Modelling Max-CSP as Partial Max-SAT*

Josep Argelich[1], Alba Cabiscol[1], Inês Lynce[2], and Felip Manyà[1]

[1] Computer Science Department
Universitat de Lleida
Jaume II, 69, E-25001 Lleida, Spain
[2] IST/INESC-ID
Technical University of Lisbon
Rua Alves Redol 9, 1000-029 Lisboa, Portugal

Abstract. We define a number of original encodings that map Max-CSP instances into partial Max-SAT instances. Our encodings rely on the well-known direct and support encodings from CSP into SAT. Then, we report on an experimental investigation that was conducted to compare the performance profile of our encodings on random binary Max-CSP instances. Moreover, we define a new variant of the support encoding from CSP into SAT which produces fewer clauses than the standard support encoding.

1 Introduction

In the last years, there has been an increasing interest in the Boolean Max-SAT problem. Taking into account the success of SAT on solving NP-complete decision problems, the SAT community investigates how to transfer the technology created for SAT to Max-SAT with the aim of developing fast Max-SAT solvers, which can be used to solve NP-hard optimization problems via their reduction to Max-SAT.

The most recent and relevant results for Max-SAT can be summarized as follows: (i) there exist solvers like Clone [21], Lazy [1], MaxSatz [18], MiniMaxSat [12], ms4 [19], SR(w) [22] and Max-DPLL [15] which solve many instances that are beyond the reach of the solvers existing just five years ago; (ii) resolution refinements, which preserve the number of unsatisfied clauses, have been incorporated into Max-SAT solvers [14, 15, 18], as well as good quality underestimations of the lower bound [16, 17, 21, 22], (iii) a resolution-style calculus for Max-SAT has been proven to be complete [5, 6], (iv) formalisms like Partial Max-SAT have been investigated for solving problems with soft constraints [2, 8, 12, 3], and (v) two evaluations of Max-SAT solvers have been performed for the first time as a co-located event of the International Conference on Theory and Applications of Satisfiability Testing (SAT-2006 and SAT-2007).

* This research was funded by MEC research projects TIN2006-15662-C02-02, TIN2007-68005-C04-04 and Acción Integrada HP2005-0147, and FCT research projects SATPot (POSC/EIA/61852/2004) and SHIPs (PTDC/EIA/64164/2006).

In this paper, we define a number of original encodings that map Max-CSP instances into partial Max-SAT instances. Our encodings rely on the well-known direct and support encodings from CSP into SAT. Then, we report on an experimental investigation that was conducted to compare the performance profile of our encodings on random binary Max-CSP instances. Interestingly, we also define a new variant of the support encoding from CSP into SAT which produces fewer clauses than the standard support encoding. Our new encoding, called *minimal support encoding*, eliminates redundant clauses.

The objective of our research is to show that different Max-SAT encodings for a same optimization problem may produce substantial differences on performance, as well as to identify features of the encodings that lead to better performance profiles. To the best of our knowledge, this is the first paper that addresses the question of how to encode Max-CSP into Max-SAT, and analyzes the impact of modelling on the performance of Max-SAT solvers.

The structure of the paper is as follows. Section 2 contains preliminary definitions about Max-SAT and Max-CSP. Section 3 surveys the support and direct encodings from CSP into SAT, and defines a new encoding that we call minimal support encoding. Section 4 defines a number of original encodings from Max-CSP into partial Max-SAT. Section 5 reports and analyses the experimental investigation. Section 6 presents the conclusions and future research directions.

2 Preliminaries

2.1 Max-SAT definitions

In propositional logic a variable x_i may take values 0 (for false) or 1 (for true). A literal l_i is a variable x_i or its negation $\bar{x}_i$. A clause is a disjunction of literals, and a CNF formula is a multiset of clauses.

An assignment of truth values to the propositional variables satisfies a literal x_i if x_i takes the value 1 and satisfies a literal $\bar{x}_i$ if x_i takes the value 0, satisfies a clause if it satisfies at least one literal of the clause, and satisfies a CNF formula if it satisfies all the clauses of the formula. An empty clause, denoted by $\Box$, contains no literals and cannot be satisfied.

The Max-SAT problem for a CNF formula ϕ is the problem of finding an assignment of values to propositional variables that maximizes the number of satisfied clauses. In this sequel we often use the term Max-SAT meaning Min-UNSAT. This is because, with respect to exact computations, finding an assignment that minimizes the number of unsatisfied clauses is equivalent to finding an assignment that maximizes the number of satisfied clauses.

We also consider the extension of Max-SAT known as Partial Max-SAT because it is more well-suited for representing and solving NP-hard problems. A Partial Max-SAT instance is a CNF formula in which some clauses are *relaxable* or *soft* and the rest are *non-relaxable* or *hard*. Solving a Partial Max-SAT instance amounts to finding an assignment that satisfies all the hard clauses and the maximum number of soft clauses. Hard clauses are represented between square brackets, and soft clauses are represented between round brackets.

