Representative Encodings to Translate Finite CSPs into SAT

Pedro Barahona[1], Steffen Hölldobler[2], and Van-Hau Nguyen[2]

[1] Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
pb@fct.unl.pt
[2]International Center for Computational Logic
Technische Universität Dresden, 01062 Dresden, Germany
sh,hau@iccl.tu-dresden.de

Abstract. Solving Constraint Satisfaction Problems (CSPs) by Boolean Satisfiability (SAT) requires suitable encodings for translating CSPs to equivalent SAT instances that should not only be effectively generated, but should also be efficiently processed by SAT solvers. In this paper we investigate hierarchical and hybrid encodings, focussing on two specific encodings: the *representative-sparse encoding*, already proposed albeit not thoroughly tested, and a new *representative-order encoding*. Compared to the *sparse* and *order* encodings, two widely used and efficient encodings, these representative encodings require a smaller number of variables and longer clauses to express complex CSP constraints. The paper shows that, concerning unit propagation, these encodings are incomparable. Nevertheless, the preliminary experimental results show that the new representative encodings are clearly competitive with the original sparse and order encodings, but further studies are needed to better understand the trading between fewer variables and longer clauses of the representative encodings.

1 Introduction

Propositional satisfiability solving (SAT) is having an increasing impact on applications in various areas, ranging from artificial intelligence to hardware design and verification, and its success inspired a wide range of real-world and challenging applications.

Many such applications can be expressed as constraint satisfaction problems (CSPs) [24], although hardly any problems are originally given by SAT formulas. Nevertheless, to benefit from powerful SAT solvers, many SAT-encodings of CSPs have been studied recently, and finding suitable SAT encodings for CSPs is of greatest interest [32,1,21,1,15,9,23,31,28,12].

In fact, choosing an appropriate encoding is regarded as important as choosing an efficient algorithm and is considered to be one of the most exciting challenges in SAT solving [15]. Nevertheless, mapping a CSP into a SAT instance largely remains more of an art than a science [32,11,22,21,23], and some guidance is

H. Simonis (Ed.): CPAIOR 2014, LNCS 8451, pp. 251–267, 2014.

required for choosing appropriate encoding schemes for specific SAT-encoded instances.

Although other alternatives have been proposed, e.g. [32,9,29,31], in practice only two types of encodings are being widely used to encode an integer variable V into SAT: the *sparse encoding* (as denoted in [14], and adopted among others by the *direct encoding* [16,32] or the *support encoding* [11]) and the *order encoding* [7,2].

The *sparse encoding* is a common and effective encoding of CSPs to SAT instances. For example, Cadoni and Schaerf introduced a compiler (NP-SPEC), that translates a problem specification to SAT [6], Berre and Lynce built a SAT-based solver which competed in many cases against CSP solvers [4], and Jeavons and Petke [14] recently showed that modern SAT solvers using the *sparse encoding* can efficiently deal with certain families of CSPs which are even challenging for conventional constraint-programming solvers.

By facilitating the propagation of bounds, many combinatorial problems can be solved more efficiently by the *order encoding*, adopted in Sugar [28], BEE [18] and in lazy-clause generation [20].

In both encodings the number of SAT variables required to represent a CSP variable is linear on its domain size, which is quite penalizing for large domains, significantly consuming the runtime of SAT solvers. In contrast, the number of SAT variables required by the *log encoding* is only logarithmic on the domain size; unfortunately, it requires much longer clauses to encode most CSP constraints, and unit propagation is much less powerful than that obtained with the former encodings. Unlike the *sparse encoding* or the *order encoding*, we are unaware of any SAT-based solvers based on the *log encoding*.

Hierarchical and hybrid SAT encodings proposed by Velev [31] use one simple SAT encoding, (a variant of) the direct encoding or (a variant of) the log encoding, to recursively partition the domain of a CSP variable into subdomains. The twelve simple encodings combined with a variety of structures led to a numerous way of translations of a domain to SAT. However, it is impractical to determine how these variants behave for a given problem. Nguyen et al. [19] studied in some detail two special cases of hierarchical and hybrid SAT encodings, namely the *log-direct* and the *log-order* encodings, which are significantly more efficient than the *direct* and the *order* encodings.

In this paper we investigate the *representative-sparse* and the *representative-order* encodings, and show that despite the fact that the propagation achieved with these encodings is not comparable, the experimental results obtained with these representative encodings are quite promising, highlighting their potential for handling hard and practical problems.

Our work generalizes the work presented in [19], in which Nguyen et al. studied two specific representative encodings, using a log encoding of the first level. Here we study a more general encoding with several values at the first level, and include not only a theoretical comparison regarding the propagation strength of these encodings (missing in [19]) but also a more substantial empirical evidence obtained from three distinct state-of-the-art SAT solvers.

The rest of the paper is organized as follows. In Section 2, we briefly overview the *sparse* and the *order* encodings in the context of CSP problems. Section 3 introduces the *representative* encodings and presents their main features, namely their complexity and their propagation strength. Section 4, presents the experimental results obtained in a number of typical benchmarks, and show the significant speed-ups obtained with the representative encodings, when compared with their flat counterparts. The final section summarizes the main results and proposes further work.

2 Background

This section briefly overviews the two most widely and successfully used SAT encodings of CSP variables and constraints, viz. the *sparse encoding* and the *order encoding*, together with the basic concepts and notation of CSPs and SAT.

2.1 Constraint Satisfaction Problems (CSPs)

A *constraint satisfaction problem (CSP)* is a triple $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, where

- $\mathcal{V} = \{V_1, V_2, ..., V_k\}$ is a finite set of variables,
- $\mathcal{D} = \{D(V_1), D(V_2), ..., D(V_k)\}$ is a finite set of domains,
- $\mathcal{C} = \{C_1, C_2, ..., C_m\}$ is a finite set of constraints.

A tuple $\langle v_1, v_2, ..., v_k\rangle \in \langle D(V_1), D(V_2), ..., D(V_k)\rangle$ *satisfies* a constraint $C_i \in \mathcal{C}$ on the set of variables $\{V_{i1}, V_{i2}, ..., V_{il}\} \in \mathcal{V}$ if the projection of the tuple into these variables is a member of the constraint, i.e. $\langle v_{i1}, v_{i2}, ..., v_{il}\rangle \in C_i$. A tuple $\langle v_1, v_2, ..., v_k\rangle$ is a *solution* of a CSP iff it satisfies all constraints of $\mathcal{C}$. The CSP problem is to determine whether there exists one such tuple. In this paper, without loss of generality we assume that all variables have domain $\{1, ..., n\}$.

