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

Josep Argelich[1], Alba Cabiscol[2], Inês Lynce[3], and Felip Manyà[4]

[1] INESC-ID, Lisbon, Portugal
[2] Computer Science Department, Universitat de Lleida, Spain
[3] IST/INESC-ID, and Technical University of Lisbon, Portugal
[4] Artificial Intelligence Research Institute (IIIA, CSIC), Spain

Abstract. We define new encodings from Max-CSP into Partial Max-SAT which are obtained by modelling the at-most-one condition with the sequential SAT encoding of the cardinality constraint $\leq 1(x_1, \ldots, x_n)$. They have fewer clauses than the existing encodings, and the experimental results indicate that they have a better performance profile.

1 Introduction

We describe, following our previous results in [2–4], novel encodings from Max-CSP into Partial Max-SAT. In [2, 3], we defined a new encoding from CSP into SAT, called *minimal support encoding*, and defined the extensions from Max-CSP into Partial Max-SAT of the *direct* and *support* encodings from CSP into SAT, as well as the extension of the minimal support encoding. The experimental results for Partial Max-SAT provide evidence that, in general, the minimal support encoding outperforms the other encodings on both pure random [2, 3] and more structured, realistic instances [4]. In the sequel, when we say direct, support and minimal support encodings we refer to the corresponding encodings from Max-CSP into Partial Max-SAT. We also refer to them as the *standard encodings*.

Recently [4], we have defined new variants of the standard encodings, called *regular direct*, *regular support* and *regular minimal support* encodings. They are obtained by modelling the at-least-one (ALO) and at-most-one (AMO) conditions of the corresponding standard encodings using a regular signed encoding [1]. This way, we get encodings with a more compact set of hard clauses, but we need to introduce auxiliary variables. Fortunately, it is sufficient to limit branching to non-auxiliary variables [4]. From a practical point of view, the regular encodings usually outperform the corresponding standard encodings.

In this paper we define new encodings —*sequential direct*, *sequential support* and *sequential minimal support* —, which are obtained by modelling the ALO

* Research funded by European project Mancoosi (FP7-ICT-214898), FCT projects Bsolo (PTDC/EIA/76572/2006) and SHIPs (PTDC/EIA/64164/2006), and the *Ministerio de Ciencia e Innovación* projects CONSOLIDER CSD2007-0022, INGENIO 2010, TIN2006-15662-C02-02, and TIN2007-68005-C04-04.

condition as in the standard encoding, and the AMO condition with the sequential SAT encoding of the cardinality constraint $\leq 1(x_1, \ldots, x_n)$ [6]. They have fewer clauses than the existing encodings, and the experimental results indicate that they have a better performance profile. In our experiments we solve both pure random and more structured, realistic instances. We refer to [2–4] for basic definitions of Max-SAT and Max-CSP.

2 Encodings from Max-CSP into Partial Max-SAT

2.1 Standard Encodings

We associate a Boolean variable x_i with each value i of the CSP variable X. If X has a domain $d(X)$ of size m, the *ALO* clause of X is $x_1 \vee \cdots \vee x_m$, and ensures that X is given a value. The *AMO* clauses are the set of clauses $\{\overline{x}_i \vee \overline{x}_j | i, j \in d(X), i < j\}$, and ensure that X takes no more than one value.

Definition 1. *The* direct encoding *(*`dir`*) 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 above ALO and AMO clauses for every CSP variable in $\mathcal{X}$, and 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 $\{X, Y\}$.*

In the *support encoding* from CSP into SAT, besides the ALO and AMO 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 [7].

In [2], we defined the *minimal support encoding*: 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.