2.2 Max-CSP definitions

Definition 1. *A* Constraint Satisfaction Problem (CSP) *instance is defined as a triple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X} = \{X_1, \ldots, X_n\}$ is a set of variables, $\mathcal{D} = \{d(X_1), \ldots, d(X_n)\}$ is a set of finite domains containing the values the variables may take, and $\mathcal{C} = \{C_1, \ldots, C_m\}$ is a set of constraints. Each constraint $C_i = \langle S_i, R_i \rangle$ is defined as a relation R_i over a subset of variables $S_i = \{X_{i_1}, \ldots, X_{i_k}\}$, called the* constraint scope. *The relation R_i may be represented extensionally as a subset of the Cartesian product $d(X_{i_1}) \times \cdots \times d(X_{i_k})$.*

Definition 2. *An* assignment *v for a CSP instance $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ is a mapping that assigns to every variable $X_i \in \mathcal{X}$ an element $v(X_i) \in d(X_i)$. An assignment v satisfies a constraint $\langle \{X_{i_1}, \ldots, X_{i_k}\}, R_i \rangle \in C$ iff $\langle v(X_{i_1}), \ldots, v(X_{i_k}) \rangle \in R_i$.*

Definition 3. *The Constraint Satisfaction Problem (CSP) for a CSP instance P consists in deciding whether there exists an assignment that satisfies P.*

In the sequel we assume that all CSPs are unary and binary; i.e., the scope of all the constraints has cardinality at most two.

Definition 4. *A CSP is* node consistent, *if for every variable X_i, every value of the domain of X_i is allowed for the unary constraints on X_i. A CSP is* arc consistent, *if for every constraint on two variables X_i and Y_j, for all $a \in d(X_i)$, there exists $b \in d(Y_j)$, such that (a, b) is in the constraint.*

Definition 5. *The Max-CSP problem for a CSP instance $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ is the problem of finding an assignment that minimizes (maximizes) the number of violated (satisfied) constraints.*

3 Encoding CSP into SAT

Mappings of binary CSPs into SAT is an area of research that has been investigated by several authors [4, 9–11, 13, 23]. They have proposed a number of encodings having different performance profiles and achieving different degrees of local propagation on SAT solvers. Among them, the most well-known are the direct encoding and the support encoding. In the rest of this section, we first define the direct encoding and the support encoding, and then define a new encoding from CSP into SAT called *minimal support encoding*.

3.1 Direct encoding and support encoding

In the *direct encoding*, we associate a Boolean variable x_{ij} with each value j that can be assigned to the CSP variable X_i. Assuming that X_i has a domain of size m, the direct encoding contains clauses that ensure that each CSP variable X_i is given a value: for each i, $x_{i1} \vee \cdots \vee x_{im}$ (called *at-least-one* clauses), and contains clauses that rule out any binary nogoods. For example, if $X_1 = 2$ and $X_3 = 1$ is not allowed, then the clause $\overline{x}_{12} \vee \overline{x}_{31}$ (called *conflict* clause) is added.

We consider the version of the direct encoding that adds clauses that ensure that each CSP variable X_i takes no more than one value: for each i, j, k with $j < k$, $\overline{x}_{ij} \vee \overline{x}_{ik}$ (called *at-most-one* clauses). These clauses are redundant, but are considered in the literature in order to maintain a one-to-one mapping between CSP models and SAT models.

In the *support encoding*, the idea is to encode into clauses the *support* for a value instead of encoding conflicts. The support for a value j of a CSP variable X_i across a constraint is the set of values of the other variable in the constraint which allow $X_i = j$. If $v_1, v_2, \ldots, v_k$ are the supporting values of variable X_l for $X_i = j$, we add the clause $\overline{x}_{ij} \vee x_{lv_1} \vee x_{lv_2} \vee \cdots \vee x_{lv_k}$ (called *support* clause). There is one support clause for each pair of variables X_i, X_l involved in a constraint, and for each value in the domain of X_i. Unlike conflict clauses, a clause in each direction is used in the literature, one for the pair X_i, X_l and one for X_l, X_i. The support clauses on their own do not provide a correct encoding of CSPs into SAT. To complete an encoding using support clauses we need to add the at-least-one and at-most-one clauses for each CSP variable to ensure that each CSP variable takes exactly one value of its domain.