2.2 Boolean Satisfiability Problem (SAT)

A *formula in conjunctive normal form (CNF)* is a finite conjunction of clauses $C_1 \wedge C_2 \wedge ... \wedge C_m$, defined on a finite set $\{x_1, x_2, ..., x_n\}$ of propositional variables $(m, n \in \mathbb{N})$. A *clause* C is a finite disjunction of literals $l_1 \vee l_2 \vee ... \vee l_k$, where each literal l_i is either a Boolean variable x_l or its negation $(\neg x_l)$.

To each variable the truth values *false* (or 0) or *true* (or 1) can be assigned. A clause is *satisfied* by a truth assignment of the variables, if at least one of its literals is assigned value *true*. A CNF-formula $\mathcal{F}$ is *satisfiable* if there is a truth assignment that satisfies all its clauses, *unsatisfiable* otherwise. A *SAT problem* consisting of a CNF-formula $\mathcal{F}$ is the question whether a CNF-formula $\mathcal{F}$ is *satisfiable*.

2.3 Translating a Finite CSP to an Equivalent SAT Instance

The Sparse Encoding To encode a CSP variable V with domain $\{1, ..., n\}$ the *sparse encoding* uses n propositional variables d_i^V and the assignment $V = i$ is modelled by assigning d_i^V to *true* and all the other propositional variables to *false*.

Hence, the *sparse* encoding requires that exactly one d_i^V variable is assigned to *true*. Such constraint is achieved by means of a single *at-least-one* (ALO) clause:

$$d_1^V \vee d_2^V \vee ... \vee d_n^V$$

and a set of *at-most-one*(AMO) clauses, which may be naturally modelled by the pairwise encoding:

$$\bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \neg(d_i^V \wedge d_j^V) \equiv \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} (\neg d_i^V \vee \neg d_j^V)$$

where $\equiv$ denotes semantic equivalence. The pairwise encoding requires $\frac{n(n-1)}{2}$ binary clauses, but there are several efficient ways which only need $O(n)$ clauses (see [8,13]), like the sequential counter encoding [26] used in this paper.

The Conflict Clause The *direct encoding* [16] maps the CSP constraints onto a set of *conflict clauses*, modeling the disallowed variable assignments.

Adopting notions and notations from [23], let $K_{V,i}$ be the set of pairs (W, j) for which the assignments $(V = i, W = j)$ violate a CSP constraint. Then these constraints can be expressed by the following formula:

$$\bigwedge_{(w,j)\in K_{V,i}} \neg(d_i^V \wedge d_j^W) \equiv \bigwedge_{(w,j)\in K_{V,i}} (\neg d_i^V \vee \neg d_j^W)$$

The Support Clause. In contrast, the *support encoding* [11] maps the CSP constraints onto *support clauses* that specify the allowed variable assignments. Let $S_{V,j,W}$ be the values in the domain of V that support $W = j$ in some constraint. Then, the support clauses that model the constraint are expressed as:

$$d_j^W \rightarrow (\bigvee_{i\in S_{V,j,W}} d_i^V) \equiv \neg d_j^W \vee (\bigvee_{i\in S_{V,j,W}} d_i^V).$$

The Order Encoding. The *order encoding* [7,2] (a reformulation of the *sequential encoding* [13]) represents a CSP variable V with domain $\{1, ..., n\}$ by a vector of $n-1$ Boolean variables $[o_1^V, ..., o_{n-1}^V]$. To specify the assignment $V = i$ the first $i-1$ variables are assigned to *true* (or 1) and the remaining to $false$ (or 0). For example, the assignments $V = 1$, $V = 3$, and $V = 5$ for a variable V with domain $\{1, 2, 3, 4, 5\}$, are represented as $[0, 0, 0, 0]$, $[1, 1, 0, 0]$, and $[1, 1, 1, 1]$, respectively.

This encoding may be specified by a set of binary clauses

$$\bigwedge_{i=1}^{n-2} \neg(\neg o_i^V \wedge o_{i+1}^V) \equiv \bigwedge_{i=1}^{n-2} (o_i^V \vee \neg o_{i+1}^V)$$

which guarantee the desired properties [25]:

- if $o_i^V = 1$, then $o_j^V = 1$ for all $1 \leq j \leq i \leq n-1$,
- if $o_i^V = 0$, then $o_j^V = 0$ for all $1 \leq i \leq j \leq n-1$.

A CSP assignment $V = i$ is modelled by imposing $o_{i-1}^V = 1$ and $o_i^V = 0$, whereas its negation $V \neq i$ is represented by $o_{i-1}^V = o_i^V$ [18] (to cope with V = 1, we assume an extra bounding variable $o_0^V = 1$).

According to [2] the main advantage of this encoding is in the representation of interval domains and the propagation of their bounds. Indeed, the value of V may be restricted to a range $i...j$, by setting $o_{i-1}^V = 1$ and $o_j^V = 0$.

As with the *sparse encoding*, CSP constraints can be represented in the *order encoding* either by conflict or support clauses, with the obvious adaptations.

3 The Representative Encodings

3.1 The Representative-Sparse Encoding

The *representative-sparse encoding* is a hierarchical hybrid encoding, where Boolean *representative* variables $g_1, g_2, ..., g_m (1 \leq m \leq \frac{n}{2})$ at level 1 divide the domain into m subdomains represented at level 2 with Boolean *sparse* variables. These variables of both levels require ALO and AMO constraints. An assignment in this encoding is as follows:[1]

$$v = v_i \Leftrightarrow \begin{cases} g_{\lfloor i/r \rfloor} \wedge x_r & \text{if } i \bmod r = 0; \\ g_{\lfloor i/r \rfloor + 1} \wedge x_{i \bmod r} & \text{otherwise.} \end{cases} \quad (1)$$

Formula 1 translates a finite CSP domain $v = \{v_1, v_2, ..., v_n\}$ into SAT clauses by using Boolean representative variables, $\{g_1, ..., g_m\}$, and a set of Boolean sparse variables, $\{x_1, ..., x_r\}$, where $r = \lceil n/m \rceil$.

Fig 1 shows how the Boolean representative variables, $g_i, 1 \leq i \leq m$, assigns a value of V to the subdomains (note that when $n < m * r$ the prohibited values must be excluded).

Proposition 1. *When indexing the domain values of the CSP variables into SAT variables, the representative-sparse encoding is sound and complete.*

Proof: The *representative* variables $g_i, 1 \leq i \leq m \leq \frac{n}{2}$, at level one partition the domain of variable V into m subdomains $\{v_1, \ldots, v_{\lceil n/m \rceil}\}, \ldots,$ $\{v_{(m-1)\lceil n/m \rceil + 1}, \ldots, v_n\}$. Moreover, the *at-least-one* and *at-most-one* clauses for

[1] $\lfloor x \rfloor$ ($\lceil x \rceil$) is the biggest (smallest) integer number not bigger (less) than x, and *mod* is the remainder operator.