Definition 2. *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 ALO and AMO clauses for every CSP variable in $\mathcal{X}$, and as soft clauses the support clauses of the minimal support encoding from CSP into SAT.*

The support encoding *is the Partial Max-SAT instance that contains as hard clauses the corresponding ALO and AMO 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.*

Example 1. The direct encoding for the Max-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:

ALO	$[x_1 \vee x_2 \vee x_3]$	$[y_1 \vee y_2 \vee y_3]$	
AMO	$[\overline{x}_1 \vee \overline{x}_2]$ $[\overline{x}_1 \vee \overline{x}_3]$ $[\overline{x}_2 \vee \overline{x}_3]$	$[\overline{y}_1 \vee \overline{y}_2]$ $[\overline{y}_1 \vee \overline{y}_3]$ $[\overline{y}_2 \vee \overline{y}_3]$	
conflict clauses	$(\overline{x}_2 \vee \overline{y}_1)$	$(\overline{x}_3 \vee \overline{y}_1)$	$(\overline{x}_3 \vee \overline{y}_2)$

We get the minimal support encoding if we replace the conflict clauses with $(\overline{x}_2 \vee y_2 \vee y_3), (\overline{x}_3 \vee y_3)$, and get the support encoding if we replace the conflict clauses with $(\overline{x}_2 \vee y_2 \vee y_3 \vee c_1), (\overline{y}_1 \vee x_1 \vee \overline{c}_1), (\overline{x}_3 \vee y_3 \vee c_1), (\overline{y}_2 \vee x_1 \vee x_2 \vee \overline{c}_1)$.

In the experiments we used the support encoding (`supxy`), and two variants of the minimal support encoding (`supl` and `supc`): `supl` is the encoding containing, for each constraint, the support clauses for the variable that produces a smaller total number of literals; and `supc` is the 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.

2.2 Regular Encodings

The regular encodings differ in the fact that they encode the ALO and AMO conditions using a regular signed encoding [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 ALO and AMO 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 $x_i^{\geq}$.

Definition 3. *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 ALO and AMO conditions.*

In [4] we proved that when solving a Max-CSP instance with a regular encoding and a Davis-Logemann-Loveland (DLL) style branch and bound solver, if branching is performed only on non-auxiliary variables, then the solver finds an optimal solution. We assume this kind of branching in the rest of the paper.

3 Sequential Encodings

Our new encodings model the ALO condition as in the standard encoding, and the AMO condition using the following SAT encoding, based on sequential counters, of the cardinality constraint $\leq 1(x_1, \ldots, x_n)$ [6]:

$$(\overline{x}_1 \vee s_1) \wedge (\overline{x}_n \vee \overline{s}_{n-1}) \bigwedge_{1<i<n} ((\overline{x}_i \vee s_i) \wedge (\overline{s}_{i-1} \vee s_i) \wedge (\overline{x}_i \vee \overline{s}_{i-1})),$$

where s_i, $1 \leq i \leq n-1$, are auxiliary variables. We refer to such an encoding as the sequential encoding of the AMO condition.

Definition 4. *The* sequential 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 sequential encoding of the AMO condition.*

Example 2. A *sequential minimal support encoding* for the Max-CSP problem of the CSP instance from Example 1 is formed by the following clauses:

hard clauses	$[x_1 \vee x_2 \vee x_3]$	$[y_1 \vee y_2 \vee y_3]$			
	$[\overline{x}_1 \vee s_1^x]$	$[\overline{x}_3 \vee \overline{s}_2^x]$	$[\overline{x}_2 \vee s_2^x]$	$[\overline{s}_1^x \vee s_2^x]$	$[\overline{x}_2 \vee \overline{s}_1^x]$
	$[\overline{y}_1 \vee s_1^y]$	$[\overline{y}_3 \vee \overline{s}_2^y]$	$[\overline{y}_2 \vee s_2^y]$	$[\overline{s}_1^y \vee s_2^y]$	$[\overline{y}_2 \vee \overline{s}_1^y]$
support clauses	$(\overline{x}_2 \vee y_2 \vee y_3)$	$(\overline{x}_3 \vee y_3)$			

We get the *sequential support encoding* if we replace the previous support clauses with $(\overline{x}_2 \vee y_2 \vee y_3 \vee c_1), (\overline{y}_1 \vee x_1 \vee \overline{c}_1), (\overline{x}_3 \vee y_3 \vee c_1), (\overline{y}_2 \vee x_1 \vee x_2 \vee \overline{c}_1)$. Finally, we get the *sequential direct encoding* if we replace the previous support clauses with $(\overline{x}_2 \vee \overline{y}_1), (\overline{x}_3 \vee \overline{y}_1), (\overline{x}_3 \vee \overline{y}_2)$.

