Efficient SAT-Encoding of Linear CSP Constraints

Pedro Barahona
Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
2829-516 Caparica, Portugal

Steffen Hölldobler and **Van Hau Nguyen**
International Center for Computational Logic
Technische Universität Dresden
01062 Dresden, Germany

Abstract

Propositional satisfiability solving (SAT) has been considerably successful in numerous industrial applications. Whereas the speed and the capacity of SAT solvers significantly improved in the last two decades, the understanding of SAT encodings is still limited and often challenging. Two well-known variable encodings, namely the *order encoding* and the *sparse encoding*, are the most widely used and successfully applied to translate constraint satisfaction problems (CSPs) to equivalent SAT instances. In this paper we analyze the strengths and drawbacks of these encoding in the context of linear CSP problems and propose a new *Sp-Or encoding*, based on redundant modeling common in Constraint Programming. We show experimentally that the runtime overhead of the Sp-Or encoding is not significant in the worst case compared to the individual encodings whereas it does outperform them in some of the CSPs. The paper concludes with some guidelines regarding the choice of suitable SAT encodings for CSP problems, taking into account several features of these problems.

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) (Rossi, Beek, and Walsh 2006), whilst hardly any problems are originally given by SAT formulas. Nevertheless, to take advantage of powerful SAT solvers, many SAT-encodings of CSPs have been studied recently, and finding suitable SAT encodings for CSPs is of greatest interest (Walsh 2000; Prestwich 2004; Kautz and Selman 2007; Gavanelli ; Velev 2007; Prestwich 2009; Tamura et al. 2009; Hölldobler et al. 2012; Nguyen, Velev, and Barahona 2013).

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 (Kautz and Selman 2007). Nevertheless, mapping a CSP into a SAT instance largely remains more of an art than a science (Walsh 2000; Gent 2002; Prestwich 2004; 2009), and some guidance is required for choosing appropriate encoding schemes for specific SAT-encoded instances.

Although other alternatives have been proposed (Iwama and Miyazaki 1994; Walsh 2000; Gavanelli ; Velev 2007), in practice only two types of encodings are being used to encode an integer variable V into SAT: the *sparse encoding* (Jeavons and Petke 2012) was adopted, among others, by the *direct encoding* (De Kleer 1989; Walsh 2000) or the *support encoding* (Gent 2002); and the *order encoding* (Bailleux and Boufkhad 2003; Ansótegui and Manyà 2004; Crawford and Baker 1994)

The sparse encoding is still the most common. For example, Cadoli and Schaerf introduced a compiler (NP-SPEC), that translates a problem specification to SAT (Cadoli and Schaerf 2005), Berre and Lynce built a SAT-based solver which competed in many cases against CSP solvers (Berre and Lynce 2008), and Jeavons and Petke (Jeavons and Petke 2012) pointed out that modern clause learning SAT solvers using the sparse encoding can efficiently deal with certain CSPs which are even challenging for conventional constraint programming solvers. Nevertheless, many combinatorial problems can be solved more efficiently by the order encoding: It was adopted in Sugar (Tamura et al. 2009), an award winning solver of global constraint categories in recent CSP solver competitions (van Dongen, Lecotre, and Rossel 2008; 2009), in BEE, which optimizes a CNF formula yielding considerable speed-ups in SAT solving time (Metodi and Codish 2012), and in lazy-clause generation (Ohrimenko, Stuckey, and Codish 2009). Furthermore, Petke and Jeavons (Petke and Jeavons 2011) showed that the order encoding can translate various tractable CSPs into tractable SAT instances.

When transforming a CSP problem into a SAT instance, one typically adopts one of the above encodings. While some efforts are taken to encode the constraints as efficiently as possible (Argelich et al. 2010), we are unaware of any work specifying clearly which encoding one should use when facing a particular CSP.

In this paper, we address the case of SAT-encodings for common CSP problems where most of the constraints have the form $X \pm c \rhd Y$, where X and Y are integer variables, c is a constant, and $\rhd$ is a relational operator. Not surprisingly, we confirm that the order encoding is better for problems with dominant inequality constraints ($\rhd \in \{>, \geq, <, \leq\}$) but in problems where this is not the case the advantages of one or the other encoding are not obvious, and one is left with the problem of choosing the adequate encoding.

In the sequel, we address this problem and discuss some features of the initial CSP problems that might be considered to prefer one or the other encodings. Moreover, we study the combination of both encodings. In constraint programming, motivated by the effectiveness of redundant constraints, (Cheng, Lee, and Wu 1996) introduced the redundant modelling that can speed up constraint propagation. In this paper we introduce redundant modelling in SAT encoding. This approach is already incipient in the *sequential encoding* (Sinz 2005) that implements the at-most-one constraint required by the sparse encoding by means of extra variables similar to the order encoding variables. Likewise, (Ansótegui and Manyà 2004) introduced the *regular encoding*, which used regular literals to avoid encoding the at-least-one (ALO) and at-most-one (AMO) constraints. On the other hand, in addition to the order encoding, BEE uses extra variables similar to those used in the sparse encoding to encode all-different constraints (Metodi and Codish 2012). We go a step further and discuss more general conditions in which the combination of the sparse and order encodings pays off, i.e., when the overhead incurred by maintaining both encodings is compensated by the pruning achieved, resulting in better runtimes.