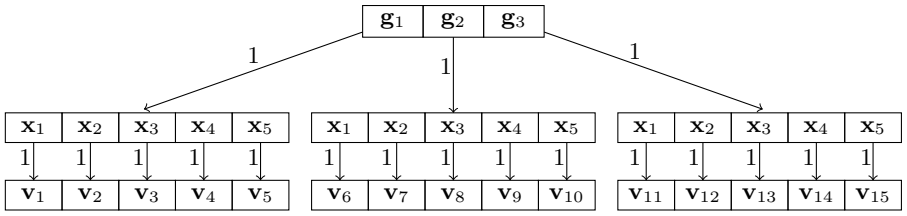

Fig. 1. An illustration of the representative-sparse encoding of the domain of a CSP variable using a group of *representative* variables at level one, $\{g_1, g_2, g_3\}$, and a set of Boolean *sparse variables* at level two, $\{x_1, x_2, x_3, x_4, x_5\}$, where exactly one variable is selected at each group

both sets of Boolean variables $\{g_1, \ldots, g_m\}$ and $\{x_1, \ldots, x_r\}$ lead to the selection of exactly one partition and exactly one value from the represented partition for the value of V. □

Note that the sparse encoding requires n Boolean variables to encode a CSP variable V with domain $\{v_1, \ldots, v_n\}$, whereas the representative-sparse encoding requires only $m + \lceil n/m \rceil$ Boolean variables, $1 \leq m \leq \frac{n}{2}$.

The *sparse encoding* is adopted by both the *direct encoding* and the *support encoding*. Unit propagation on the *direct encoding* maintains a form of consistency called forward checking [32], while unit propagation on the support encoding preserves arc-consistency [11]. Propagation for *representative-sparse encoding* is not stronger nor weaker with respect to the *sparse encoding* as stated in the following proposition.

Proposition 2. *Unit propagation applied to the representative-sparse encoding (when $m \geq 2$) is not comparable to the sparse encoding.*

Proof: We prove for the case $m = 3$ (the others are similar). Let CSP variables V and W have domain $\{1, .., 15\}$, as in Fig 1, and be constrained by $V \neq 3 \vee W \neq 5$. In the *representative-sparse encoding* the constraint is represented by a clause $\neg g_1^V \vee \neg x_3^V \vee \neg g_1^W \vee \neg x_5^W$ whereas by a simpler clause $\neg d_3^V \vee \neg d_5^W$ in the *sparse encoding*. For a subsequent assignment $W = 5$ (obtained during search or propagation), Unit propagation (UP) in the *sparse encoding* results in the unit clause $\neg d_3^V$, that may be further propagated, whereas in the *representative-sparse encoding* it leads to the non-unit binary clause $\neg g_1^V \vee \neg x_3^V$. Hence, in this case, UP in the *sparse encoding* is stronger than in the *representative-sparse encoding*.

On the other hand, let $V = \{2, 7, 12, 13\}$ be a constraint to be encoded into SAT. The constraint is represented by clauses $(x_2^V \vee g_3^V) \wedge (x_2^V \vee x_3^V)$ in the *representative-sparse encoding*, and by clause $d_2^V \vee d_7^V \vee d_{12}^V \vee d_{13}^V$ in the *sparse encoding*. For a subsequent assignment $V < 11$ (i.e. $\neg g_3^V$), then UP in the *representative-sparse encoding* results to the unit clause x_2^V, whereas in the *sparse encoding* leads to $d_2^V \vee d_7^V$, that is not further propagated. Hence, in this

case UP in the *representative-sparse encoding* is stronger than in the *sparse encoding*. □

In general, we may tailor the representative-sparse encodings for specific problems. For example, when the elimination of all odd or even values from a domain are frequent, we could set $m = n/2$, with only two variables at the second level, one for the odd and the other for the even values.

If the *representative-sparse encoding* cannot be compared with the *sparse encoding*, it may be so with the *log encoding*.

Proposition 3. *Unit propagation with the representative-sparse encoding (with $m = 2$) is more effective than with the log encoding.*

Proof: Similar to the proof of Theorem 15 in [32], after noting that the *log encoding* and the *representative-sparse encoding* share the same one literal when representing a CSP value domain. □

Finally, it is worth noting that one of the key strengths of the *representative-sparse encoding* is the ability to represent interval variables significantly better than the *sparse encoding* (in terms of the length clauses), namely when the interval does not cross the partitions. For example, to represent $V \geq 11$, the *sparse encoding* requires the "long "clause $d^V_{11} \vee d^V_{12} \vee d^V_{13} \vee d^V_{14} \vee d^V_{15}$, whereas the *representative-sparse encoding* simply requires g^V_3 to be set to *true*.

3.2 The Representative-Order Encoding

The *representative-order encoding* is a new hierarchical hybrid encoding, where level one is composed of Boolean *representative* variables $g_1, g_2, ..., g_m$ ($1 \leq m \leq \frac{n}{2}$) dividing the domain into m partitions represented at level two with *order* encoded variables. The variables of level one require ALO and AMO constraints, while the variables of level two require the condition of the order encoding. An assignment in this encoding is as follows:

$$v = v_i \Leftrightarrow \begin{cases} g_{\lfloor i/(r+1) \rfloor} \wedge x_r & \text{if } i \bmod (r+1) = 0; \\ g_{\lfloor i/(r+1) \rfloor + 1} \wedge x_1 & \text{if } i \bmod (r+1) = 1; \\ g_{\lfloor i/(r+1) \rfloor + 1} \wedge x_{i \bmod (r+1)-1} \wedge x_{i \bmod (r+1)} & \text{otherwise.} \end{cases} \tag{2}$$

Formula 2 for translating a finite CSP domain $v = \{v_1, v_2, ..., v_n\}$ into SAT clauses by using Boolean representative variables, $\{g_1, ..., g_m\}$, and a set of Boolean order variables, $\{x_1, ..., x_r\}$, where $r = \lceil n/m \rceil - 1$.

Note that the order encoding requires $n - 1$ Boolean variables to encode a CSP variable V with domain $\{v_1, \ldots, v_n\}$, whereas the representative-order encoding requires only $m + \lceil n/m \rceil - 1$ Boolean variables, $1 \leq m \leq \frac{n}{2}$. Fig 2 shows how the representative variables, $g_i, 1 \leq i \leq m$, assigns a value V to the subdomains. As with the *representative-sparse encoding* when $n < m * r$ the prohibited values must be excluded. This assignment is correct as stated in the following proposition.