In the sequential encodings, the number of clauses for modelling the ALO and AMO conditions for a CSP variable X with domain $d(X)$ is on $\mathcal{O}(d(X))$. Observe that, for large domains, there are fewer clauses in the sequential encodings than in the regular and standard encodings.

Proposition 1. *When solving a Max-CSP instance with a sequential encoding and a DLL style branch and bound solver, if branching is performed only on non-auxiliary variables, then the solver finds an optimal solution.*

4 Experimental Results

We conducted experiments on a cluster with 2 GHz AMD Opteron 248 Processors, 1 GB of memory. The benchmarks are random binary Max-CSP instances as the ones solved in [3], as well as the instances of clique trees with different constraint tightness (Kbtree 10–90) and warehouse location solved in [4], which are more structured and realistic. We used the solver WMaxSatz [5] because its code is available, and we had to modify it for implementing a branching scheme

		s-supl		s-dir		s-supc		s-supxy	
Kbtree (t)	#	nb	b	nb	b	nb	b	nb	b
10	50	0.05(50)	0.10(50)	**0.03(50)**	15.01(50)	0.05(50)	1.24(50)	34.27(50)	89.18(44)
20	50	0.78(50)	46.02(50)	**0.36(50)**	345.30(49)	0.70(50)	57.07(50)	682.66(36)	0.00(0)
30	50	3.97(50)	559.65(38)	**2.92(50)**	1440.02(2)	3.78(50)	562.56(40)	0.00(0)	0.00(0)
40	50	**20.19(50)**	1175.17(3)	31.28(50)	0.00(0)	21.92(50)	1063.29(4)	0.00(0)	0.00(0)
50	50	55.31(50)	0.00(0)	96.69(50)	0.00(0)	**48.80(50)**	0.00(0)	0.00(0)	0.00(0)
60	50	**233.63(50)**	0.00(0)	549.40(50)	0.00(0)	345.89(50)	0.00(0)	0.00(0)	0.00(0)
70	50	**586.40(44)**	0.00(0)	892.17(30)	0.00(0)	1072.85(17)	0.00(0)	0.00(0)	0.00(0)
80	50	0.00(0)	0.00(0)	**1252.77(6)**	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)
90	50	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)
Solved	450	**344**	141	336	101	317	144	86	44
		s-supl		s-dir		s-supc		s-supxy	
Warehouse	#	nb	b	nb	b	nb	b	nb	b
warehouse	2	0.09(1)	2.37(1)	**0.08(1)**	2.35(1)	0.09(1)	2.36(1)	0.26(1)	1.34(1)
Solved	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Table 1. Comparison between branching schemes

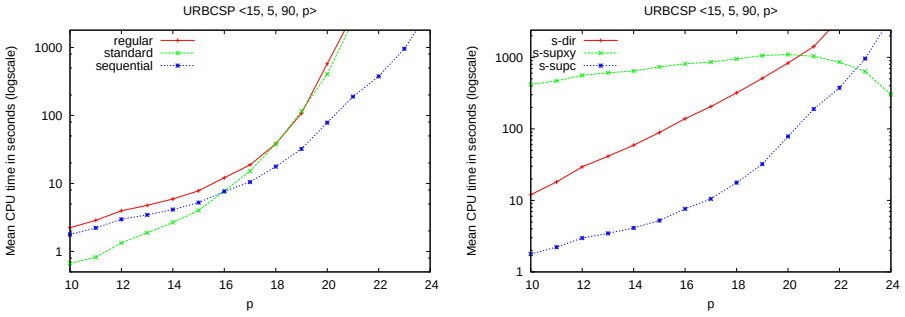

Fig. 1. Results for Random Max-CSP instances

that ignores auxiliary variables. The sequential versions of the encodings `dir`, `supc`, `supl` and `supxy` are denoted by `s-dir`, `s-supc`, `s-supl` and `s-supxy`.

