Regular Encodings from Max-CSP into Partial Max-SAT*

Josep Argelich
INESC-ID
josep@sat.inesc-id.pt

Alba Cabiscol
DIEI, UdL
alba@diei.udl.cat

Inês Lynce
IST, INESC-ID
ines@sat.inesc-id.pt

Felip Manyà
IIIA, CSIC
felip@iiia.csic.es

Abstract

We define a number of original encodings, called regular encodings*, that map Max-CSP instances into Partial Max-SAT instances. First, we obtain new direct and (minimal) support encodings by modelling the at-least-one and at-most-one conditions using a regular signed encoding. This way, we obtain encodings in which the hard part is more compact. Second, even when we need to introduce auxiliary variables in the regular encodings, we prove that it is sufficient to limit branching to non-auxiliary variables. Third, we report on an experimental investigation which provides evidence that the minimal support encoding is well-suited on more structured, realistic instances (the experiments performed so far were limited to randomly generated binary CSPs), and that the regular encodings defined here have a very competitive performance profile when branching is limited to non-auxiliary variables. We show that regular encodings allow to solve more instances and more efficiently than using the existing encodings from Max-CSP into Partial Max-SAT.*

1 Introduction

Solving combinatorial decision and optimization problems via their reduction to Satisifability Problems (e.g., SAT and Max-SAT) depends on both the solver and the encoding. In this paper, we define new encodings from Max-CSP into Partial Max-SAT, and analyze their impact in the performance of Partial Max-SAT solvers.

In ISMVL-2008 [2], we presented a number of original encodings —that rely on the well-known direct and support encodings from CSP into SAT [10, 13, 18]— that map Max-CSP instances into Partial Max-SAT instances. We showed there that the minimal support encoding (first defined in [2]) was superior to the rest of the encodings, at least on random binary Max-CSP instances. On the other hand, in [3] we showed that the minimal support encoding from CSP into SAT does not maintain arc consistency through unit propagation and, in contrast with what happens with Max-SAT, is not a suitable alternative for SAT solvers.

This paper is a continuation of [2, 3] in which we define new encodings —called *regular encodings— from Max-CSP into Partial Max-SAT*, with the ultimate goal of knowing the impact of the encodings in the performance of solvers, and identifying features of the encodings that could be useful to predict their behavior. First, we obtain new direct and (minimal) support encodings by modelling the at-least-one and at-most-one conditions using a regular signed encoding. This way, we obtain encodings in which the hard part is more compact. Second, even when we need to introduce auxiliary variables in the regular encodings, we prove that it is sufficient to limit branching to non-auxiliary variables. Third, we report on an experimental investigation which provides evidence that the minimal support encoding is well-suited on more structured, realistic instances (the experiments performed so far were limited to randomly generated binary CSPs [2, 3]), and that the regular encodings defined here have a very competitive performance profile when branching is limited to non-auxiliary variables. We show that regular encodings allow to solve more instances and more efficiently than using the existing encodings from Max-CSP into Partial Max-SAT.

The structure of the paper is as follows. Section 2 contains preliminary definitions about Max-SAT and Max-CSP. Section 3 surveys the direct, support and minimal support encodings from Max-CSP into Partial Max-SAT. Section 4 defines the new regular encodings from Max-CSP into Partial Max-SAT, which encode the at-least-one and at-most-one conditions using existing results of regular signed logics. Section 5 reports and analyses the experimental investigation, which is based on realistic instances. Section 6 presents the conclusions and future research directions.

*This research was funded by the Generalitat de Catalunya under grant 2005-SGR-00093, the *Ministerio de Ciencia e Innovación* research projects CONSOLIDER CSD2007-0022, INGENIO 2010, TIN2006-15662-C02-02, TIN2007-68005-C04-04 and Acción Integrada HP2005-0147, FCT research projects BSOLO (PTDC/EIA/76572/2006) and SHIPs (PTDC/EIA/64164/2006), and European project Mancoosi (FP7-ICT-214898).

0195-623X/09 $25.00 © 2009 IEEE
DOI 10.1109/ISMVL.2009.23

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.

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 $\langle v(X_{i_1}), \ldots, v(X_{i_k}) \rangle \in R_i$.

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

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.