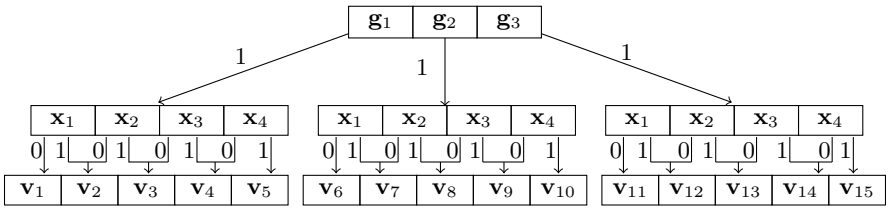

Fig. 2. An illustration of the representative-sparse encoding of the domain of a CSP variable using a group of *representative* variables at level one, $\{g_1, g_2, g_3\}$, where exactly one variable is selected, and a set of Boolean *order variables* at level two, $\{x_1, x_2, x_3, x_4\}$, where these variables are set in the constraint of the order encoding

Proposition 4. *When indexing the domain values of the CSP variables into SAT variables, the representative-order encoding is sound and complete.*

Proof: The representative variables at level one, $g_i, 1 \leq i \leq m \leq \frac{n}{2}$, divide the domain of variable V into m subdomains $\{v_1, \ldots, v_{\lceil n/m \rceil}\}, \ldots,$ $\{v_{(m-1)\lceil n/m \rceil+1}, \ldots, v_n\}$.

Given the *at-least-one* and *at-most-one* clauses for the representative Boolean variables $\{g_1, \ldots, g_m\}$ exactly one partition can be selected. Moreover, the constraints on the Boolean variables $\{x_1, \ldots, x_r\}$ imposed by the *order encoding*, lead to the selection for V of exactly one value in the selected partition. □

The comparison of the strength of unit propagation with the *order* and *representative-order* encodings is not straightforward. In the case of convex domains (intervals) the *order* encoding is usually simpler, as it sets one single positive and one single negative literal for the interval limits, whereas the *representative-order* encoding requires the setting of the representative literals as well. In the above example of CSP variables V having domain $\{1, .., 15\}$, an interval $V \in \{3..10\}$ is imposed by setting $o_2^V \wedge \neg o_{10}^V$ in the *order* encoding whereas it requires setting $(g_1^V \wedge x_2^V) \vee g_2^V$ in the *representative-order* encoding.

On the other hand, for non-convex domains, the *representative-order* encoding may be more compact. For example domain $V \in \{[2..4]; [7..9]; [12..14]\}$ is represented as $x_1^V \wedge \neg x_4^V$ the *representative-order* encoding whereas the *order* encoding requires the setting of $(o_1^V \wedge \neg o_4^V) \vee (o_6^V \wedge \neg o_9^V) \vee (o_{10}^V \wedge \neg o_{14}^V)$, i.e., 8 ternary clauses. Admittedly, this is an extreme example. Nevertheless, if the *order* encoding is usually more compact, many cases exist where the *representative-order* is superior, especially with non-convex domains.

4 Experiments

All experiments reported in this section were performed on a 2.66 Ghz, Intel Core 2 Quad processor with 3.8 GB of memory, under Ubuntu 10.04. Runtimes reported in CPU-time are in seconds. The used solvers are *riss3G* [17] (SAT competition 2013 version), *lingeling* [5] (SAT competition 2013 version) and *clasp* [10]

(*clasp2.1.3x86_64linux*) with default configurations, conflict-driven clause learning SAT solvers, which were ranked first on application and craft benchmarks in different categories at recent SAT competitions. [2] We use the *sequential* counter encoding [26] for the at-most-one constraint in the *representative-sparse* encodings. The columns S_1,S_2,$S_{\sqrt{n}}$, and $S_{n/2}$ (O_1,O_2,$O_{\sqrt{n}}$, and $O_{n/2}$) refer to the *representative-sparse* encodings (the *representative-order* encodings) with corresponding partitions (1, 2, $\sqrt{n}$ and $n/2$).

4.1 The Pigeon-Hole Problem

The goal of the problem is to prove that p pigeons can not be fit in $h = p-1$ holes. This is a trivially unsatisfiable problem, composed exclusively of disequality constraints on the CSP variables, which SAT solvers have difficulty to handle (there is no all different global constraint in SAT). This is confirmed in Table 1, for different size of the partitions (2, $\sqrt{n}$ and $n/2$), which are very similar in the case of domains of this sizes of ($n \in 10 \dots 15$). Nevertheless, the speedups show a large variation, depending on the solver, but the most relevant finding is that, despite their different performance, all solvers exhibit a speed-up of one to two orders of magnitude, when the problem is encoded with the representative encodings. The main reason for this speedups is the number of SAT variables required to encode the problems, that is much smaller with the representative encodings, as shown in Table 2, and compensate the fact that the clauses to encode disequality constraints are longer (but slightly less) in the representative encodings. Column *Inst* refers to p, the number of pigeons.

Table 1. The running time comparison of encodings performed by *riss3G*, *lingeling* and *clasp* on unsatisfiable Pigeon-Hole instances. Runtimes reported are in seconds.

Solvers	*Inst*	S_1	S_2	$S_{\sqrt{n}}$	$S_{n/2}$	O_1	O_2	$O_{\sqrt{n}}$	$O_{n/2}$
	11	64.23	1.59	0.73	0.82	0.28	0.27	0.75	0.93
	12	6773.73	13.26	2.82	2.97	0.61	1.08	2.34	1.04
riss3G	13	> 7200	22.14	43.74	263.73	4.23	1.71	9.51	27.02
	14	> 7200	240.24	944.55	773.06	22.69	6.28	31.37	36.72
	15	> 7200	3865.60	2085.45	> 7200	680.79	106.76	331.02	1820.30
	11	8.10	1.22	1.47	1.50	0.53	0.68	1.29	1.43
	12	863.76	7.69	5.43	2.61	1.45	3.91	4.38	6.80
lingeling	13	> 7200	33.96	12.40	37.55	10.02	15.68	5.57	27.38
	14	> 7200	514.35	101.09	273.52	24.31	36.15	79.14	53.97
	15	> 7200	4540.23	738.28	4293.51	136.83	137.08	98.79	2600.62
	11	10.94	0.52	0.91	0.53	1.43	0.62	1.867	0.53
	12	110.11	2.84	4.92	3.56	8.26	2.73	14.15	4.37
clasp	13	1127.97	12.93	14.58	12.17	45.45	9.95	11.26	12.22
	14	> 7200	75.10	94.71	77.50	602.51	32.34	100.32	73.67
	15	> 7200	386.52	333.03	386.26	4264.07	153.42	305.97	368.51