In all the solved benchmarks, we observed that it is better to perform branching only on non-auxiliary variables. Table 1 compares this branching (nb) with the normal branching (b) for the instances in [4]. The gains of the new branching scheme are clear; for example, we solve up to 3 times more instances of clique trees using the sequential direct encoding `s-dir`. For the warehouse instances, the new branching scheme reduces the time needed to solve one instance. In the rest of experiments we assume that the branching is performed only on non-auxiliary variables. In all the tables, the cutoff time is of 30 minutes.

The left plot of Figure 1 compares standard, regular, and sequential encodings of Random Max-CSP instances with the minimal support encoding `supc`. We display encoding `supc` because it is the best performing encoding for this benchmark. The instances were obtained with a generator of uniform random binary CSPs 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. We observe that

		supc		supl		dir		supxy	
Kbtree (t)	#	sequential	regular	sequential	regular	sequential	regular	sequential	regular
10	50	1.24(50)	0.36(50)	0.10(50)	**0.07(50)**	15.01(50)	1.58(50)	89.18(44)	150.22(47)
20	50	57.07(50)	70.91(50)	**46.02(50)**	57.12(50)	345.30(49)	375.07(48)	0.00(0)	0.00(0)
30	50	**562.56(40)**	627.38(36)	559.65(38)	664.75(35)	1440.02(2)	0.00(0)	0.00(0)	0.00(0)
40	50	**1063.29(4)**	1341.48(2)	1175.17(3)	1714.93(2)	0.00(0)	0.00(0)	0.00(0)	0.00(0)
Solved	450	**144**	138	141	137	101	98	44	47
		supc		supl		dir		supxy	
Warehouses	#	sequential	regular	sequential	regular	sequential	regular	sequential	regular
warehouse	2	2.36(1)	2.43(1)	2.38(1)	2.46(1)	2.35(1)	2.43(1)	1.34(1)	**1.44(1)**
Solved instances	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Table 2. Comparison between sequential encodings and regular encodings

the sequential encoding is up to one order of magnitude faster than the standard and regular encodings. The right plot compares the different sequential encodings (direct, minimal and support) defined in this paper. We observe that the minimal encoding is the best performing except for large values of p, where the support encoding dominates. For lower values of p, there is a big gap between the minimal and support encodings. It is also remarkable the superiority of the minimal encoding wrt the direct encoding.

Table 2 compares sequential encodings with regular encodings on the instances used in [4]. Standard encodings are not included because they are worse than regular encodings [4]. We see that, in general, the sequential encodings outperform the regular encodings on both the time needed to solve an instance and the number of solved instances. We also see that the minimal encodings are the best performing encodings.

Finally, we notice that our encodings may be easily extended with weigths because there is exactly one violated clause for every violated constraint, as well as that the direct encoding may incorporate non-binary constraints. As future work, we plan to investigate structural properties of encodings that may be useful to predict their performance.

References

1. C. Ansótegui and F. Manyà. Mapping problems with finite-domain variables into problems with Boolean variables. In *SAT-2004*, pages 1–15. Springer 3542, 2004.
2. J. Argelich, A. Cabiscol, I. Lynce, and F. Manyà. Encoding Max-CSP into Partial Max-SAT. In *ISMVL-2008*. 2008.
3. J. Argelich, A. Cabiscol, I. Lynce, and F. Manyà. Modelling Max-CSP as Partial Max-SAT. In *SAT-2008*, pages 1–14. Springer LNCS 4996, 2008.
4. J. Argelich, A. Cabiscol, I. Lynce, and F. Manyà. Regular encodings from Max-CSP into Partial Max-SAT. In *ISMVL-2009*. 2009.
5. J. Argelich, C. M. Li, and F. Manyà. An improved exact solver for Partial Max-SAT. In *NCP-2007*, pages 230–231, 2007.
6. C. Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In *CP-2005*, pages 827–831. Springer LNCS 3709, 2005.
7. T. Walsh. SAT v CSP. In *CP-2000*, pages 441–456. Springer LNCS 1894, 2000.