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

3 Existing Encodings from Max-CSP into Partial Max-SAT

We introduce a number of encodings such that, given a Max-CSP instance P, they produce a Partial Max-SAT instance ϕ in which 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. These encodings were first presented in [2, 3].

In the encodings below we associate a Boolean variable x_i with each value i that can be assigned to the CSP variable X. Assuming that X has a domain $d(X)$ of size m, the *at-least-one* clause of X is $x_1 \vee \cdots \vee x_m$, and ensures that the CSP variable X is given a value. The *at-most-one* clauses of X are the set of clauses $\{\overline{x}_i \vee \overline{x}_j | i, j \in d(X), i < j\}$, and ensure that the CSP variable X takes no more than one value. The at-least-one and at-most-one clauses together ensure that each CSP variable takes exactly one value of its domain.

3.1 Direct Encoding

Definition 3. 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 $\{X, Y\}$. Recall that a nogood is a forbidden assignment of values to X and Y.

Example 1. The Partial Max-SAT direct encoding for the Max-CSP problem of the CSP instance $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle = \langle \{X, Y\}, \{d(X) = \{1, 2, 3\}, d(Y) = \{1, 2, 3\}\}, \{X \leq Y\} \rangle$ 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	$(\overline{x}_2 \vee \overline{y}_1)$	$(\overline{x}_3 \vee \overline{y}_1)$	$(\overline{x}_3 \vee \overline{y}_2)$

3.2 Support Encoding

In the *support encoding* from CSP into SAT [13, 10, 18], besides the at-least-one and at-most-one clauses, there are

clauses that encode the *support* for a value instead of encoding conflicts. The support for a value j of a CSP variable X across a binary constraint with scope $\{X, Y\}$ is the set of values of Y which allow $X = j$. If $v_1, v_2, \ldots, v_k$ are the supporting values of variable Y for $X = j$, we add the clause $\overline{x}_j \vee y_{v_1} \vee y_{v_2} \vee \cdots \vee y_{v_k}$ (called *support* clause). There is one support clause for each pair of variables X, Y involved in a constraint, and for each value in the domain of X. In the standard support encoding, a clause in each direction is used: one for the pair X, Y and one for Y, X.

In [2], we defined the *minimal support encoding* from CSP into SAT. It 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.

We now introduce the support encodings from Max-CSP into Partial Max-SAT that we defined by adapting the support and minimal support encodings from CSP into SAT:

Definition 4. 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 2. 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	$(\overline{x}_2 \vee y_2 \vee y_3)$	$(\overline{x}_3 \vee y_3)$	

Definition 5. 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 constraint when mapping weighted Max-CSP instances into weighted Max-SAT instances. 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 3. 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	$(\overline{x}_2 \vee y_2 \vee y_3 \vee c_1)$	$(\overline{y}_1 \vee x_1 \vee \overline{c}_1)$	
	$(\overline{x}_3 \vee y_3 \vee c_1)$	$(\overline{y}_2 \vee x_1 \vee x_2 \vee \overline{c}_1)$	

We use three different support encodings in the experimentation: the support encoding (`supxy`), and two variants of the minimal support encoding (`supl` and `supc`). 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 $X = \{X, Y\}, d(X) = d(Y) = \{1, 2, 3, 4\}, C = \{C_{XY}\} = \{\{(1, 2), (1, 3), (1, 4)\}\}$, `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 $\overline{x}_1 \vee y_2 \vee y_3 \vee y_4$.

4 Regular Encodings from Max-CSP into Partial Max-SAT

In this paper we propose novel direct and support encodings from Max-CSP into Partial Max-SAT. These new encodings, called regular encodings, differ in the fact that they encode the at-least-one and at-most-one conditions using an encoding based on the regular signed literals used in many-valued automated theorem proving [1, 6]. To this end, for every CSP variable X, we associate a Boolean variable x_i with each value i that can be assigned to the CSP variable X in such a way that x_i is true if $X = i$. Moreover, we associate a Boolean variable $x_i^{\geq}$ with each value i of the domain of X such that $x_i^{\geq}$ is true if $X \geq i$. Then, the *regular encoding of the at-least-one and at-most-one conditions* for a variable X with $d(X) = \{1, \ldots, n\}$ is formed by the following clauses [1]:

$$
\begin{array}{ll}
x_n^{\geq} \rightarrow x_{n-1}^{\geq} & x_1 \leftrightarrow \overline{x}_2^{\geq} \\
x_{n-1}^{\geq} \rightarrow x_{n-2}^{\geq} & x_2 \leftrightarrow x_2^{\geq} \wedge \overline{x}_3^{\geq} \\
\cdots\cdots\cdots\cdots & \cdots\cdots\cdots\cdots \\
x_3^{\geq} \rightarrow x_2^{\geq} & x_i \leftrightarrow x_i^{\geq} \wedge \overline{x}_{i+1}^{\geq} \\
x_2^{\geq} \rightarrow x_1^{\geq} & \cdots\cdots\cdots\cdots \\
 & x_{n-1} \leftrightarrow x_{n-1}^{\geq} \wedge \overline{x}_n^{\geq} \\
 & x_n \leftrightarrow x_n^{\geq}
\end{array} \quad (1)
$$

The clauses on the left encode the relationship among the different regular literals of a variable while the clauses on the right link the variables of the form x_i with the variables of the form $x_i^{\geq}$.

While the number of clauses derived with the standard encoding of the at-least-one and at-most-one conditions for a CSP variable X with domain $d(X)$ is on $\mathcal{O}(d(X)^2)$, we have that:

Proposition 6. *The number of clauses derived with the regular encoding of the at-least-one and at-most-one conditions for a CSP variable X with domain $d(X)$ is on $\mathcal{O}(d(X))$.*

PROOF: It follows from the fact that all the added clauses have at most three literals, and the number of added clauses is in $\mathcal{O}(d(X))$. ■

Definition 7. The *regular direct, support, and minimal support encodings* are, respectively, the standard direct, support, and minimal support encodings from Max-CSP into Partial Max-SAT but using the regular encoding of the at-least-one and at-most-one conditions.

Example 4. A *regular minimal support encoding* for the Max-CSP problem of the CSP instance from Example 1 is formed by the following clauses:

hard clauses	$[\overline{x}_3^{\geq} \vee x_2^{\geq}]$	$[\overline{x}_2^{\geq} \vee x_1^{\geq}]$	
	$[\overline{x}_1 \vee \overline{x}_2^{\geq}]$	$[x_1 \vee x_2^{\geq}]$	
	$[\overline{x}_2 \vee x_2^{\geq}]$	$[\overline{x}_2 \vee \overline{x}_3^{\geq}]$	$[x_2 \vee \overline{x}_2^{\geq} \vee x_3^{\geq}]$
	$[\overline{x}_3 \vee x_3^{\geq}]$	$[x_3 \vee \overline{x}_3^{\geq}]$	
	$[\overline{y}_3^{\geq} \vee y_2^{\geq}]$	$[\overline{y}_2^{\geq} \vee y_1^{\geq}]$	
	$[\overline{y}_1 \vee \overline{y}_2^{\geq}]$	$[y_1 \vee y_2^{\geq}]$	
	$[\overline{y}_2 \vee y_2^{\geq}]$	$[\overline{y}_2 \vee \overline{y}_3^{\geq}]$	$[y_2 \vee \overline{y}_2^{\geq} \vee y_3^{\geq}]$
	$[\overline{y}_3 \vee y_3^{\geq}]$	$[y_3 \vee \overline{y}_3^{\geq}]$	
support	$(\overline{x}_2 \vee y_2 \vee y_3)$	$(\overline{x}_3 \vee y_3)$	

We get the *regular support encoding* if we replace the previous support clauses with:

support	$(\overline{x}_2 \vee y_2 \vee y_3 \vee c_1)$	$(\overline{y}_1 \vee x_1 \vee \overline{c}_1)$
	$(\overline{x}_3 \vee y_3 \vee c_1)$	$(\overline{y}_2 \vee x_1 \vee x_2 \vee \overline{c}_1)$

Finally, we get the *regular direct encoding* if we replace the support clauses with the following conflict clauses:

$$\text{conflict} \quad (\overline{x}_2 \vee \overline{y}_1) \quad (\overline{x}_3 \vee \overline{y}_1) \quad (\overline{x}_3 \vee \overline{y}_2)$$