Finally, it is interesting to note that the *representative-order* encodings perform better than the *representative-sparse* encodings, even if difference constraints do not require handling of bounds of the domains, and this is possibly

[2] `http://www.satcompetition.org`

Table 2. The number of variables and clauses on unsatisfiable Pigeon-Hole instances

Inst	S_1	S_2	$S_{\sqrt{n}}$	$S_{n/2}$	O_1	O_2	$O_{\sqrt{n}}$	$O_{n/2}$
11	220| 869	121| 704	110| 715	88| 704	110| 638	66| 605	66| 638	66| 671
12	264|1086	156| 942	120| 894	120| 978	132| 834	84| 810	72| 810	84| 918
13	312|1391	169|1157	130|1105	117|1183	156|1066	91|1014	78|1014	91|1144
14	364|1715	210|1477	168|1435	154|1561	182|1337	112|1295	98|1309	112|1491
15	420|2085	225|1770	180|1725	150|1845	210|1650	120|1575	105|1590	120|1800

caused by the use of the *sequential counter* to code the AMO constraints in the *representative-sparse* encodings.

4.2 The Golomb Ruler Problem

A Golomb ruler (m,g) can be defined as a sequence $0 \leq a_1 < a_2 < \cdots < a_m$ of m integers such that the $m(m-1)/2$ differences $a_j - a_i$ $(1 \leq i < j \leq m)$ are all distinct. We executed the satisfiable version of the problem to check whether there is a ruler with $a_m \leq g$.

Table 3. The running time comparison of encodings performed by *riss3G*, *lingeling* and *clasp* for finding one solutions on Golomb Ruler instances. Runtimes reported are in seconds.

Solvers	*Inst*	S_1	S_2	$S_{\sqrt{n}}$	$S_{n/2}$	O_1	O_2	$O_{\sqrt{n}}$	$O_{n/2}$
riss3G	G(9,44)	0.49	0.76	0.82	0.52	0.13	0.80	1.56	0.20
	G(10,55)	2.84	1.21	3.46	2.82	1.23	1.46	3.42	5.90
	G(11,72)	1.16	9.72	29.7	48.18	33.68	54.24	9.02	0.69
	G(12,85)	185.26	719.41	578.62	3121.12	235.89	35.06	705.06	5.85
lingeling	G(9,44)	0.51	1.22	0.94	4.62	0.34	2.16	1.30	2.09
	G(10,55)	9.27	2.26	13.33	5.53	1.09	6.78	13.42	16.72
	G(11,72)	112.13	35.40	20.47	76.47	125.24	51.07	73.83	152.17
	G(12,85)	566.23	753.03	458.22	84.60	685.95	356.89	2272.25	5200.11
clasp	G(9,44)	0.91	2.24	3.15	3.52	0.26	4.59	4.49	1.26
	G(10,55)	2.48	7.38	21.85	30.73	2.51	15.78	28.73	25.40
	G(11,72)	6.76	69.63	128.88	654.78	101.67	473.63	1155.79	854.12
	G(12,85)	409.53	1229.56	6992.35	> 7200	1217.28	5720.18	4703.78	4593.86

The problem includes disequality and inequality constraints on the CSP variables. Despite these latter constraints, the *representative-sparse* encodings seem to perform better than the *representative-order* encodings. The domains are now larger (from 44 to 85) and the execution times obtained in this problem suggest that in this problem the speed-ups obtained with the representative encodings are not very significant, with the notable exception of the *riss3G* solver.

Table 4. The number of variables and clauses of different encodings on Golomb Ruler instances

Inst	S_1		S_2		$S_{\sqrt{n}}$		$S_{n/2}$	
G(9,44)	3978	105694	2043	102733	945	101608	1143	110563
G(10,55)	6070	203175	3135	198735	1210	196535	1760	215235
G(11,72)	9526	411767	4840	404650	1672	401152	2596	439806
G(12,85)	13284	686628	6786	676830	2184	672150	3666	737670
Inst	O_1		O_2		$O_{\sqrt{n}}$		$O_{n/2}$	
G(9,44)	1898	135718	1044	133181	585	127610	1044	110410
G(10,55)	3035	168182	1595	260056	770	247681	1595	238041
G(11,72)	4763	350770	2453	536448	1067	512424	2453	439582
G(12,85)	6624	593115	3432	900206	1404	863516	3432	817664

Table 4 shows the number of variables and clauses required by different encodings. Clearly, the variability of the speedups suggest that the combined effect of less variables and longer clauses must be further investigated.

Table 5. The running time comparison of different encodings performed by *riss3G* on Graph-Colouring instances. S/U indicates the satisfiability or unsatisfiability of instances. K is the number of colours used.

Inst	K	C	S_1	S_2	$S_{\sqrt{n}}$	$S_{n/2}$	O_1	O_2	$O_{\sqrt{n}}$	$O_{n/2}$
jean	9	U	11.03	0.48	0.19	0.18	0.13	0.18	0.16	0.19
	10	S	0.02	0.03	0.02	0.02	0.01	0.02	0.03	0.03
anna	10	U	204.87	2.02	0.76	2.62	0.53	0.85	0.49	2.51
	11	S	0.09	0.06	0.05	0.06	0.03	0.05	0.06	0.05
david	10	U	156.84	2.50	0.81	2.11	0.45	1.02	0.70	2.17
	11	S	0.06	0.05	0.04	0.04	0.02	0.04	0.06	0.04
DSJC125.9	10	U	940.19	6.10	6.03	5.11	1.92	5.23	5.36	5.90
	11	U	>7200	13.86	10.18	5.23	5.82	7.67	8.55	6.13
huck	10	U	213.12	2.39	1.09	2.90	0.42	0.71	1.07	2.22
	11	S	0.04	0.03	0.03	0.02	0.02	0.04	0.04	0.03
miles500	10	U	606.15	2.92	1.36	2.96	1.09	0.97	1.13	3.03
	11	U	5872.35	8.93	2.87	2.15	12.32	2.78	3.93	3.29
miles750	10	U	580.88	2.38	2.18	2.61	0.84	1.13	0.93	2.74
	11	U	>7200	15.50	10.92	2.93	4.51	5.60	5.93	2.98
miles1000	10	U	531.56	2.98	2.12	3.58	0.90	1.82	3.87	2.71
	11	U	>7200	8.54	2.31	2.45	2.59	3.60	4.00	7.75
miles1500	10	U	959.88	4.61	3.66	4.43	1.66	3.62	4.52	4.96
	11	U	>7200	11.75	6.58	4.12	3.49	6.67	7.21	4.65
queen12_12	10	U	270.75	2.51	1.71	1.96	1.02	1.64	1.34	3.30
	11	U	4926.12	7.97	4.12	4.49	2.93	4.44	11.73	4.72
queen13_13	10	U	258.80	3.35	2.00	2.86	0.93	1.82	1.90	2.90
	11	U	2359.51	11.20	7.10	7.54	7.12	5.61	7.05	6.69
queen14_14	10	U	276.38	4.38	1.66	2.72	1.06	2.22	2.32	2.61
	11	U	3057.18	19.35	2.46	2.76	3.64	5.16	4.34	18.61
queen15_15	10	U	472.29	3.36	3.24	3.14	1.22	2.47	2.59	3.95
	11	U	6478.20	23.94	7.95	2.96	6.74	6.04	9.75	6.55
queen16_16	10	U	170.77	3.74	2.89	3.40	1.68	3.02	3.44	4.12
	11	U	2829.32	16.24	9.52	14.33	10.25	6.56	4.32	9.35