The paper is organized as follows. Firstly, we briefly present the strengths and drawbacks of the sparse and order encodings. Secondly, we propose the *Sp-Or encoding* which combines them to encode CSPs dominated by constraints of the form $X \pm c \ \ \triangleright \ \ Y$. Thirdly, we justify our proposal through a number of experiments. Finally, we present conclusions and future works.

Background

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

Constraint Satisfaction Problem (CSP)

A *constraint satisfaction problem* is a triple $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, where
$\mathcal{V} := \{V_1, ..., V_k\}$ is a finite set of variables,
$\mathcal{D} := \{D(V_1), ..., D(V_k)\}$ is a finite set of domains,
$\mathcal{C} := \{C_1, ..., C_m\}$ is a finite set of constraints,
and $k, m \in \mathbb{N}$. A tuple $\langle v_1, ..., v_k\rangle \in \langle D(V_1), ..., D(V_k)\rangle$ *satisfies* a constraint $C_i, 1 \leq i \leq m$, defined on the set of variables $\{V_{i1}, ..., V_{il}\} \in \mathcal{V}$ if the projection of the tuple into these variables is a member of the constraint, i.e., if $\langle v_{i1}, ..., v_{il}\rangle \in C_i$. A tuple $\langle v_1, ..., v_k\rangle$ is a *solution* to a CSP iff it satisfies all constraints in $\mathcal{C}$. The *CSP problem* is to determine whether there exists one such tuple.

A CSP consisting of a finite set of variables and a finite domain for each variable is called a *finite CSP*. In this paper, we restrict our attention to *binary finite CSPs*, i.e. finite CSPs where all constraints are between two variables. Furthermore and without loss of generality we assume that all variables have domain $\{1, ..., n\}$.

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, where $m, n \in \mathbb{N}$. To each variable the truth values *false* (or 0) or *true* (or 1) can be assigned. A *clause* C is a finite disjunction of literals $l_1 \vee l_2 \vee ... \vee l_k$. A *literal* l is either a Boolean variable x_i or its negation $\neg x_i, 1 \leq i \leq n$, being denoted as a *positive* or *negative literal*, respectively.

A clause is *satisfied* by a truth assignment of the variables, if at least one of its literals is assigned value *true*. A formula $\mathcal{F}$ is *satisfiable* if there is a truth assignment that satisfies all its clauses, *unsatisfiable* otherwise. The *SAT problem* is to determine whether a given formula $\mathcal{F}$ is *satisfiable*.

Translating a Finite CSP as a SAT

In order to translate a CSP to a SAT instance, one must encode finite domain variables onto propositional variables and constraints among the variables onto a SAT formula.

Translating a Finite Domain We briefly present the two main encodings, viz. the sparse and the order encodings.

The Sparse Encoding The term "sparse" was first used in (Hoos 1999) to refer to the encoding that was introduced by Kleer (De Kleer 1989). It is also called the *direct encoding* (Walsh 2000). This method is the most straightforward way to transform a CSP into a SAT instance. Here, we follow Jeavons and Petke (Jeavons and Petke 2012).

To encode a CSP variable V with domain $\{1, ..., n\}$, the sparse encoding uses n propositional variables $d_i^V, 1 \leq i \leq n$, 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 the *at-most-one* (AMO) clauses (cf. later).

Whereas many proposals share the same way of mapping each finite domain value into SAT, they differ on the way of translating CSP constraints (Walsh 2000; Gent 2002; Velev 2007), several of them are omitting the AMO clauses (Selman, Levesque, and Mitchell 1992; Velev 2007), thus loosing the equivalence between SAT and CSP solutions.

The Order Encoding The *order encoding* (Bailleux and Boufkhad 2003; Ansótegui and Manyà 2004; Crawford and Baker 1994) (reformulated as the *sequential counter encoding* in (Hölldobler and Nguyen 2013)) 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 by $[0, 0, 0, 0]$, $[1, 1, 0, 0]$, 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 following desired properties (Silva and Lynce 2007):

- 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 $o_{i-1}^V = o_i^V$ (Metodi and Codish 2012); to cope with V = 1, we assume an extra bounding variable $o_0^V = 1$.

According to (Bailleux and Boufkhad 2003) 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 $o_j^V = 0$.

Translating a Constraint A finite constraint can be encoded by two main methods: conflict or support. The first uses *conflict clauses* to specify the disallowed variable assignments in the original constraint, whereas the second uses *support clauses* to specify the allowed variable assignments (see (Gent 2002)). We present these methods in the context of sparse encoding; the order encoding is similar.

The Conflict Clause The *direct encoding* (Walsh 2000) maps the CSP constraints onto a set of *conflict clauses*, modeling the disallowed variable assignments.

Adopting notions and notations from (Prestwich 2009), 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* (Gent 2002) 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 constraints. Then, the support clauses that model the constraint are expressed as:

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

As Gent (Gent 2002) pointed out, the support clause can be derived from the conflict clause by binary resolution.

