New Encodings from Max-CSP into Partial Max-SAT

J. Argelich, A. Cabiscol, I. Lynce and F. Manyà

December 2008
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 that 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 quickly than using the existing encodings from Max-CSP into Partial Max-SAT.

1 Introduction

Solving combinatorial decision and optimization problems via their reduction to Satisfiability Problems (e.g., SAT and MaxSAT) depends on both the solver and the encoding. 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 [8, 11, 13]— 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 encodings, at least on random binary CSPs.

This paper is a continuation of [2] 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 that 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]), 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 quickly 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. 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.

The Max-SAT problem for a CNF formula \( \phi \) 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 \( \langle \{X_{i_1}, \ldots, X_{i_k}\}, R_i \rangle \in \mathcal{C} \) 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 CSP 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 CSP instance \( P \), they produce a Partial Max-SAT instance \( \phi \) 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 \( \phi \) that are falsified by a Boolean assignment. These encodings were first presented in [2].

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 of size \( m \), the at-least-one clause of \( X \) is \( x_1 \lor \cdots \lor x_m \), and ensures that the CSP variable \( X \) is given a value. The at-most-one clauses of \( X \) are the set of clauses \( \{x_i \lor x_j| i, j \in d(X), i < j\} \), 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 \( \langle X, D, C \rangle \) 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 \( X \), and contains a soft clause \( x_i \lor y_j \) for every nogood \( (X = i, Y = j) \) of every constraint of \( C \) 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 \( \langle X, D, C \rangle = \langle \{X, Y\}, \{d(X) = \{1, 2, 3\}, d(Y) = \{1, 2, 3\}\}, \{X \leq Y\} \rangle \) is as follows:

\[
\begin{align*}
\text{at-least-one} & : [x_1 \lor x_2 \lor x_3] & [y_1 \lor y_2 \lor y_3] \\
\text{at-most-one} & : [\overline{x}_1 \lor \overline{x}_2] & [\overline{x}_1 \lor \overline{x}_3] & [\overline{x}_2 \lor \overline{x}_3] \\
\text{conflict} & : [\overline{y}_1 \lor \overline{y}_2] & [\overline{y}_1 \lor \overline{y}_3] & [\overline{y}_2 \lor \overline{y}_3]
\end{align*}
\]

3.2 Support Encoding

In the support encoding from CSP into SAT [11, 8, 13], 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 \( x_j \lor y_{v_1} \lor y_{v_2} \lor \cdots \lor y_{v_k} \) (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 next introduce the support encodings for 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 \((X, D, C)\) 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 \(X\), 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** \([x_1 \lor x_2 \lor x_3] [y_1 \lor y_2 \lor y_3]\)
- **at-most-one** \([\overline{x}_1 \lor \overline{x}_2] [\overline{y}_1 \lor \overline{y}_3] [\overline{y}_2 \lor \overline{y}_3]\)
- **support** \((\overline{x}_2 \lor y_2 \lor y_3) (y_1 \lor x_1 \lor \overline{c}_1) (x_3 \lor y_1 \lor c_1) (\overline{y}_2 \lor x_1 \lor x_2 \lor \overline{c}_1)\)

**Definition 5.** The support encoding of a Max-CSP instance \((X, D, C)\) 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 \(X\), and contains, for every constraint \(C_k \in C\) with scope \(\{X, Y\}\), a soft clause of the form \(S_{X=j} = c_k\) 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 \(S_{Y=m} = \overline{c}_k\) 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** \([x_1 \lor x_2 \lor x_3] [y_1 \lor y_2 \lor y_3]\)
- **at-most-one** \([\overline{x}_1 \lor \overline{x}_2] [\overline{x}_1 \lor \overline{x}_3] [\overline{x}_2 \lor \overline{x}_3]\)
- **support** \((\overline{x}_2 \lor y_2 \lor y_3 \lor c_1) (\overline{y}_1 \lor x_1 \lor \overline{c}_1) (x_3 \lor y_1 \lor c_1) (\overline{y}_2 \lor x_1 \lor x_2 \lor \overline{c}_1)\)

We use three different support encodings in the experimentation: the support encoding \((sup_{xy})\), and two variants of the minimal support encoding \((sup_{l} and sup_{c})\). The encoding \(sup_{l}\) 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 \texttt{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 \( X = \{X, Y\}, d(X) = d(Y) = \{1, 2, 3, 4\}, C = \{C_{XY}\} = \{(1, 2), (1, 3), (1, 4)\}\), \texttt{supc} prefers three binary support clauses \( x_1 \lor \overline{y}_2, x_1 \lor \overline{y}_3, x_1 \lor \overline{y}_4 \) rather than the quaternary support clause \( \overline{x}_1 \lor y_2 \lor y_3 \lor y_4 \), while \texttt{supl} prefers \( \overline{x}_1 \lor y_2 \lor y_3 \lor y_4 \).