Example 1. The direct encoding of the CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle = \langle \{X, Y\}, \{d(X) = \{1, 2, 3\}, d(Y) = \{1, 2, 3\}\}, \{X \leq Y\} \rangle$ contains the following clauses:

at-least-one	$x_1 \vee x_2 \vee x_3$	$y_1 \vee y_2 \vee y_3$	
at-most-one	$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee \overline{x}_3$	$\overline{x}_2 \vee \overline{x}_3$
	$\overline{y}_1 \vee \overline{y}_2$	$\overline{y}_1 \vee \overline{y}_3$	$\overline{y}_2 \vee \overline{y}_3$
conflict	$\overline{x}_2 \vee \overline{y}_1$	$\overline{x}_3 \vee \overline{y}_1$	$\overline{x}_3 \vee \overline{y}_2$

and the support encoding for that CSP contains the at-least-one clauses, the at-most-one clauses, and the following support clauses:

support	$\overline{x}_2 \vee y_2 \vee y_3$	$\overline{y}_1 \vee x_1$
	$\overline{x}_3 \vee y_3$	$\overline{y}_2 \vee x_1 \vee x_2$

The support clause for x_1 is missing because it is subsumed by $y_1 \vee y_2 \vee y_3$, and the support clause for y_3 is missing because it is subsumed by $x_1 \vee x_2 \vee x_3$.

3.2 Minimal support encoding

Our first contribution in this paper is to give a new version of the support encoding, which we call *minimal support encoding*. Our definition follows from the observation that the support encoding contains redundant clauses. More precisely, given a binary constraint C_k with scope $\{X, Y\}$, it is enough to add the support clauses either for the values of X or for the values of Y; it is not necessary to add a clause in each direction. Despite of the number of papers dealing with the support encodings, this fact has gone unnoticed so far.

Definition 6. *The* minimal support encoding *is like the support encoding except for the fact that, for every constraint C_k with scope $\{X, Y\}$, we only add either the support clauses for all the domain values of the CSP variable X or the support clauses for all the domain values of the CSP variable Y.*

Example 2. A minimal support encoding for the CSP instance from Example 1 contains the following clauses:

at-least-one	$x_1 \vee x_2 \vee x_3$	$y_1 \vee y_2 \vee y_3$	
at-most-one	$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee \overline{x}_3$	$\overline{x}_2 \vee \overline{x}_3$
	$\overline{y}_1 \vee \overline{y}_2$	$\overline{y}_1 \vee \overline{y}_3$	$\overline{y}_2 \vee \overline{y}_3$
support	$\overline{x}_2 \vee y_2 \vee y_3$	$\overline{x}_3 \vee y_3$	

Proposition 1. *The* minimal support encoding *is correct.*

Proof: We assume, without loss of generality, that we add the support clauses for all the domain values of the CSP variable X for every constraint C_k with scope $\{X, Y\}$. Given a CSP assignment, we construct its corresponding Boolean assignment by setting the variable x_i to true if the CSP assignment assigns the value i to X; otherwise, we set the variable x_i to false. Given a Boolean assignment that satisfies the minimal support encoding of a CSP, we construct its corresponding CSP assignment by assigning to the CSP variable X the value i if x_i is true. Note that there is exactly one x_i for each CSP variable X which is true because the minimal support encoding contains the at-least-one and at-most-one clauses. So, it is a valid CSP assignment.

We prove first that if a CSP assignment satisfies all the constraints of a CSP instance, then its corresponding Boolean assignment satisfies its minimal encoding. Since a CSP assignment assigns exactly one value to each CSP variable, the Boolean assignment satisfies the at-least-one and at-most-one clauses. For every constraint C_k with scope $\{X, Y\}$, the CSP assignment assigns a value i to X and a value j to Y. Since $(X = i, Y = j)$ is an allowed combination, among the clauses encoding that constraint, there is a clause of the form $\overline{x}_i \vee y_j \vee \cdots$ which is satisfied by the Boolean encoding because y_j is true. The remaining clauses are also satisfied by the Boolean assignment because they are of the form $\overline{x}_l \vee \cdots$, where $l \neq i$, and the Boolean assignment assigns the value false to all variables x_l with $l \neq i$.

We prove now that if a Boolean assignment satisfies the minimal support encoding of a CSP instance P, then its corresponding CSP assignment satisfies P. Assume that the CSP assignment does not satisfy P. Therefore, there exists a constraint C_k of P with scope $\{X, Y\}$ which is violated because the CSP assignment assigns a value i to X and a value j to Y which corresponds to a forbidden combination. In this case, there is exactly one support clause of the form $\overline{x}_i \vee y_{j_1} \vee \cdots \vee y_{j_k}$ among the support clauses encoding C_k which is not satisfied by the Boolean assignment because x_i is true and $y_{j_1} \neq y_j, \ldots, y_{j_k} \neq y_j$. The rest of support clauses encoding C_k are satisfied by the Boolean assignment because it assigns the value false to all variables x_l with $l \neq i$. ■

Unlike the support encoding [11, 13], the minimal support encoding does not maintain arc consistency through unit propagation. Recall that the direct encoding does not maintain arc consistency too.