One feature of the regular encoding —whose impact on performance is discussed in the section of the experimental investigation— is that it is not necessary to perform branching on the auxiliary variables:

Proposition 8. *When solving a Max-CSP instance with a regular encoding and a Davis-Putnam style branch and bound solver, if branching is performed only on non-auxiliary variables, then the solver finds an optimal solution.*

PROOF: It follows from the analysis of (1). If we branch on x_i, all the regular literals are instantiated in such a way that $x_j^{\geq}$ becomes true if $j \leq i$ and becomes false if $j > i$. On the other hand, if we had branched on $\overline{x}_i$ for all the elements of the domain of X except for one value, say j, then all the literals of the form $x_j^{\geq}$ would have been instantiated and x_j would become true as a result of the sentence $x_j \leftrightarrow x_j^{\geq} \wedge \overline{x}_{j+1}^{\geq}$. Therefore, it is not necessary to branch on auxiliary variables because they are implied when branching is performed on non-auxiliary variables. ■

Branching limited to non-auxiliary variables has shown to be effective in SAT-encoded scheduling and planning problems [7, 11, 16].

5 Experimental Results

We conducted an empirical evaluation with the following goals: (i) analyse the performance profile of the standard direct and support encodings on more structured, realistic instances because the experiments in [2, 3] were limited to random binary Max-CSP instances; (ii) analyse the performance profile of the new encodings defined here; and (iii) study the impact of performing only branching on non-auxiliary variables. We used two of the fastest Partial Max-SAT solvers in the Max-SAT evaluations [5] held so far: MiniMaxSat [12], and WMaxSatz [4] (the Weighted Partial Max-SAT version of MaxSatz [14, 15]).

The evaluation was performed on a cluster with 160 2 GHz AMD Opteron 248 Processors with 1 GB of memory. We used a cutoff time of 30 minutes for each instance.

As benchmarks we encoded, into Partial Max-SAT, instances that have been used in the constraint programming community working on soft constraints [8, 9]. Such instances include instances of clique trees and of the warehouse location problem.

Table 1 and Table 2 present the experimental results which show that the encodings we defined in [2, 3] are well-suited on more realistic, structured instances. We denote the number of instances by #. In Table 1, which contains the results of solving our testbed with MiniMaxSat, we observe that the two variants of the minimal support encoding

Kbtree (t)	#	supc	supl	dir	supxy
10	50	**0.01(50)**	**0.01(50)**	**0.01(50)**	33.10(49)
20	50	0.31(50)	0.28(50)	**0.22(50)**	409.40(39)
30	50	**0.73(50)**	0.77(50)	0.90(50)	-
40	50	**4.98(50)**	5.17(50)	12.93(50)	-
50	50	**9.22(50)**	13.36(50)	37.44(50)	-
60	50	**13.59(50)**	41.71(50)	106.97(50)	-
70	50	**22.79(50)**	77.25(50)	264.81(49)	-
80	50	**20.47(50)**	49.68(50)	444.33(48)	-
90	50	**18.19(50)**	40.44(50)	650.27(40)	-
Solved	450	**450**	**450**	437	88
Warehouse l.	#	supc	supl	dir	supxy
cap	37	**720.11(3)**	881.09(2)	1028.03(2)	-
warehouse	2	0.64(2)	**0.63(2)**	0.64(2)	0.42(1)
Solved	39	**5**	4	4	1

Table 1. Clique trees with different constraint tightness (Kbtree 10–90) and warehouse location instances (Warehouse l.) solved with MiniMaxsat. Mean time in seconds.

(`supc` and `supl`) solve all the clique tree instances (450 instances) while the support encoding `supxy` solves only 88 instances. For the warehouse instances, we observe that `supc` solves 5 instances whereas `supxy` solves only 1 instance. Comparing `supc` and `supl`, we have that `supc` is superior. The direct encoding is not as good as the minimal support encodings. We put "-" in a cell when no instance can be solved within the cutoff time.