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 [4, 1]. 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]:

\[
\begin{align*}
    x_n^{\geq} &\rightarrow x_{n-1}^{\geq} \\
    x_n^{\geq} &\rightarrow x_{n-1}^{\geq} \\
    \cdots \cdots \cdots &\cdots \cdots \cdots \\
    x_3^{\geq} &\rightarrow x_2^{\geq} \\
    x_2^{\geq} &\rightarrow x_1^{\geq} \\
    x_2 \leftrightarrow x_1^{\geq} \land \overline{x_3^{\geq}} \\
    &\vdots \\
    x_{n-1}^{\geq} &\leftrightarrow x_{n-2}^{\geq} \land \overline{x_{n-1}^{\geq}} \\
    x_n &\leftrightarrow x_{n-1}^{\geq} \land \overline{x_n^{\geq}} \\
\end{align*}
\]

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 \( x_i^\geq \) with the variables of the form \( x_i \).

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 \( \mathcal{O}(d(X)) \). \( \square \)

**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:

\[
\begin{align*}
&[x_2 \geq 3] \lor [x_1 \geq 3] \\
&[x_2 \geq 2] \lor [x_1 \geq 2] \\
&[x_2 \geq 2] \lor [x_1 \geq 1] \\
&[x_3 \geq 2] \lor [x_1 \geq 2] \lor [x_2 \geq 3] \\
&[y_2 \geq 2] \lor [y_1 \geq 2] \\
&[y_2 \geq 1] \lor [y_1 \geq 1] \\
&[y_2 \geq 1] \lor [y_1 \geq 2] \\
&[y_2 \geq 1] \lor [y_1 \geq 2] \lor [y_2 \geq 3] \\
&[y_3 \geq 2] \lor [y_1 \geq 2] \lor [y_2 \geq 3] \\
&[y_3 \geq 2] \lor [y_1 \geq 2] \lor [y_2 \geq 3] \\
&[y_3 \geq 2] \lor [y_1 \geq 2] \lor [y_2 \geq 3] \\

\end{align*}
\]

support:

\[
(x_2 \lor y_2 \lor y_3) \land (x_3 \lor y_3)
\]

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

\[
\begin{align*}
&[x_2 \lor y_2 \lor y_3 \lor c_1] \\
&[y_1 \lor x_1 \lor c_1] \\
&[y_3 \lor y_3 \lor y_3] \\
&[y_3 \lor y_3 \lor y_3]
\end{align*}
\]

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

\[
\begin{align*}
&[x_2 \lor y_1] \\
&[x_3 \lor y_1] \\
&[x_3 \lor y_1] \\
&[x_3 \lor y_1]
\end{align*}
\]

One feature of the regular encoding—which 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 \(x_j \geq i\) 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 \(x_j \geq i\) would have been instantiated and \(x_j\) would become true as a result of the sentence \(x_j \leftrightarrow x_j \geq i \land \overline{x}_j \geq j+1\). 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 [5, 9].