Proposition 2. *The minimal support encoding does not maintain arc consistency through unit propagation.*

Proof: We give a counterexample to prove the proposition. Given the CSP instance $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X} = \{X, Y\}, d(X) = d(Y) = \{1,2,3\}, \mathcal{C} = \{C_{XY}\} = \{\{(1,1),(2,2),(3,3)\}\}$ with the following minimal support encoding:

at-least-one	$x_1 \vee x_2 \vee x_3$	$y_1 \vee y_2 \vee y_3$	
at-most-one	$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee \overline{x}_3$	$\overline{x}_2 \vee \overline{x}_3$
	$\overline{y}_1 \vee \overline{y}_2$	$\overline{y}_1 \vee \overline{y}_3$	$\overline{y}_2 \vee \overline{y}_3$
support	$\overline{x}_1 \vee y_1$	$\overline{x}_2 \vee y_2$	$\overline{x}_3 \vee y_3$,

if x_1 is set to false, then $\overline{y}_1$ is not derived by unit propagation, and the domain of Y is not arc consistent. Observe that if the support clauses are $\overline{y}_1 \vee x_1, \overline{y}_2 \vee x_2, \overline{y}_3 \vee x_3$, then $\overline{y}_1$ is derived by unit propagation, and the domain of Y becomes arc consistent. However, if y_1 is set to false, then arc consistency is not maintained in the last case. ■

4 Encoding Max-CSP into Partial Max-SAT

4.1 Direct encoding for Partial Max-SAT

Given a CSP instance P, our goal is to define a version of the direct encoding that produces a partial Max-SAT instance ϕ such that the minimum number of constraints of P that are violated by a CSP assignment is exactly the same as the minimum number of clauses of ϕ that are falsified by a Boolean assignment.

Definition 7. *The* direct encoding *of a Max-CSP instance $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ is the Partial Max-SAT instance that contains as hard clauses the corresponding at-least-one and at-most-one clauses for every CSP variable in $\mathcal{X}$, and contains a soft clause $\overline{x}_i \vee \overline{y}_j$ for every nogood $(X = i, Y = j)$ of every constraint of $\mathcal{C}$ with scope $\{X, Y\}$.*

Example 3. The Partial Max-SAT direct encoding for the Max-CSP problem of the CSP instance from Example 1 is as follows:

at-least-one	$[x_1 \vee x_2 \vee x_3]$	$[y_1 \vee y_2 \vee y_3]$	
at-most-one	$[\overline{x}_1 \vee \overline{x}_2]$	$[\overline{x}_1 \vee \overline{x}_3]$	$[\overline{x}_2 \vee \overline{x}_3]$
	$[\overline{y}_1 \vee \overline{y}_2]$	$[\overline{y}_1 \vee \overline{y}_3]$	$[\overline{y}_2 \vee \overline{y}_3]$
conflict	$(\overline{x}_2 \vee \overline{y}_1)$	$(\overline{x}_3 \vee \overline{y}_1)$	$(\overline{x}_3 \vee \overline{y}_2)$

Proposition 3. *Solving a Max-CSP instance is equivalent to solving the Partial Max-SAT problem of its direct encoding.*

Proof: The hard clauses ensure that exactly one of the Boolean variables that encode a CSP variable is true and the rest are false in a feasible assignment. Therefore, we have that there is a one-to-one mapping between the set of CSP assignments and the set of feasible assignments of the Partial Max-SAT instance and, moreover, at most one of the conflict clauses that encode a certain constraint can be falsified by a feasible assignment. If the CSP assignment satisfies a constraint, then the corresponding Boolean assignment also satisfies the conflict clauses that encode that constraint because there is no clause forbidding allowed values. If the CSP assignment violates a constraint, then the corresponding Boolean assignment does not satisfy the conflict clause that encodes the forbidden values of the two variables involved in the constraint, and satisfies the remaining clauses. ■

There are other options for defining the direct encoding which amount to introducing auxiliary variables. For example, you can add all the clauses representing nogoods as hard clauses by adding an auxiliary literal c_i to every clause encoding a nogood of every constraint $C_i \in \mathcal{C}$, and adding the unit clause $\overline{c}_i$ as a soft clause. Nevertheless, we do not consider this encoding because we realized that its performance profile is worse than the performance profile of the direct encoding (at least for the benchmarks considered in our empirical evaluation).

4.2 Support encoding for Partial Max-SAT

The support encoding for Partial Max-SAT may be defined by adapting the minimal support encoding from CSP into SAT:

Definition 8. *The* minimal support encoding *of a Max-CSP instance $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ is the Partial Max-SAT instance that contains as hard clauses the corresponding at-least-one and at-most-one clauses for every CSP variable in $\mathcal{X}$, and contains as soft clauses the support clauses of the minimal support encoding from CSP into SAT.*

Example 4. A minimal Partial Max-SAT support encoding for the Max-CSP problem of the CSP instance from Example 1 contains the following clauses:

at-least-one	$[x_1 \vee x_2 \vee x_3]$	$[y_1 \vee y_2 \vee y_3]$	
at-most-one	$[\overline{x}_1 \vee \overline{x}_2]$	$[\overline{x}_1 \vee \overline{x}_3]$	$[\overline{x}_2 \vee \overline{x}_3]$
	$[\overline{y}_1 \vee \overline{y}_2]$	$[\overline{y}_1 \vee \overline{y}_3]$	$[\overline{y}_2 \vee \overline{y}_3]$
support	$(\overline{x}_2 \vee y_2 \vee y_3)$	$(\overline{x}_3 \vee y_3)$	