The *Sp-Or* Encoding

We now focus on encoding finite binary constraints of the form $X \pm c \rhd Y$, where $\rhd$ is a relational operator in $\{=, \neq, >, \geq, <, \leq\}$. Without loss of generality and to simplify the notation we restrict $\pm$ to be $+$ and c to be a positive integer.

- In the spirit of its semantics (a "negation "), a disequality constraint of the form $X + c \neq Y$ may be modelled by a set of conflict clauses

$$\bigwedge_{i=1}^{n-c} \neg(X = i \wedge Y = i + c)$$

  – encoded in the sparse encoding as:

$$\bigwedge_{i=1}^{n-c} (\neg d_i^X \vee \neg d_{i+c}^Y)$$

  – and in the order encoding as

$$\bigwedge_{i=1}^{n-c} (\neg o_{i-1}^X \vee o_i^X \vee \neg o_{i+c-1}^Y \vee o_{i+c}^Y).$$

- In contrast, an equality constraint of the form $X + c = Y$ is modeled as:

$$\bigwedge_{i=1}^{n-c} (X = i \leftrightarrow Y = c + i)$$

  together with clauses to tighten the domain bounds, either on variable X ($X \leq n-c$) or on variable Y ($Y > c$). The constraint is thus encoded as

  – in the sparse encoding (together with bounding clauses $\bigwedge_{i=1}^{c} \neg d_{n-c+i}^X$):

$$\bigwedge_{i=1}^{n-c} ((\neg d_i^X \vee d_{i+c}^Y) \wedge (d_i^X \vee \neg d_{i+c}^Y)$$

  – in the order encoding, adapting the optimization proposed in (Argelich et al. 2010) that equates the order vectors:

$$\bigwedge_{i=1}^{n-c} ((\neg o_i^X \vee o_{i+c}^Y) \wedge (o_i^X \vee \neg o_{i+c}^Y))$$

  together with the bounding conditions $\neg o_{n-c}^X$ and o_c^Y.

We illustrate the representation of inequality CSP constraints with constraint of the form $X + c \leq Y$ (the other operators $\{>, \geq, <, \}$ are similar).

- In the sparse encoding and assuming support clauses (conflict clauses would be similar), the range of Y values compatible with $X = i$ are modelled by:

$$\bigwedge_{i=1}^{n-c} (X = i \rightarrow \bigvee_{j=c}^{n-i} Y = i + j) \Longleftrightarrow \bigwedge_{i=1}^{n-c} (\neg d_i^X \vee \bigvee_{j=c}^{n-i} d_{i+j}^Y)$$

- For the order encoding the range of Y values may be represented directly:

$$\bigwedge_{i=1}^{n-c} (X = i \rightarrow Y \geq i + c)$$
$$\Longleftrightarrow \bigwedge_{i=1}^{n-c} (\neg o_{i-1}^X \vee o_i^X \vee o_{i+c-1}^Y)$$

  together with the bounding condition o_c^Y.

Analyzing the size of the clauses required by the different encodings one may conjecture that the sparse encoding outperforms the order encoding in most problems in which equalities and disequalities are the dominating constraints, whereas the order encoding outperforms the sparse encoding in problems in which inequalities are the dominating constraints.

For the above conjecture, we have not considered the AMO constraints required by the sparse encoding. These may be naturally modelled by the pairwise encoding

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

which requires $\frac{n(n-1)}{2}$ binary clauses, but there are several efficient ways which only need $O(n)$ clauses (Gent and Nightingale 2004; Sinz 2005; Frisch and Giannoros 2010; Hölldobler and Nguyen 2013), all requiring less clauses but additional auxiliary variables. In all these proposals, besides encoding the AMO constraint, these auxiliary variables do not get involved in other clauses.

Hence, rather than simply selecting either the sparse or the order encodings to encode a specific CSP, we conjecture that combining them may produce better results than adopting one of them in isolation. To implement such combination and use the most adequate propositional variables in the representation of CSP constraints there must be additional *bridging* clauses between the two representations of every CSP variable X, namely the following set of clauses:

$$\begin{array}{rcl} \mathit{Bridge} & := & \bigwedge_{i=1}^{n}(d_i^X \leftrightarrow o_{i-1}^X \wedge \neg o_i^X) \\ & \equiv & \bigwedge_{i=1}^{n}((\neg d_i^X \vee o_{i-1}^X) \\ & & \quad \wedge\ (\neg d_i^X \vee \neg o_i^X) \\ & & \quad \wedge\ (d_i^X \vee \neg o_{i-1}^X \vee o_i^X)) \end{array}$$

In the context of this paper, the *Sp-Or encoding* represents a CSP composed of equality and inequality constraints by the SAT formula

$$\mathcal{F}_{\mathit{Sp\text{-}Or}} := \mathit{Axiom} \wedge \mathit{Bridge} \wedge \mathit{Eqlt} \wedge \mathit{Ineq},$$

where

- *Axiom* is the conjunction of the clauses that represent the vector of order variables o;
- *Bridge* are the clauses that bridge the sparse and the order variables (d and o, resp.);
- *Eqlt* are the clauses that represent all CSP equalities and disequalities expressed both on the sparse variables and order variables (d and o resp.);
- *Ineq* are the clauses that represent all CSP inequalities expressed both on the sparse variables and order variables (d and o resp.).

