. On the other hand, in the aggregated time plot we can observe a tendency to decrease the time needed per iteration from iteration 5. The number of iterations needed to solve the 28 instances ranges from 12 to 18 and in average is 15.80.

As further work we want to study more deeply the method with other problems, and extend the method for integer variables.

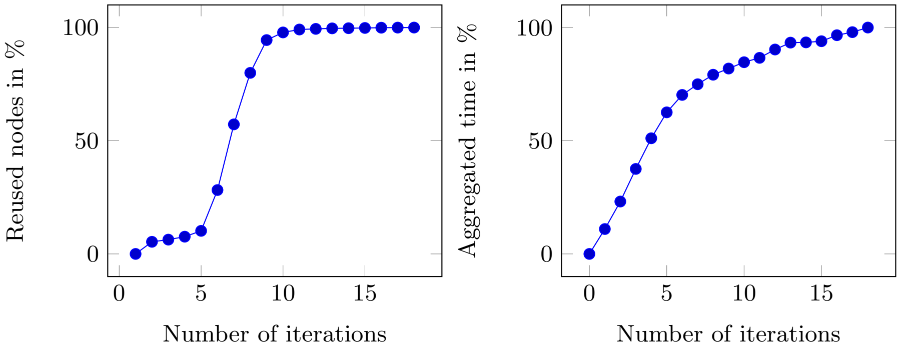

Fig. 3. Percentage of nodes reused and aggregated time per iteration of the *sbacp_h2_ml3* variant.

References

1. I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and V. Mayer-Eichberger. A new look at bdds for pseudo-boolean constraints. *J. Artif. Intell. Res. (JAIR)*, 45:443–480, 2012.
2. C. Ansótegui, M. Bofill, M. Palahí, J. Suy, and M. Villaret. A Proposal for Solving Weighted CSPs with SMT. In *Proceedings of the 10th International Workshop on Constraint Modelling and Reformulation (ModRef 2011)*, pages 5–19, 2011.
3. C. Ansótegui, M. Bofill, M. Palahí, J. Suy, and M. Villaret. Solving weighted csps with meta-constraints by reformulation into satisfiability modulo theories. *Constraints*, 18(2):236–268, 2013.
4. C. Ansótegui, M. L. Bonet, and J. Levy. Solving (weighted) partial maxsat through satisfiability testing. In O. Kullmann, editor, *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 427–440. Springer, 2009.
5. E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155–225, 2002.
6. S. de Givry, M. Zytnicki, F. Heras, and J. Larrosa. Existential arc consistency: getting closer to full arc consistency in weighted CSPs. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 84–89, 2005.
7. N. En and N. Srensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.
8. E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1-3):21 – 70, 1992.
9. J. Larrosa and T. Schiex. Solving Weighted CSP by Maintaining Arc-Consistency. *Artificial Intelligence*, 159(1–2):1–26, 2004.
10. P. Meseguer, F. Rossi, and T. Schiex. Soft constraints. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 9. Elsevier, 2006.
11. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.