Proposition 4. *Solving a Max-CSP instance is equivalent to solving the Partial Max-SAT problem of its minimal support encoding.*

PROOF: Proposition 1 proves that there is one unsatisfied clause for every violated constraint. Since the minimal support encoding is correct, and the hard clauses ensure a one-to-one mapping between Max-CSP and feasible Partial Max-SAT assignments, the optimal solutions of Max-CSP are exactly the same as the optimal solutions of Partial Max-SAT. ■

We now define how to adapt to Partial Max-SAT the support encoding from CSP into SAT.

Definition 9. *The* support encoding *of a Max-CSP instance $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ is the Partial Max-SAT instance that contains as hard clauses the corresponding* *at-least-one and at-most-one clauses for every CSP variable in $\mathcal{X}$, and contains, for every constraint $C_k \in \mathcal{C}$ with scope $\{X, Y\}$, a soft clause of the form $S_{X=j} \vee c_k$ for every support clause $S_{X=j}$ encoding the support for the value j of the CSP variable X, where c_k is an auxiliary variable , and contains a soft clause of the form $S_{Y=m} \vee \overline{c}_k$ for every support clause $S_{Y=m}$ encoding the support for the value m of the CSP variable Y.*

Observe that we introduce an auxiliary variable for every constraint. This is due to the fact that there are two unsatisfied soft clauses for every violated constraint of the Max-CSP instance if we do not introduce auxiliary variables. It is particularly important to have one unsatisfied clause for every violated constraints when mapping weighted Max-CSP instances into weighted Max-SAT instances.[1] In this case, all the clauses encoding a certain constraint have as weight the weight associated to that constraint. When a constraint is violated with weight w, this guarantees that there is exactly one unsatisfied clause with weight w.

Example 5. The Partial Max-SAT support encoding for the Max-CSP problem of the CSP instance from Example 1 is as follows:

at-least-one	$[x_1 \vee x_2 \vee x_3]$	$[y_1 \vee y_2 \vee y_3]$	
at-most-one	$[\overline{x}_1 \vee \overline{x}_2]$	$[\overline{x}_1 \vee \overline{x}_3]$	$[\overline{x}_2 \vee \overline{x}_3]$
	$[\overline{y}_1 \vee \overline{y}_2]$	$[\overline{y}_1 \vee \overline{y}_3]$	$[\overline{y}_2 \vee \overline{y}_3]$
support	$(\overline{x}_2 \vee y_2 \vee y_3 \vee c_1)$	$(\overline{y}_1 \vee x_1 \vee \overline{c}_1)$	
	$(\overline{x}_3 \vee y_3 \vee c_1)$	$(\overline{y}_2 \vee x_1 \vee x_2 \vee \overline{c}_1)$	

Proposition 5. *Solving a Max-CSP instance is equivalent to solving the Partial Max-SAT problem of its support encoding.*

Proof: By introducing auxiliary variables we ensure that the optimal solutions of Max-CSP are exactly the same as the optimal solutions of Partial Max-SAT. The auxiliary variables allow to violate exactly one clause for every violated constraint. ■

In the following proposition we assume that Partial Max-SAT solvers incorporate the rule that replaces any two complementary unit clauses with an empty clause. Actually, most of the solvers we know implement such a rule.

Proposition 6. *When solving a Max-CSP instance with the support encoding on a Partial Max-SAT solver, it is not necessary to branch on the auxiliary variables.*

Proof: For every violated constraint C_k with scope $\{X, Y\}$, there is exactly one unsatisfied support clause of the form $\overline{x}_i \vee y_{j_1} \vee \cdots \vee y_{j_k}$ and one unsatisfied support clause of the form $\overline{y}_l \vee x_{m_1} \vee \cdots \vee x_{m_s}$ in the support encoding from

[1] In weighted Max-CSP (Max-SAT), each constraint (clause) has a weight and the goal is to minimize the sum of the weights of the violated constraints (falsified clauses).

CSP into SAT. Therefore, these clauses will produce the derivation of the two complementary unit clauses in the support encoding from Max-CSP into Partial Max-SAT: c_k (from $\overline{x}_i \vee y_{j_1} \vee \cdots \vee y_{j_k} \vee c_k$) and $\overline{c}_k$ (from $\overline{y}_l \vee x_{m_1} \vee \cdots \vee x_{m_s} \vee \overline{c}_k$). The solver will then derive a contradiction from these two clauses. If C_k is satisfied, both support clauses are satisfied and the fact of adding an extra literal does not affect their satisfaction. ■

On the solved benchmarks we did not see significant differences between branching including auxiliary variables and branching without including them. So, we only report results for branching including auxiliary variables. However, there may exist differences on other types of instances and solvers.