4.3 The Graph Colouring Problem

The well known Graph Colouring problem consists of finding an assignment of k colours to the vertices of an undirected graph, such that no two adjacent vertices share the same colour. We experimented on several hard unsatisfiable instances from [30], and report the results obtained in Table 5 and Table 6 (results obtained with *clasp* were similar to those with *riss3G* and were omitted for lack of space).

Like with the pigeon hole problem, the *representative-order* encodings perform better than the *representative-sparse* encodings, and in the former the representative encodings have similar runtimes, with some marginal speedup. The effect of the representative encoding much stronger in the *sparse* case, where in both solvers, speed-ups of one or two orders of magnitude are achieved with the *representative-sparse* encodings, typically favoring a $\sqrt{n}$ number of partitions.

Table 6. The running time comparison of different encodings performed by *lingeling* on Graph-Colouring instances. *S/U* indicates the satisficability or unsatisfiability of instances. K is the number of colors used. Runtimes reported are in seconds.

Inst	*K*	*S/U*	S_1	S_2	$S_{\sqrt{n}}$	$S_{n/2}$	O_1	O_2	$O_{\sqrt{n}}$	$O_{n/2}$
jean	9	U	0.02	0.02	0.02	0.02	0.74	0.68	0.65	0.80
	10	S	20.86	0.81	1.53	0.84	0.01	0.01	0.02	0.02
anna	10	U	142.53	1.74	2.28	1.70	1.06	1.52	1.44	2.01
	11	S	0.04	0.07	0.07	0.07	0.04	0.05	0.04	0.05
david	10	U	130.68	2.24	2.03	3.17	1.03	1.52	2.02	2.03
	11	S	0.02	0.05	0.05	0.06	0.03	0.04	0.03	0.04
huck	10	U	119.64	1.77	1.78	1.86	0.98	1.36	1.90	1.44
	11	S	0.02	0.04	0.04	0.04	0.02	0.02	0.02	0.02
DSJC125.9	10	U	0.04	2.85	2.32	2.96	1.92	2.04	1.29	2.41
	11	U	0.04	13.17	10.49	6.96	13.65	4.67	4.70	5.73
miles500	10	U	1532.22	1.78	2.34	2.26	3.91	1.90	1.60	2.79
	11	U	>7200.0	25.77	9.62	8.36	5.48	3.71	5.37	3.68
miles750	10	U	2041.76	2.39	2.72	2.08	1.90	2.04	1.05	2.80
	11	U	6685.31	36.54	7.96	7.80	6.94	5.14	2.94	6.50
miles1000	10	U	2.76	2.49	2.63	2.07	2.32	2.23	1.17	2.11
	11	U	2331.27	47.70	10.36	6.35	8.08	4.96	3.56	6.56
miles1500	10	U	0.56	3.85	2.62	3.34	3.96	3.04	1.63	3.15
	11	U	4.18	43.68	8.17	7.62	15.23	9.22	3.90	6.48
queen12_12	10	U	721.29	2.70	3.04	2.28	1.22	2.08	1.06	2.09
	11	U	2185.73	20.20	5.42	6.62	2.68	5.60	3.01	3.89
queen13_13	10	U	488.37	2.35	1.28	2.08	1.73	2.07	1.15	2.64
	11	U	>7200.0	35.73	6.60	6.47	3.74	4.69	5.26	4.64
queen14_14	10	U	715.68	2.80	1.38	2.81	2.96	2.43	1.36	1.98
	11	U	>7200.0	29.44	6.50	16.59	6.79	5.56	6.16	5.74
queen15_15	10	U	1760.25	2.96	1.60	2.42	4.52	2.53	1.46	2.91
	11	U	>7200.0	27.35	7.59	5.76	15.30	5.10	3.75	4.93
queen16_16	10	U	1679.02	3.25	1.83	3.13	4.33	2.94	1.63	2.20
	11	U	>7200.0	34.58	5.68	6.49	11.94	6.12	3.97	5.10

4.4 The Open Shop Scheduling Problem

Given a set of jobs, each consisting of a set of tasks that must be processed once in any order on a set of machines, the goal of this combinatorial optimization problem is to find a schedule of all tasks so as to complete all jobs within a given makespan. All CSP constraints of this problem are inequalities. The benchmarks used here were taken from [27]. Column *Instance* identifies the instances of the problem, where a_b refers to the b^{th} instance of the benchmark with a jobs and a machines. Again, results obtained with *clasp* were similar to those with *riss3G* and were omitted for lack of space.

Table 7. The running time comparison of encodings performed by *riss3G* on Open Shop Scheduling instances. M is the makespan used. S/U indicates the satisfiability or unsatisfiability of instances. Runtimes reported are in seconds.