In summary, we expect the $\mathcal{F}_{\mathit{Sp\text{-}Or}}$ to take advantage of the following features:

- the variables required to encode the CSP are the same required by the sparse encoding using the sequential encoding to translate the AMO constraint.
- there is no cost for ALO and AMO constraints in the sparse encoding;
- the redundant representation of constraints will improve the pruning obtained with each encoding in isolation.

Alternatively, we will also consider a trimmed encoding *Sp-Or*$_{opt}$ that aims at taking advantage of the most adequate encodings for the different types of constraints. It represents a CSP by the SAT formula

$$\mathcal{F}_{\mathit{Sp\text{-}Or}_{opt}} := Axiom \wedge Bridge \wedge Eqlt_{sparse} \wedge Ineq_{order},$$

where

- $Eqlt_{sparse}$ are the clauses that represent all CSP equalities and disequalities, *solely* on the sparse variables d;
- $Ineq_{order}$ are the clauses that represent all CSP inequalities, *solely* on the order variables o.

If the CSP includes other constraints, these may be modeled either with the sparse or the order variables, but these are out of the scope of this paper (in our experiments all problems only had equalities, disequalities and inequalities).

Experiments

To understand the importance of the encodings in different conditions, several experiments were conducted on various CSP problems that present a different mix of *equality*, *disequality* and *inequality* constraints.

The results presented in this section were obtained with a 2.66 Ghz, Intel Core 2 Quad processor with 3.8 GB of memory, under Ubuntu 10.04. Runtimes reported are in seconds. The used solver are *Clasp* (Gebser, Kaufmann, and Schaub 2009) (*clasp2.1.3x86_64linux* version) and *Lingeling* (Biere 2013) (*aqw* version) state-of-the-art Conflict-Driven Clause Learning SAT solvers, which were ranked the first on Application and Craft benchmarks in different categories at recent SAT competitions.[1]

In the tables presented below, the columns *Spa* and *Ord* refer to the sparse and order encoding, resp. Columns *SpOr* and *SpOr*$_{opt}$ refer to the corresponding encodings. All runtimes reported are in seconds.

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 the minimum makespan to complete all jobs. Tamura et al. solved many benchmarks successfully with the order encoding (Tamura et al. 2009). The benchmarks were generated by the Taillard's method (Taillard 1993). 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. For the AMO constraints required by the sparse encoding we used the *bimander encoding* introduced in (Hölldobler and Nguyen 2013).

[1]http://www.satcompetition.org

Inst	*M*	*S*	*Clasp*			*Lingeling*		
			Spa	*Ord*	*SpOr*	*Spa*	*Ord*	*SpOr*
4_4	249	U	42.80	**0.13**	0.33	23.03	**0.25**	0.46
	250	S	44.13	**0.14**	0.47	27.11	**0.21**	0.67
4_5	294	U	77.12	**0.12**	1.62	41.87	**0.34**	1.01
	295	S	78.51	**0.14**	1.55	42.16	**0.28**	1.27
5_1	299	U	104.26	**0.62**	2.38	192.11	**1.07**	6.73
	300	S	101.72	**0.57**	4.02	185.54	**0.40**	2.49
5_2	261	U	130.90	**0.83**	2.15	132.14	**1.02**	5.40
	262	S	129.55	**0.71**	4.32	106.02	**0.61**	2.91

Table 1: The running time comparison of different encodings performed by *Clasp* and *Lingeling* on open shop scheduling instances. *M* is the makespan used. S/U indicates the satisficability or unsatisfiability of instances.

These instances of the open shop scheduling problem show the order encoding to be more efficient than either the sparse or any of the redundant Sp-Or encodings. Our experimental results with the *two-dimensional strip packing problem* (Soh et al. 2010) (not shown) also confirm that, in CSP problems where inequality constraints are dominant, the order encoding dramatically outperforms the sparse encoding but less so regarding the *Sp-Or* encoding.

The Quasigroup With Holes Problem

We now turn to CSP problems where disequalities are dominant. A quasigroup is a square of values $x_{ij}, 1 \leq i, j \leq n$ where each number $[1..n]$ occurs exactly once in each row and column. Achlioptas et al. (Achlioptas et al. 2000) introduced a method for generating satisfiable quasigroup with holes (QWH) instances in which some of the x_{ij} are given. QWH instances can be considered as a multiple permutation problem in which the variables may occur in more than one permutation problem. In a permutation the dominating constraints are disequalities of type $X \neq Y$ and, more specifically, they have some structure in the form of all-different constraints (on the rows and columns of the quasigroup). We experimented with QWH instances of different levels of hardness, and for the sparse encoding we used the *bimander encoding* introduced by (Hölldobler and Nguyen 2013).