5 Experimental results

We conducted an empirical evaluation to assess the impact of the defined encodings on the performance of two of the best performing Partial Max-SAT solvers: MiniMaxSat [12] and PMS [3]. Moreover, we compared the support encoding and the minimal support encoding when solving SAT-encoded CSP instances with MiniSat [7] and zChaff [20]. The evaluation was conducted on a cluster with 160 2 GHz AMD Opteron 248 Processors with 1 GB of memory.

As benchmarks we considered binary CSPs, which were obtained with a generator of uniform random binary CSPs[2] —designed and implemented by Frost, Bessière, Dechter and Regin— that implements the so-called model B: in the class $\langle n, d, p_1, p_2 \rangle$ with n variables of domain size d, we choose a random subset of exactly $p_1 n(n-1)/2$ constraints (rounded to the nearest integer), each with exactly $p_2 d^2$ conflicts (rounded to the nearest integer); p_1 may be thought of as the *density* of the problem and p_2 as the *tightness* of constraints. The difficulty of the instances depends on the selected values for n, d, p_1 and p_2. We selected values that allowed to solve the instances in a reasonable amount of time in each solver.

We used the following encodings: the direct encoding (`dir`), the support encoding (`supxy`), and three variants of the minimal support encoding (`supx`, `supl`, `supc`). The encoding `supx` refers to the minimal support encoding of a binary CSP containing only the support clauses for the CSP variable X and not for the variable Y for every constraint with scope $\{X, Y\}$; we do not show results for the encoding containing only support clauses for the CSP variable Y because its behaviour is very close to `supx` for the solved random instances. The encoding `supl` refers to the minimal support encoding containing, for each constraint, the support clauses for the variable that produces a smaller total number of literals. The encoding `supc` refers to the minimal support encoding containing, for each constraint, the support clauses for the variable that produces smaller size clauses; we give a score of 16 to unit clauses, a score of 4 to binary clauses and a score of 1 to ternary clauses, and choose the variable with higher sum of scores. For instance, given the CSP instance $\langle X, D, C \rangle$, where $X = \{X, Y\}, d(X) =$

[2] http://www.lirmm.fr/~bessiere/generator.html

$d(Y) = \{1, 2, 3, 4\}, C = \{C_{XY}\} = \{\{(1, 2), (1, 3), (1, 4)\}\}$, `supc` prefers three binary support clauses $x_1 \vee \overline{y}_2, x_1 \vee \overline{y}_3, x_1 \vee \overline{y}_4$ rather than the quaternary support clause $\overline{x}_1 \vee y_2 \vee y_3 \vee y_4$, while `supl` prefers $\overline{x}_1 \vee y_2 \vee y_3 \vee y_4$.

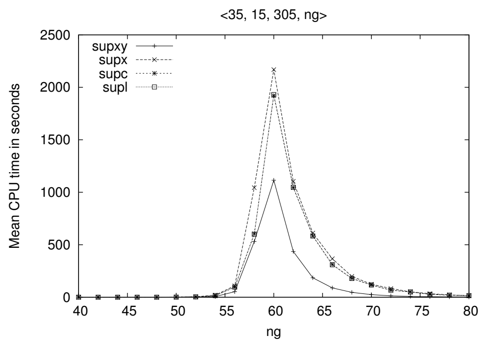

Fig. 1. Experimental results for MiniSat

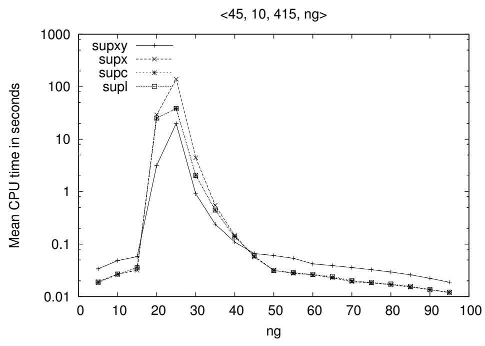

Fig. 2. Experimental results for zChaff (log scale)

In the first experiment we solved 100 CSP instances with MiniSat and zChaff for each data point. We compared all the support encodings from CSP into SAT. With MiniSat, we solved CSP instances with 35 variables, domains of 15 elements, 305 constraints and variable tightness (we vary the number of nogoods (ng)). The obtained results are shown in Figure 1. With zChaff, we solved CSP instances with 45 variables, domains of 10 elements, 415 constraints and variable tightness. The obtained results are shown in Figure 2. We observe that the support encoding outperforms the three variants of the minimal support encoding.