In Table 2, which contains the results of solving our testbed with WMaxSatz, we observe that the two variants of the minimal support encoding (`supc` and `supl`) solve more clique tree instances (174 and 159 instances, respectively) than the support encoding `supxy` (50 instances). For the warehouse instances, `supxy` needs more time than the rest of the encodings. Comparing `supc` and `supl`, we have that `supc` is superior. The direct encoding is not as good as the minimal support encodings. We do not include instances for constraint tightness of 60–90 because there is no encoding able to solve these instances within the cutoff time.

Table 3 presents the experimental results which show that performing branching only on non-auxiliary variables leads to substantial performance improvements. In this case, we show only results with WMaxSatz because the source code of MiniMaxSat is not publicly available and, therefore, we could not program the new branching scheme. For the clique tree instances, the gains of the new branching scheme are clear: the regular direct encoding with standard branching (`r-dir`) solves 98 instances while with branching only on non-auxiliary variables (`nb-r-dir`) solves 325 instances; the regular version of the minimal encoding `supc` with standard branching (`r-supc`) solves 138 instances while with branching only on non-auxiliary variables (`nb-r-supc`) solves 320 instances; the regular version of the minimal encoding `supl` with standard branching (`r-supl`) solves 137 instances while with branching only on non-auxiliary variables (`nb-r-supc`) solves 342 instances. Worst results, which are not shown for lack of space, are obtained with the regular version of the support encoding : 47 solved instances with standard branching, and 103 solved instances with branching only on non-auxiliary variables. For the warehouse instances, the new branching scheme reduces the time needed to solve one instance.

To evaluate the advantages of regular encodings over standard encodings we should compare Table 2 (standard encodings) and Table 3 (regular encodings). We observe, for example, that the minimal encoding `supl` solves 159 clique trees while the regular minimal encoding `nb-r-supl` solves 342 instances, and that the direct encoding `dir` solves 150 clique trees while the regular direct encoding `nb-r-supc` solves 325 instances. Overall, the best results are obtained with the regular encodings branching only on non-auxiliary variables, which have been first introduced in this paper.

6 Conclusions

We have defined new encodings from Max-CSP into Partial Max-SAT, called regular encodings, and provided experimental evidence that, besides being more compact, they outperform, on the tested instances, existing encodings when branching is limited to non-auxiliary variables.

Kbtree (t)	#	supc	supl	dir	supxy
10	50	0.02(50)	0.02(50)	**0.01(50)**	23.16(50)
20	50	53.27(50)	35.62(50)	**2.21(50)**	-
30	50	415.82(50)	503.19(50)	**329.94(50)**	-
40	50	**1166.35(20)**	1504.73(9)	-	-
50	50	**1253.20(4)**	-	-	-
Solved	250	**174**	159	150	50
Warehouse l.	#	supc	supl	dir	supxy
warehouse	2	**0.06(1)**	**0.06(1)**	**0.06(1)**	0.23(1)
Solved	2	**1**	**1**	**1**	**1**

Table 2. Clique trees with different constraint tightness (Kbtree 10–50) and warehouse location instances (Warehouse l.) solved with W-MaxSatz. Mean time in seconds.

		r-supc		r-supl		r-dir	
Kbtree (t)	#	nb	b	nb	b	nb	b
10	50	**0.03(50)**	0.36(50)	**0.03(50)**	0.07(50)	**0.03(50)**	1.58(50)
20	50	0.82(50)	70.91(50)	0.89(50)	57.12(50)	**0.30(50)**	375.07(48)
30	50	4.75(50)	627.38(36)	4.39(50)	664.75(35)	**4.03(50)**	-
40	50	**30.74(50)**	1341.48(2)	36.85(50)	1714.93(2)	43.43(50)	-
50	50	**56.05(50)**	-	147.93(50)	-	176.35(50)	-
60	50	**390.55(50)**	-	334.63(49)	-	656.88(48)	-
70	50	889.75(20)	-	**618.76(43)**	-	898.49(25)	-
80	50	-	-	-	-	**1390.29(2)**	-
Solved	400	320	138	**342**	137	325	98
		r-supc		r-supl		r-dir	
Warehouse l.	#	nb	b	nb	b	nb	b
warehouse	2	**0.12(1)**	2.43(1)	**0.12(1)**	2.45(1)	**0.12(1)**	2.43(1)
Solved	2	**1**	**1**	**1**	**1**	**1**	**1**

Table 3. Regular encodings with W-MaxSatz with standard branching (b) and branching only on non-auxiliary variables (nb). Mean time in seconds.