Inst	*Clasp*			*Lingeling*		
	Spa	*Ord*	*SpOr*	*Spa*	*Ord*	*SpOr*
order10.holes100	0.0	0.1	**0.0**	0.0	0.1	**0.0**
order18.holes120	0.0	0.1	**0.0**	0.0	0.1	**0.0**
order20.holes400	0.0	0.1	**0.0**	0.0	0.1	**0.0**
order30.holes320	**0.2**	0.7	0.6	0.2	0.8	**0.1**
order30.holes316	**0.2**	1.7	4.9	**0.1**	2.0	0.2
order30.holes900	**0.7**	6.1	2.0	**2.2**	2.5	3.1
order33.holes381	79.7	>7200	**59.6**	435.8	6891.0	**110.3**
order35.holes405	**3.5**	858.7	71.2	10.1	736.9	**8.5**
order40.holes528	**103.9**	>7200	308.1	1220.9	>7200	**486.6**
order40.holes544	**100.8**	>7200	119.1	1624.0	>7200	**1516.4**
order40.holes560	**21.2**	3,757.9	174.6	134.5	>7200	**65.5**
order40.holes1600	9.9	2,317.1	**2.4**	**4.0**	6.4	6.1

Table 2: The running time comparison of encodings performed by *Clasp* and *Lingeling* on satisfiable QWH instances.

Inst	*K*	*Clasp*			*Lingeling*		
		Spa	*Ord*	*SpOr*	*Spa*	*Ord*	*SpOr*
1-Insertion_4	4	17.2	34.6	**11.7**	**30.0**	68.5	93.0
	5	0.5	2.0	**0.4**	0.0	0.1	0.0
2-FullIns_5	6	22.5	19.6	15.9	64.6	22.4	19.8
	7	**0.3**	0.6	0.5	**0.1**	0.3	0.3
4-Fullins_4	7	5.9	1.8	**1.3**	12.5	3.0	**2.8**
	8	0.2	0.3	**0.1**	**0.1**	0.2	0.2
anna	10	13.5	1.1	**1.0**	142.3	1.1	**0.9**
	11	0.1	0.1	**0.1**	0.1	0.1	**0.1**
huck	10	6.8	1.2	**1.1**	119.4	1.1	**0.9**
	11	0.0	0.0	**0.0**	0.1	0.1	**0.1**
miles500	10	51.2	**3.7**	4.1	1566.4	3.8	**3.0**
	11	1458.3	5.2	**4.0**	7200	5.8	**5.1**
miles750	10	109.5	**1.2**	2.5	2034.8	**1.9**	2.1
	11	2535.5	7.9	**6.4**	6534.4	7.1	**7.0**
miles1000	10	179.8	1.8	**0.9**	2.7	**2.4**	3.0
	11	3908.2	8.6	**7.5**	2375.8	8.4	**8.1**
miles1500	10	254.0	**1.5**	1.9	0.1	**4.2**	5.3
	11	4632.8	6.8	**5.3**	**4.2**	14.8	10.3
queen12_12	10	16.9	1.0	**0.7**	717.2	**1.3**	2.1
	11	91.3	**2.6**	2.8	2164.3	2.8	**2.3**
queen13_13	10	35.9	1.5	**1.3**	485.7	**1.8**	3.2
	11	275.2	**3.8**	4.7	>7200	4.1	**3.9**
queen14_14	10	75.6	1.4	**1.2**	689.2	**3.1**	4.2
	11	679.7	6.0	**5.4**	>7200	7.0	**6.2**

Table 3: The running time comparison of different encodings performed by *Clasp* and *Lingeling* on graph colouring instances. K is the number of colors used.

Table 2 presents the runtimes for finding the first solution of satisfiable QWHs instances of different sizes. They show that, the sparse encoding is far better than the order encoding, as would be expected since this encoding was proposed for interval variables, subject to CSP inequality constraints that do not exist in this case. Generally, the Sp-Or encoding produces similar results to the sparse encoding. Specifically, the redundancy results in a more significant slow down of the run time by *Clasp*, whereas its performance is clearly better than the sparse encoding by *Lingeling*.

These findings were corroborated in the *Hamiltonian cycle problem (HCP)*, which has similar characteristics as the QWH problem in that the dominating CSP constraints are disequalities $X \neq Y$, most of them structured as all-different constraints.

The Graph Colouring Problem

The *graph coloring problem* is a well known NP-complete problem (Garey and Johnson 1979) which is the problem of finding a minimum number of colors for the vertices of an undirected graph, such that no two adjacent vertices share the same color. We experimented with widely-used hard unsatisfiable instances, obtained from different generators.[2] For the first six benchmarks we got optimal solutions (the instance with lowest K is unsatisfiable and the other is satisfiable), and the remaining benchmarks are unsatisfiable. The results obtained are shown in Table 3, adopting the sequential counter encoding (Sinz 2005) for the AMO constraints.

[2]http:// mat.gsia.cmu.edu/COLOR/instances.html.