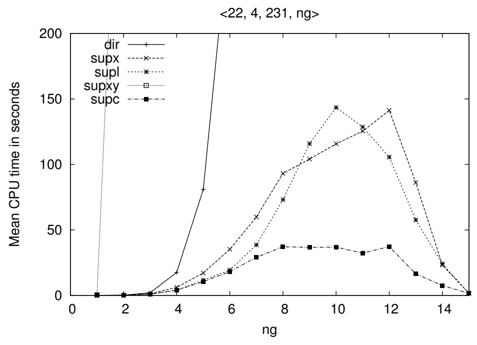

Fig. 3. Experimental results for MiniMaxSat

We believe that this is due to the fact that the support encoding, unlike the minimal support encoding, maintains arc consistency through unit propagation.

In the second experiment we solved 100 Max-CSP instances with MiniMaxSat for each data point; the instances had 22 variables, domains of 4 elements, 231 constraints and variable tightness. We compared all the defined encodings of Max-CSP into Partial Max-SAT. The obtained results are shown in Figure 3.

In the third experiment we solved 100 Max-CSP instances with MiniMaxSat for each data point; the instances had 25 variables, domains of 5 elements, 150 constraints and variable tightness. We compared all the defined encodings of Max-CSP into Partial Max-SAT. The obtained results are shown in Figure 4. We omit the results for the encodings `dir` and `supxy` because they are not competitive.

In the fourth experiment we solved 100 Max-CSP instances with PMS for each data point; the instances had 15 variables, domains of 4 elements, 120 constraints and variable tightness. We compared all the defined encodings of Max-CSP into Partial Max-SAT. The obtained results are shown in Figure 5.

In the fifth experiment we solved 100 Max-CSP instances with PMS for each data point; the instances had 14 variables, domains of 5 elements, 91 constraints and variable tightness. We compared all the defined encodings of Max-CSP into Partial Max-SAT. The obtained results are shown in Figure 6.

We observe that support encodings from Max-CSP into Partial Max-SAT, which have been introduced for the first time in this paper, outperform the direct encoding for both solvers. In MiniMaxSat, the best performing encoding is the minimal support encoding. Among the different versions of the minimal support encoding, we observe that `supc` is up to 6 times faster than the other two encodings (`supx` and `supl`). In PMS, the best encoding for high values of tightness is the support encoding while the best encodings for lower values are `supc` and `supl` in Figure 5, and `supl` in Figure 6.

Finally, in Table 1 we show, for each different encoding, the average number of clauses of some sets of Max-CSP instances solved in the third experiment

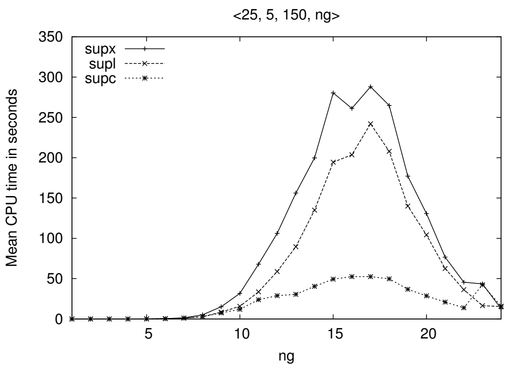

Fig. 4. Experimental results for MiniMaxSat

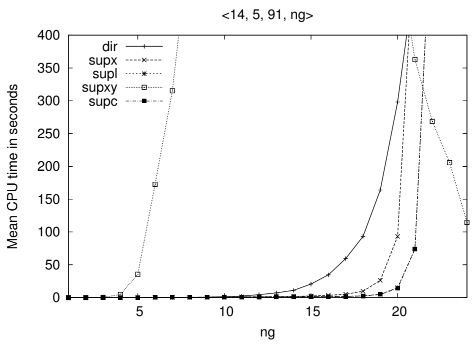

Fig. 5. Experimental results for PMS

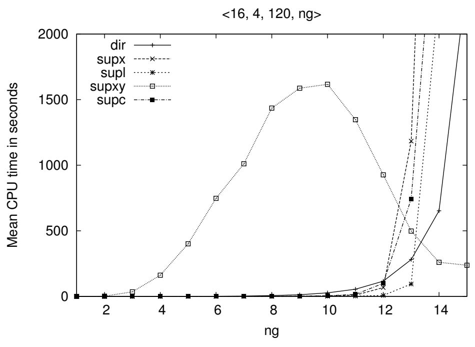

Fig. 6. Experimental results for PMS

(Figure 4) in order to illustrate the differences on the number of clauses among all the defined encodings from Max-CSP into Partial Max-SAT. Observe that encodings `supx`, `supl` and `supc` produce instances with a similar number of clauses.