Inst	M	C	S_1	S_2	$S_{\sqrt{n}}$	$S_{n/2}$	O_1	O_2	$O_{\sqrt{n}}$	$O_{n/2}$
4_1	192	U	2.47	0.38	0.13	1.01	0.04	0.06	0.13	0.82
	193	S	1.96	0.34	0.12	0.93	0.03	0.03	0.13	0.82
4_2	235	U	7.23	0.90	0.31	2.36	0.12	0.11	0.28	2.88
	236	S	5.45	0.66	0.29	2.61	0.03	0.05	0.26	1.40
4_3	270	U	7.92	1.03	0.37	3.97	0.10	0.08	0.25	3.48
	271	S	6.56	0.91	0.23	2.51	0.06	0.04	0.16	2.51
4_4	249	U	6.08	1.06	0.33	2.73	0.09	0.12	0.30	3.43
	250	S	8.25	0.95	0.28	2.38	0.10	0.12	0.32	3.16
4_5	294	U	11.63	1.80	0.42	4.85	0.14	0.12	0.46	4.52
	295	S	10.32	1.32	0.30	4.52	0.09	0.07	0.25	3.44
5_1	299	U	34.26	5.34	1.81	14.98	0.56	0.64	1.34	14.07
	300	S	28.44	3.96	1.20	16.76	0.26	0.36	1.16	17.72
5_2	261	U	22.97	3.52	1.56	13.38	0.41	0.80	1.32	11.89
	262	S	16.83	2.61	0.91	10.52	0.38	0.30	0.60	9 .13
5_3	322	U	57.62	8.71	2.08	27.43	0.74	1.02	1.71	27.35
	323	S	59.26	8.36	2.45	33.86	0.63	0.46	2.02	37.72
5_4	309	U	49.51	9.57	2.81	31.54	0.81	0.81	2.06	27.39
	310	S	55.83	7.74	2.59	30.57	0.64	0.78	1.53	28.57
5_5	325	U	89.77	14.22	4.74	48.54	1.32	1.62	3.27	45.41
	326	S	73.65	5.45	3.07	32.27	0.97	1.20	2.72	29.58
7_1	434	U	734 .63	76.14	53.58	438.75	1.52	2.03	19.52	406.17
	435	S	3091.31	428.58	75.24	2282.58	3.84	9.64	47.15	2454.82
7_2	442	U	613.37	28.63	9.23	124.79	1.41	2.24	14.24	471.24
	443	S	2763.21	96.94	63.52	2617.25	7.83	14.76	39.42	1990.25
7_3	467	U	234.10	18.38	14.62	251.62	1.69	6.07	37.15	229.25
	468	S	>7200	1666.27	267.08	3986.71	41.64	30.81	187.10	4756.72
7_4	462	U	434.92	23.73	13.41	195.62	1.38	6.05	27.93	339.06
	463	S	5729.13	567.61	92.15	3914.25	6.20	22.28	75.75	3905.14
7_5	415	U	272.00	20.22	30.14	226.33	1.24	7.12	7.72	477.94
	416	S	1938.91	489.42	84.89	1909.78	7.26	7.59	58.20	1057.52

As expected, in this problem where CSP inequality constraints dominate, the *order* encoding is much faster than the *sparse* encoding. The former is particularly adequate to propagate changes in the bounds, and no speedups were expected with representative encodings (and in fact a slow-down is observed).

What is more interesting is that the *representative-sparse* encodings perform very similarly to the *representative-order* encodings, in both solvers, possibly due to the fact that the partition of the large size domains (size 192 to 416) efficiently propagates the changes in the bounds, for which the sparse encoding is not suited. Overall, the representative encodings perform worse than the *order* encoding but significantly better than the *sparse* encoding, specially when the number of SAT variables in levels 1 and 2 are balanced ($S_{\sqrt{n}}$).

Table 8. The running time comparison of encodings performed by *lingeling* on Open Shop Scheduling instances. M is the makespan used. S/U indicates the satisfiability or unsatisfiability of instances. Runtimes reported are in seconds.

Inst	M	C	S_1	S_2	$S_{\sqrt{n}}$	$S_{n/2}$	O_1	O_2	$O_{\sqrt{n}}$	$O_{n/2}$
4_1	192	U	1.81	0.92	0.31	3.72	0.12	0.17	0.66	2.15
	193	S	8.31	1.10	0.32	3.78	0.12	0.16	0.69	2.23
4_2	235	U	22.14	2.40	1.11	4.60	0.28	0.28	0.98	4.88
	236	S	18.16	2.13	1.03	4.67	0.10	0.21	0.94	4.16
4_3	270	U	43.43	2.81	1.26	8.82	0.30	0.28	0.50	7.37
	271	S	47.99	2.80	1.19	6.91	0.24	0.26	0.49	7.12
4_4	249	U	23.03	2.67	1.12	6.71	0.25	0.29	0.93	6.88
	250	S	27.11	2.36	1.15	5.26	0.21	0.22	0.91	5.18
4_5	294	U	41.87	3.39	1.42	9.54	0.34	0.33	0.68	11.04
	295	S	42.16	2.77	1.29	7.03	0.28	0.31	0.51	7.11
5_1	299	U	192.11	47.74	3.87	37.16	1.07	1.23	3.46	34.22
	300	S	185.54	47.71	4.14	32.39	0.40	1.05	2.33	37.01
5_2	261	U	132.14	30.11	3.54	27.93	1.02	1.17	3.22	25.20
	262	S	106.02	2.53	2.67	19.09	0.61	0.70	2.79	20.82
5_3	322	U	242.74	66.64	4.41	40.80	1.29	2.46	3.52	48.04
	323	S	230.84	58.65	6.24	38.61	1.62	1.63	3.52	46.48
5_4	309	U	271.32	51.13	6.48	38.97	1.59	1.89	5.35	51.42
	310	S	114.86	47.44	5.92	36.84	0.94	1.24	5.06	51.57
5_5	325	U	351.43	76.02	8.64	53.36	2.91	0.30	4.56	62.49
	326	S	249.76	72.24	6.43	40.63	2.32	0.31	5.74	45.34
7_1	434	U	6883.68	602.19	55.61	661.07	1.95	1.62	51.16	628.88
	435	S	>7200	644.73	73.62	685.32	9.54	11.40	75.56	522.86
7_2	442	U	3675.26	389.77	50.41	358.64	4.03	2.14	46.13	468.23
	443	S	3454.09	414.08	76.16	594.33	8.86	9.70	50.07	822.75
7_3	467	U	6679.69	804.14	82.39	404.46	2.26	16.43	88.32	2108.54
	468	S	>7200	4026.52	162.19	2074.28	34.81	33.72	91.38	2236.85
7_4	462	U	>7200	1853.84	58.75	944.45	2.09	1.86	53.96	910.24
	463	S	6767.76	1668.02	62.32	913.24	23.04	20.57	73.70	520.93
7_5	415	U	4049.08	712.38	47.01	405.28	2.20	3.46	40.76	306.70
	416	S	2279.14	807.89	48.15	404.55	6.98	9.21	79.28	735.05

5 Conclusions and Future Works

This paper introduces two specific hierarchical hybrid encodings, the *representative-sparse* and *representative-order* encodings, for modeling CSPs as propositional SAT problems, which aim at taking advantage of the sparse and order encodings, but requiring significant smaller number of SAT variables. The new encodings, that use only two levels in the hierarchy (all experiments we did with three or more levels were clearly less efficient) can be parameterised by different sizes of the level one of the hierarchy, and in general these hierarchical encodings are incomparable with respect to their flat counterparts (the *sparse* and *-order* encodings which are special cases with one single partition). The experimental results in a set of representative benchmarks show that, regardless of the variability of run times in different SAT solvers, the representative encodings are quite competitive and usually outperform (sometimes very significantly) the *sparse* and *order* encodings, with the exception of CSP problems where inequality constraints dominate, in which case the *order* encoding is still the best option. More work remains to be done to further assess the merit of these new encodings and, in particular, we intend to further investigate how to tune the number of partitions for different CSP problems and SAT solvers. One appealing work into apply some redundancy for the second level of the representative encodings, as outlined in [3].