Although there are no inequality constraints in these problems, the order encoding produces results that are better than those obtained with the sparse encoding. There are two possible causes for this ”unexpected” behavior. A possible reason is the smaller size of the domains (vertices may have only about 10 colors, whereas in the previous problems the domains were larger). Another possible explanation is based on the structure of the disequality CSP constraints. Whereas in the former problems these constraints were structured as all-different constraints, in the graph coloring this is not the case. Although a more comprehensive study needs to be done, we favor this justification much in line with the engineering of BEE, where the order encoding is adopted for modeling bounds consistency in the CP solver (hence CSP inequalities) but sparse encodings were also used when all-different constraints need to be represented (Metodi and Codish 2012). More interestingly, the Sp-Or encoding now produces better results than those obtained with the order encoding for most of the instances. Hence, we expect that in problems where CSP disequality constraints are dominant (wrt to inequalities) and they are not structured as all different constraints, the overhead incurred by maintaining d and o variables and expressing redundantly the disequality constraints in both representations is more than compensated by the superior pruning obtained, resulting in significant execution speed ups.

The Pigeon-Hole Problem

The goal of this problem is to prove that *p* pigeons can not fit in $h = p - 1$ holes, which results in hard SAT encodings. In the problems tested the domains are small (10 to 15 holes) which given the discussion above should favor the order encoding. However, the disequality CSP constraints are organized as an all-different constraint, which favors the sparse encoding. The results obtained with these encodings shown in Table 4 reflect this trade off, and the runtimes with both encodings are quite similar (again, the bimander encoding was used to model AMO with the sparse encoding). But the results in Table 4 also show that the Sp-Or encoding outperforms both the other encodings on all instances, suggesting that in these conditions the redundant representation of domains and constraints significantly pays-off.

Inst	*Clasp*			*Lingeling*		
	Spa	*Ord*	*SpOr*	*Spa*	*Ord*	*SpOr*
10	0.2	0.3	**0.2**	1.1	0.1	**0.1**
11	2.1	3.3	**2.0**	3.4	0.6	**0.4**
12	26.0	13.2	**10.6**	12.8	1.6	**1.2**
13	64.9	72.0	**64.3**	31.5	8.7	**5.1**
14	560.0	1,013.4	**353.9**	150.0	21.3	**17.3**
15	6918.5	7,394.6	**2,706.3**	646.3	116.6	**102.5**

Table 4: The running time comparison of encodings performed by *Clasp* and *Lingeling* on unsatisfiable Pigeon-Hole instances.

Solvers	*Problem*	*Inst*	*Spa*	*Ord*	$SpOr_{opt}$	*SpOr*
Clasp	Round Robin	8	0.4	0.4	**0.3**	0.4
		10	1.8	1.0	**0.8**	0.9
		12	115.0	54.6	41.5	**38.8**
		14	>7200	1,398.4	709.0	**651.2**
	Golomb Ruler	G(9,44)	0.91	0.3	1.1	**0.3**
		G(10,55)	2.5	2.51	9.4	**1.8**
		G(11,72)	**6.7**	101.67	23.2	47.4
		G(12,85)	409.53	1217.28	442.2	**141.2**
Lingeling	Round Robin	8	0.7	**0.1**	1.4	0.5
		10	**1.4**	3.9	2.2	1.0
		12	**19.2**	79.8	43.3	64.8
		14	3472.7	>7200	**454.6**	2104.3
	Golomb Ruler	G(9,44)	0.6	**0.4**	3.9	2.7
		G(10,55)	9.7	**1.1**	8.0	14.9
		G(11,72)	109.5	137.1	32.4	**24.5**
		G(12,85)	568.0	702.5	152.7	**86.5**

Table 5: The running time comparison of encodings performed by *Clasp* and *Lingeling* on Round Robin Scheduling and Golomb Ruler instances. The column *Inst* shows the number of teams.

The Round Robin and Golomb Ruler Problems

The *problem of scheduling a tournament* (CSPLIB prob026 in (Hnich et al. 2013)) in which every two teams play each other exactly once, and each game is scheduled on a certain field at a certain week has been studied for a long time in the operations research and computer science communities since it is a challenging and real-life application. The problem requires the consideration of both disequality and inequality CSP constraints. Similar mix of constraints appear in the *Golomb Ruler problem*, aiming at finding an increasing sequence of numbers where all differences between them are different (CSPLIB prob006 in (Hnich et al. 2013)).[3] In both problems we used the sequential counter encoding for the AMO constraints in the sparse encoding.

The results in Table 5 show a similar trade off between the order and the sparse encodings. The small size of the domains and the existence of inequality constraints favor the former, whereas the all-different constraints favor the latter, although less so. But once again a combination of both pays off, as shown in the table for the Sp-Or encoding. Given the mix of disequality and inequality constraints of this problem, we tried the $Sp\text{-}Or_{opt}$ encoding, that explores less redundancy (as explained). The results show that in *Clasp* the redundant representation significantly improves propagation and the pruning of the search space, specially in larger instances, whereas with *Lingeling* the benefits of redundancy are not so clear and the $Sp\text{-}Or_{opt}$ encoding might be preferable to the *Sp-Or* encoding.

[3]In fact both these problem requires non-binary CSP constraints such as $A + B \rhd C + D$ but these can be transformed in binary constraint constraints $X \rhd Y$ after introducing ternary equality constraints such as $X = A + B$, which requires a straightforward generalization of the techniques described above. A similar transformation with the introduction of additional variables was done for constraints such $A_1 + A_2 + \cdots + A_m \rhd c$.