On the other hand, we have also provided experimental evidence that the encodings defined in [2, 3] are well-suited on more realistic, structured Max-CSP instances.

It is worth to notice that the results of this paper, which are based on results from the research community working on automated theorem proving on many-valued logics, bridge the gap between Max-CSP and Partial Max-SAT. Moreover, they also show that it makes sense to pursue the investigation of encodings from Max-CSP into Partial Max-SAT that we started in [2, 3].

Future research directions include analyzing the so-called logarithmic encodings and generalizing our results on support encodings to n-ary constraints and many-valued Max-SAT. We are also interested in analyzing the behavior of the sequential encoding of cardinality constraints[17] for the at-most-one condition.

References

[1] C. Ansótegui and F. Manyà. Mapping problems with finite-domain variables into problems with boolean variables. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (Revised Selected Papers), SAT-2004, Vancouver, Canada*, pages 1–15. Springer LNCS 3542, 2004.

[2] J. Argelich, A. Cabiscol, I. Lynce, and F. Manyà. Encoding Max-CSP into partial Max-SAT. In *Proceedings, 38th International Symposium on Multiple-Valued Logics (ISMVL), Texas/TX, USA*. IEEE CS Press, 2008.

[3] J. Argelich, A. Cabiscol, I. Lynce, and F. Manyà. Modelling Max-CSP as partial Max-SAT. In *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing, SAT-2008, Guangzhou, China*, pages 1–14. Springer LNCS 4996, 2008.

[4] J. Argelich, C. M. Li, and F. Manyà. An improved exact solver for partial Max-SAT. In *Proceedings of the International Conference on Nonconvex Programming: Local and Global Approaches, NCP-2007, Rouen, France*, pages 230–231, 2007.

[5] J. Argelich, C. M. Li, F. Manyà, and J. Planes. The first and second Max-SAT evaluations. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:251–278, 2008.

[6] R. Béjar, R. Hähnle, and F. Manyà. A modular reduction of regular logic to classical logic. In *Proceedings, 31st International Symposium on Multiple-Valued Logics (ISMVL), Warsaw, Poland*, pages 221–226. IEEE CS Press, Los Alamitos, 2001.

[7] J. M. Crawford and A. B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the 12th National Conference on Artificial Intelligence, AAAI'94, Seattle/WA, USA*, pages 1092–1097. AAAI Press, 1994.

[8] S. de Givry, F. Heras, J. Larrosa, and M. Zytnicki. Existencial arc consistency: Getting closer to full arc consistency in weighted CSPs. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-2005, Edinburgh, Scotland*, pages 84–89. Morgan Kaufmann, 2005.

[9] S. de Givry, T. Schiex, and G. Verfaillie. Exploiting tree decomposition and soft local consistency in weighted CSP. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-2006, Boston/MA, USA*, 2006.

[10] I. P. Gent. Arc consistency in SAT. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI), Lyon, France*, pages 121–125, 2002.

[11] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the rest will follow: Exploiting determinism in planning as satisability. In *Proceedings of the 15th National Conference on Artificial Intelligence, AAAI'98, Madison/WI, USA*, pages 948–953. AAAI Press, 1998.

[12] F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSat: A new weighted Max-SAT solver. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing, SAT-2007, Lisbon, Portugal*, pages 41–55, 2007.

[13] S. Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, 45:275–286, 1990.

[14] C. M. Li, F. Manyà, and J. Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAAI-2006, Boston/MA, USA*, pages 86–91, 2006.

[15] C. M. Li, F. Manyà, and J. Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.

[16] I. Lynce and J. P. Marques-Silva. Towards robust CNF encodings of cardinality constraints. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming, CP-2007, Providence/RI, USA*, pages 483–497. Springer LNCS 4741, 2007.

[17] C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, CP-2005, Sitges, Spain*, pages 827–831. Springer LNCS 3709, 2005.

[18] T. Walsh. SAT v CSP. In *Proceedings of the 6th International Conference on Principles of Constraint Programming, CP-2000, Singapore*, pages 441–456. Springer LNCS 1894, 2000.