Acknowledgment. We thank Christoph Wernhard, Norbert Manthey, Peter Steinke, and Tobias Philipp for useful suggestions. We are grateful to Miroslav N. Velev for the previous works.

References

1. Ansótegui, C., del Val, A., Dotú, I., Fernández, C., Manyà, F.: Modeling Choices in Quasigroup Completion: SAT vs. CSP. In: McGuinness, D.L., Ferguson, G. (eds.) Proceedings of the 19th National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, pp. 137–142. AAAI Press / The MIT Press (2004)
2. Bailleux, O., Boufkhad, Y.: Efficient CNF Encoding of Boolean Cardinality Constraints. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 108–122. Springer, Heidelberg (2003)
3. Barahona, P., Hölldobler, S., Nguyen, V.H.: Efficient SAT-Encoding of Linear CSP Constraints. In: 13rd International Symposium on Artificial Intelligence and Mathematics - ISAIM, Fort Lauderdale, Florida, USA, January 6-8 (2014)
4. Berre, D.L., Lynce, I.: CSP2SAT4J: A Simple CSP to SAT Translator. In: van Dongen, M., Lecotre, C., Rossel, O. (eds.) Proceedings of the Second International CSP Solver Competition, pp. 43–54 (2006)
5. Biere, A.: Lingeling, Plingeling and Treengeling Entering the SAT Competition 2013. In: Adrian Balint, A.B., Heule, M., Järvisalo, M. (eds.) Proceedings of SAT Competition 2013, pp. 51–52 (2013)
6. Cadoli, M., Schaerf, A.: Compiling Problem Specifications into SAT. Artificial Intelligence 162(1-2), 89–120 (2005)

7. Crawford, J.M., Baker, A.B.: Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems. In: Hayes-Roth, B., Korf, R.E. (eds.) Proceedings of the 12th National Conference on Artificial Intelligence, vol. 2, pp. 1092–1097. AAAI Press / The MIT Press (1994)
8. Frisch, A.M., Giannoros, P.A.: SAT Encodings of the At-Most-k Constraint. Some Old, Some New, Some Fast, Some Slow. In: Proc. of the Tenth Int. Workshop of Constraint Modelling and Reformulation (2010)
9. Gavanelli, M.: The Log-Support Encoding of CSP into SAT. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 815–822. Springer, Heidelberg (2007)
10. Gebser, M., Kaufmann, B., Schaub, T.: The Conflict-Driven Answer Set Solver clasp: Progress Report. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 509–514. Springer, Heidelberg (2009)
11. Gent, I.P.: Arc Consistency in SAT. In: van Harmelen, F. (ed.) Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI 2002, pp. 121–125. IOS Press (2002)
12. Hölldobler, S., Manthey, N., Nguyen, V., Steinke, P.: Solving Hidokus Using SAT Solvers. In: Proc. INFOCOM-5, pp. 208–212 (2012) ISSN 2219-293X
13. Hölldobler, S., Nguyen, V.H.: On SAT-Encodings of the At-Most-One Constraint. In: Katsirelos, G., Quimper, C.G. (eds.) Proc. Twelfth International Workshop on Constraint Modelling and Reformulation, Uppsala, Sweden, September 16-20, pp. 1–17 (2013)
14. Jeavons, P., Petke, J.: Local Consistency and SAT-Solvers. J. Artif. Intell. Res (JAIR) 43, 329–351 (2012)
15. Kautz, H., Selman, B.: The State of SAT. Discrete Appl. Math. 155, 1514–1524 (2007)
16. de Kleer, J.: A Comparison of ATMS and CSP Techniques. In: IJCAI, pp. 290–296 (1989)
17. Manthey, N.: The SAT Solver RISS3G at SC 2013. Department of Computer Science Series of Publications B, vol. B-2013-1, pp. 72–73. University of Helsinki, Helsinki, Finland (2013)
18. Metodi, A., Codish, M.: Compiling Finite Domain Constraints to SAT with BEE. Theory and Practice of Logic Programming 12(4-5), 465–483 (2012)
19. Nguyen, V.H., Velev, M.N., Barahona, P.: Application of Hierarchical Hybrid Encodings to Efficient Translation of CSPs to SAT. In: Brodsky, A. (ed.) Proc. 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013), Special Track on SAT and CSP. pp. 1028–1035. Conference Publishing Services (2013)
20. Ohrimenko, O., Stuckey, P., Codish, M.: Propagation via Lazy Clause Generation. Constraints 14(3), 357–391 (2009)
21. Prestwich, S.D.: Full Dynamic Substitutability by SAT Encoding. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 512–526. Springer, Heidelberg (2004)
22. Prestwich, S.D.: Local Search on SAT-encoded Colouring Problems. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 105–119. Springer, Heidelberg (2004)
23. Prestwich, S.D.: CNF Encodings, ch. 2, pp. 75–98. IOS Press (2009)
24. Rossi, F., Beek, P.V., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York (2006)
25. Marques-Silva, J., Lynce, I.: Towards Robust CNF Encodings of Cardinality Constraints. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 483–497. Springer, Heidelberg (2007)

26. Sinz, C.: Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 827–831. Springer, Heidelberg (2005)
27. Taillard, É.: http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html
28. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling Finite Linear CSP into SAT. Constraints 14(2), 254–272 (2009)
29. Tanjo, T., Tamura, N., Banbara, M.: A Compact and Efficient SAT-Encoding of Finite Domain CSP. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 375–376. Springer, Heidelberg (2011)
30. Trick, M.: DIMACS Graph-Coloring Problems, http://mat.gsia.cmu.edu/COLOR/instances.html
31. Velev, M.N.: Exploiting Hierarchy and Structure to Efficiently Solve Graph Coloring as SAT. In: International Conference on Computer-Aided Design (ICCAD 2007), San Jose, CA, USA, pp. 135–142. IEEE (2007)
32. Walsh, T.: SAT *v* CSP. In: Dechter, R. (ed.) CP 2000. LNCS, vol. 1894, pp. 441–456. Springer, Heidelberg (2000)