Conclusion

In this paper we investigate the efficiency of the two most widely used variable SAT encodings, the sparse and the order encodings, when transforming a CSP into a SAT instance, and propose their combination in certain types of CSPs. In general, CSP problems are quite diverse, and so we focussed in common binary CSP problems with the usual relational operators. Our experiments provide some hints on the most adequate encodings for a variety of CSPs.

Not surprisingly, we noticed that if inequalities are the dominant CSP constraints, the order encoding should be used. In fact, this encoding is specially tailored to represent interval variables, and in this case, the CSP constraints only affect the variable domain bounds.

However, if variable domains are large and the dominant CSP constraints are disequalities structured as all-different constraints, the sparse encoding yields SAT encodings with shorter runtimes.

The results reported also confirm the advantages of our proposed Sp-Or encoding, inspired by the redundant modelling commonly adopted in Constraint Programming, namely when the CSP problems include disequality constraints which are not structured as all-different constraints, with small to medium domains, possibly mixed with inequality constraints. The encoding *redundantly* combines the propositional variables and constraints used in both the sparse and order encodings, and despite the overhead of maintaining both representations this redundancy pays off in execution time, due to the better pruning achieved by the SAT solvers.

Interestingly enough, even when either the sparse or the order encoding in isolation yield the fastest runtimes, the overhead presented by the Sp-Or encoding with respect to the best encoding is never very significant, making it a fairly *robust* encoding, able to produce fast SAT encodings without sophisticated optimization techniques.

These encoding guidelines should be more thoroughly tested, namely with other CSP problems and SAT solvers, together with the identification of structural features of relevance in these problems. In particular we are interested to investigate robustness of the proposed Sp-Or encoding with CSP problems including non-linear constraints and more general non-binary constraints (e.g. table constraints). Finally, we plan to apply this encoding to the second level of the *log-direct* and *log-order* encodings proposed by (Nguyen, Velev, and Barahona 2013) that adopt the order and the sparse encodings, respectively, for their second level (the first level of both uses the *log encoding* (Iwama and Miyazaki 1994; Walsh 2000).

Acknowledgment

We are grateful to Amit Metodi for his fruitful supports, and to Inês Lynce for useful suggestions. We thank Carla Gomes for her kindly providing the QWH instances.

References

Achlioptas, D.; Gomes, C. P.; Kautz, H. A.; and Selman, B. 2000. Generating Satisfiable Problem Instances. In Kautz, H. A., and Porter, B. W., eds., *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence*, 256–261. AAAI Press / The MIT Press.

Ansótegui, C., and Manyà, F. 2004. Mapping Problems with Finite-Domain Variables into Problems with Boolean Variables. In Hoos, H. H., and Mitchell, D. G., eds., *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, volume 3542 of *LNCS*, 1–15. Springer-Verlag.

Argelich, J.; Cabiscol, A.; Lynce, I.; and Manyà, F. 2010. New Insights into Encodings from MaxCSP into Partial MaxSAT. In *40th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2010*, 46–52. IEEE Computer Society.

Bailleux, O., and Boufkhad, Y. 2003. Efficient CNF Encoding of Boolean Cardinality Constraints. In Francesca, R., ed., *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *LNCS*. Springer.

Berre, D. L., and Lynce, I. 2008. CSP2SAT4J: A Simple CSP to SAT Translator. In van Dongen, M.; Lecotre, C.; and Rossel, O., eds., *Proceedings of the Second International CSP Solver Competition - CP 2006, 12th International Conference*.

Biere, A. 2013. Lingeling, Plingeling and Treengeling Entering the SAT Competition 2013. In Adrian Balint, A. B.; Heule, M.; and Järvisalo, M., eds., *Proceedings of SAT Competition 2013*, 51–52. Department of Computer Science Series of Publications B , vol. B-2013-1 , University of Helsinki , Helsinki.

Cadoli, M., and Schaerf, A. 2005. Compiling Problem Specifications into SAT. *Artificial Intelligence* 162(1-2):89–120.

Cheng, B. M. W.; Lee, J. H.-M.; and Wu, J. C. K. 1996. Speeding Up Constraint Propagation By Redundant Modeling. In Freuder, E. C., ed., *Principles and Practice of Constraint Programming - CP1996, 2nd International Conference*, volume 1118 of *LNCS*, 91–103. Springer.

Crawford, J. M., and Baker, A. B. 1994. Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems. In Hayes-Roth, B., and Korf, R. E., eds., *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, Volume 2*, 1092–1097. AAAI Press / The MIT Press.

De Kleer, J. 1989. A Comparison of ATMS and CSP Techniques. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'89, 290–296. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Frisch, A. M., and Giannoros, P. A. 2010. SAT Encodings of the At-Most-k Constraint. Some Old, Some New, Some Fast, Some Slow. In *Proc. of the Ninth Int. Workshop of*

Constraint Modelling and Reformulation - CP2010, St. Andrews, Scotland (UK).

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Gavanelli, M. The Log-support Encoding of CSP into SAT. In Bessiere, C., ed., *Principles and Practice of Constraint Programming - CP2007, 13h International Conference, Proceedings*, volume 4741 of *LNCS*, 815–822. Springer.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2009. The Conflict-Driven Answer Set Solver clasp: Progress Report. In Erdem, E.; Lin, F.; and Schaub, T., eds., *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009*, volume 5753 of *LNCS*, 509–514. Springer.