**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]
<table>
<thead>
<tr>
<th>Kbtree (t)</th>
<th>#</th>
<th>supc</th>
<th>supl</th>
<th>dir</th>
<th>supxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>50</td>
<td><strong>0.01(50)</strong></td>
<td><strong>0.01(50)</strong></td>
<td><strong>0.01(50)</strong></td>
<td>33.10(49)</td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>0.31(50)</td>
<td>0.28(50)</td>
<td><strong>0.22(50)</strong></td>
<td>409.40(39)</td>
</tr>
<tr>
<td>30</td>
<td>50</td>
<td><strong>0.73(50)</strong></td>
<td>0.77(50)</td>
<td>0.90(50)</td>
<td>-</td>
</tr>
<tr>
<td>40</td>
<td>50</td>
<td><strong>4.98(50)</strong></td>
<td>5.17(50)</td>
<td>12.93(50)</td>
<td>-</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td><strong>9.22(50)</strong></td>
<td>13.36(50)</td>
<td>37.44(50)</td>
<td>-</td>
</tr>
<tr>
<td>60</td>
<td>50</td>
<td><strong>13.59(50)</strong></td>
<td>41.71(50)</td>
<td>106.97(50)</td>
<td>-</td>
</tr>
<tr>
<td>70</td>
<td>50</td>
<td><strong>22.79(50)</strong></td>
<td>77.25(50)</td>
<td>264.81(49)</td>
<td>-</td>
</tr>
<tr>
<td>80</td>
<td>50</td>
<td><strong>20.47(50)</strong></td>
<td>49.68(50)</td>
<td>444.33(48)</td>
<td>-</td>
</tr>
<tr>
<td>90</td>
<td>50</td>
<td><strong>18.19(50)</strong></td>
<td>40.44(50)</td>
<td>650.27(40)</td>
<td>-</td>
</tr>
<tr>
<td>Solved</td>
<td>450</td>
<td>450</td>
<td>450</td>
<td>437</td>
<td>88</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Warehouse l.</th>
<th>#</th>
<th>supc</th>
<th>supl</th>
<th>dir</th>
<th>supxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>cap</td>
<td>37</td>
<td><strong>720.11(3)</strong></td>
<td>881.09(2)</td>
<td>1028.03(2)</td>
<td>-</td>
</tr>
<tr>
<td>warehouse</td>
<td>2</td>
<td>0.64(2)</td>
<td><strong>0.63(2)</strong></td>
<td>0.64(2)</td>
<td>0.42(1)</td>
</tr>
<tr>
<td>Solved</td>
<td>39</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Kbtree (t)</th>
<th>#</th>
<th>supc</th>
<th>supl</th>
<th>dir</th>
<th>supxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>50</td>
<td>0.02(50)</td>
<td>0.02(50)</td>
<td>0.01(50)</td>
<td>23.16(50)</td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>53.27(50)</td>
<td>35.62(50)</td>
<td>2.21(50)</td>
<td>-</td>
</tr>
<tr>
<td>30</td>
<td>50</td>
<td>415.82(50)</td>
<td>503.19(50)</td>
<td>329.94(50)</td>
<td>-</td>
</tr>
<tr>
<td>40</td>
<td>50</td>
<td>1166.35(20)</td>
<td>1504.73(9)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>1253.20(4)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Solved</td>
<td>250</td>
<td>174</td>
<td>159</td>
<td>150</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Warehouse l.</th>
<th>#</th>
<th>supc</th>
<th>supl</th>
<th>dir</th>
<th>supxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>warehouse</td>
<td>2</td>
<td>0.06(1)</td>
<td>0.06(1)</td>
<td>0.06(1)</td>
<td>0.23(1)</td>
</tr>
<tr>
<td>Solved</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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 state-of-the-art Partial Max-SAT solvers: MiniMaxSat [10], and WMaxSatz [3] (the Weighted Partial Max-SAT version of MaxSat [12]).

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 [6, 7]. Such instances include instances of clique trees and of the warehouse location problem.

Table 1 and Table 2 show the experimental results that show that the encodings we defined in [2] 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 (supc and supl) solve all the clique tree instances (450 instances) while the support encoding supxy only solves 88 instances. For the warehouse instances, we observe that supc solves 5 instances but supxy only solves 1. Comparing supc and supl, we have that supc is superior. The direct encoding is not so 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 so 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>
<thead>
<tr>
<th>Kbtree (t)</th>
<th>r-supc</th>
<th>r-supl</th>
<th>r-dir</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#</td>
<td>nb</td>
<td>b</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>0.03(50)</td>
<td>0.36(50)</td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>0.82(50)</td>
<td>70.91(50)</td>
</tr>
<tr>
<td>30</td>
<td>50</td>
<td>4.75(50)</td>
<td>627.38(36)</td>
</tr>
<tr>
<td>40</td>
<td>50</td>
<td><strong>30.74(50)</strong></td>
<td>1341.48(2)</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td><strong>56.05(50)</strong></td>
<td>-</td>
</tr>
<tr>
<td>60</td>
<td>50</td>
<td><strong>390.55(50)</strong></td>
<td>-</td>
</tr>
<tr>
<td>70</td>
<td>50</td>
<td>889.75(20)</td>
<td>-</td>
</tr>
<tr>
<td>80</td>
<td>50</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Solved</td>
<td>400</td>
<td>320</td>
<td>138</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Warehouse</th>
<th>r-supc</th>
<th>r-supl</th>
<th>r-dir</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#</td>
<td>nb</td>
<td>b</td>
</tr>
<tr>
<td>warehouse</td>
<td>2</td>
<td>0.12(1)</td>
<td>2.43(1)</td>
</tr>
<tr>
<td>Solved</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3. Regular encodings with W-MaxSat with standard branching (b) and branching only on non-auxiliary variables (nb). Mean time in seconds.
Table 3 provides the experimental results that show that performing branching only on non-auxiliary variables leads to substantial performance improvements. In this case, we just show results with WMaxSat because the source code of MiniMaxSat is not publicly available and, therefore, we could not program the new branching. For the clique tree instances, the gains of the new branching 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. For the warehouse instances, the new branching reduces the time needed to solve an 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, or 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 the encodings defined so far when branching is limited to non-auxiliary variables. On the other hand, we have also provided experimental evidence that the encodings defined in [2] 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].

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.

Acknowledgements

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, and by European project Mancoosi (FP7-ICT-214898) and by FCT project Bsolo (PTDC/EIA/76572/2006).
References