Parameters	dir	supxy	supx	supc	supl
$\langle 25, 5, 150, 2\rangle$	575	824	551	549	524
$\langle 25, 5, 150, 4\rangle$	875	1201	738	720	685
$\langle 25, 5, 150, 6\rangle$	1175	1445	861	826	804
$\langle 25, 5, 150, 8\rangle$	1475	1602	939	905	890
$\langle 25, 5, 150, 10\rangle$	1775	1690	983	959	950
$\langle 25, 5, 150, 12\rangle$	2075	1739	1007	993	991
$\langle 25, 5, 150, 14\rangle$	2375	1762	1019	1012	1012
$\langle 25, 5, 150, 16\rangle$	2675	1771	1023	1021	1021
$\langle 25, 5, 150, 18\rangle$	2975	1774	1025	1024	1024
$\langle 25, 5, 150, 20\rangle$	3275	1775	1025	1025	1025

Table 1. Number of clauses for different encodings

6 Conclusions

We have defined the minimal support encoding, which is a new encoding from CSP into SAT, and a number of original encodings (`dir`, `supxy`, `supx`, `supl`, `supc`) that map Max-CSP instances into partial Max-SAT instances, and have provided experimental evidence that different Max-SAT encodings for a given optimization problem may produce substantial differences on the performance of a solver. Since our mappings produce one unsatisfied clause for every violated constraints, they can be easily extended to mappings from weighted Max-CSP instances into weighted Max-SAT instances; all the clauses encoding a certain constraint should have as weight the weight associated to that constraint.

To the best of our knowledge, this is the first paper that addresses the question of how to encode Max-CSP into Max-SAT, and analyzes the impact of modelling on the performance of Max-SAT solvers. Future research directions include analyzing the degree of soft local consistency achieved by each encoding, conducting an experimental investigation with benchmarks other than random binary Max-CSP instances, and generalizing our results to n-ary constraints.

References

1. T. Alsinet, F. Manyà, and J. Planes. Improved exact solver for weighted Max-SAT. In *Proceedings of SAT-2005*, pages 371–377. Springer LNCS 3569, 2005.
2. J. Argelich and F. Manyà. Exact Max-SAT solvers for over-constrained problems. *Journal of Heuristics*, 12(4–5):375–392, 2006.

3. J. Argelich and F. Manyà. Partial Max-SAT solvers with clause learning. In *Proceedings of SAT-2007*, pages 28–40. Springer LNCS 4501, 2007.
4. C. Bessière, E. Hebrard, and T. Walsh. Local consistencies in SAT. In *Proceedings of SAT-2003, Santa Margarita Ligure, Italy*, pages 299–314. Springer LNCS 2919, 2003.
5. M. L. Bonet, J. Levy, and F. Manyà. A complete calculus for Max-SAT. In *Proceedings of SAT-2006, Seattle, USA*, pages 240–251. Springer LNCS 4121, 2006.
6. M. L. Bonet, J. Levy, and F. Manyà. Resolution for Max-SAT. *Artificial Intelligence*, 171(8–9):240–251, 2007.
7. N. Een and N. Sorensson. An extensible SAT-solver. In *Proceedings of SAT-2003*, pages 502–518. Springer LNCS 2919, 2003.
8. Z. Fu and S. Malik. On solving the partial MAX-SAT problem. In *Proceedings of SAT-2006*, pages 252–265. Springer LNCS 4121, 2006.
9. M. Gavanelli. The log-support encoding of CSP into SAT. In *Proceedings of CP-2007*, pages 815–822. Springer LNCS 4741, 2007.
10. R. Génisson and P. Jégou. Davis and Putnam were already checking forward. In *Proceedings of the ECAI-1996*, pages 180–184, 1996.
11. I. P. Gent. Arc consistency in SAT. In *Proceedings of ECAI-2002*, pages 121–125, 2002.
12. F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSat: A new weighted Max-SAT solver. In *Proceedings of SAT-2007*, 2007.
13. S. Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, 45:275–286, 1990.
14. J. Larrosa and F. Heras. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In *Proceedings of IJCAI-2005, Edinburgh, Scotland*, pages 193–198. Morgan Kaufmann, 2005.
15. J. Larrosa, F. Heras, and S. de Givry. A logical approach to efficient max-sat solving. *Artificial Intelligence*, 172(2–3):204–233, 2008.
16. C. M. Li, F. Manyà, and J. Planes. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In *Proceedings of CP-2005*, pages 403–414. Springer LNCS 3709, 2005.
17. C. M. Li, F. Manyà, and J. Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In *Proceedings of AAAI-2006*, pages 86–91, 2006.
18. C. M. Li, F. Manyà, and J. Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.
19. J. Marques-Silva and J. Planes. Algorithms for Maximum Satisfiability using Unsatisfiable Cores. In *Proceedings of DATE-2008*, 2008.
20. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *39th Design Automation Conference*, 2001.
21. K. Pipatsrisawat and A. Darwiche. Clone: Solving weighted max-sat in a reduced search space. In *20th Australian Joint Conference on Artificial Intelligence, AI-07*, pages 223–233, 2007.
22. M. Ramírez and H. Geffner. Structural relaxations by variable renaming and their compilation for solving MinCostSAT. In *Proceedings of CP-2007*, pages 605–619. Springer LNCS 4741, 2007.
23. T. Walsh. SAT v CSP. In *Proceedings of CP-2000*, pages 441–456. Springer LNCS 1894, 2000.