Gent, I., and Nightingale, P. 2004. A New Encoding of AllDifferent into SAT. In Frisch, A. M., and Miguel, I., eds., *Proceedings 3rd International Workshop on Modelling and Reformulating Constraint Satisfaction Problems - CP2004, Toronto, Canada*, 95–110.

Gent, I. P. 2002. Arc Consistency in SAT. In van Harmelen, F., ed., *Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI'2002*, 121–125. IOS Press.

Hnich, B.; Miguel, I.; Gent, I. P.; ; and Walsh, T. 2013. CSPLib: A Problem Library for Constraints. `http://www.csplib.org/`.

Hölldobler, S., and Nguyen, V. H. 2013. On SAT-Encodings of the At-Most-One Constraint. In Katsirelos, G., and Quimper, C.-G., eds., *Proc. The Twelfth International Workshop on Constraint Modelling and Reformulation - CP2013, Uppsala, Sweden*, 1–17.

Hölldobler, S.; Manthey, N.; Nguyen, V.; and Steinke, P. 2012. Solving Hidokus Using SAT Solvers. In *Proc. INFOCOM-5*, 208–212. ISSN 2219-293X.

Hoos, H. H. 1999. SAT-Encodings, Search Space Structure, and Local Search Performance. In Dean, T., ed., *Proceedings of the 16th International Joint Conference on Artificial Intelligence, IJCAI 99, 2 Volumes*, 296–302. Morgan Kaufmann.

Iwama, K., and Miyazaki, S. 1994. SAT-Variable Complexity of Hard Combinatorial Problems. In *In Proceedings of the World Computer Congress of the IFIP (Volume 1)*, 253–258. Elsevier Science B.V.

Jeavons, P., and Petke, J. 2012. Local Consistency and SAT-Solvers. *J. Artif. Intell. Res. (JAIR)* 43:329–351.

Kautz, H., and Selman, B. 2007. The State of SAT. *Discrete Applied Mathematics* 155:1514–1524.

Metodi, A., and Codish, M. 2012. Compiling Finite Domain Constraints to SAT with BEE. *Theory and Practice of Logic Programming* 12(4-5):465–483.

Nguyen, V. H.; Velev, M. N.; and Barahona, P. 2013. Application of Hierarchical Hybrid Encodings to Efficient Translation of CSPs to SAT. In *Proc. 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI2013), Special Track on SAT and CSP*, 1028–1035. Conference Publishing Services.

Ohrimenko, O.; Stuckey, P.; and Codish, M. 2009. Propagation via Lazy Clause Generation. *Constraints* 14(3):357–391.

Petke, J., and Jeavons, P. 2011. The Order Encoding: From Tractable CSP to Tractable SAT. In Sakallah, K. A., and Simon, L., eds., *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, Proceedings*, volume 6695 of *LNCS*. Springer.

Prestwich, S. D. 2004. Full Dynamic Substitutability by SAT Encoding. 3258.

Prestwich, S. D. 2009. *CNF Encodings*. IOS Press. chapter 2, 75–98.

Rossi, F.; Beek, P. v.; and Walsh, T. 2006. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. New York, NY, USA: Elsevier Science Inc.

Selman, B.; Levesque, H. J.; and Mitchell, D. G. 1992. A New Method for Solving Hard Satisfiability Problems. In Swartout, W. R., ed., *AAAI*, 440–446. AAAI Press / The MIT Press.

Silva, J. M., and Lynce, I. 2007. Towards Robust CNF Encodings of Cardinality Constraints . In *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, Proceedings*, volume 4741 of *LNCS*, 483–497. Springer.

Sinz, C. 2005. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, Proceedings*, volume 3709 of *LNCS*, 827–831. Springer.

Soh, T.; Inoue, K.; Tamura, N.; Banbara, M.; and Nabeshima, H. 2010. A SAT-based Method for Solving the Two-dimensional Strip Packing Problem. *Fundam. Inform.* 102(3-4):467–487.

Taillard, É. 1993. Benchmarks for Basic Scheduling Problems. *European Journal of Operational Research* 64(2):278 285.

Tamura, N.; Taga, A.; Kitagawa, S.; and Banbara, M. 2009. Compiling Finite Linear CSP into SAT. *Constraints* 14(2):254–272.

van Dongen, M.; Lecotre, C.; and Rossel, O. 2008. Third International CSP Solver Competition . `http://www.cril.univ-artois.fr/CPAI08/`.

van Dongen, M.; Lecotre, C.; and Rossel, O. 2009. Fourth International Constraint Solver Competition. `http://cpai.ucc.ie/09/`.

Velev, M. N. 2007. Exploiting Hierarchy and Structure to Efficiently Solve Graph Coloring as SAT. In *International Conference on Computer-Aided Design (ICCAD'07)*, 135–142. IEEE.

Walsh, T. 2000. SAT v CSP. In *Principles and Practice of Constraint Programming - CP2000, 6th International Conference*, volume 1894 of *LNCS*, 441–456. Springer.