Computing Science Group

THE ORDER ENCODING: FROM TRACTABLE CSP TO TRACTABLE SAT

Justyna Petke and Peter Jeavons

CS-RR-11-04

Oxford University, Department of Computer Science
Wolfson Building, Parks Road, Oxford OX1 3QD

The order encoding: from tractable CSP to tractable SAT*

Justyna Petke and Peter Jeavons

Abstract

Many mathematical and practical problems can be expressed as *constraint satisfaction problems* (CSPs). One way to solve a CSP instance is to encode it into SAT and use a SAT-solver. However, important information about the problem can get lost during the translation stage. For example, although the general constraint satisfaction problem is known to be NP-complete, there are some classes of CSP instances that have been shown to be tractable. These include the classes of CSP instances that contain only *max-closed* or *connected-row-convex* constraints. In this paper we show that translating such instances using some common standard encodings results in SAT instances which do not fall into known tractable classes. However, translating such instances using the *order encoding* results in SAT instances that do fall into known tractable classes. Moreover, modern clause learning SAT-solvers can then solve them efficiently. Hence, we give a theory-based argument to prefer the order encoding for certain families of constraint satisfaction problems.

*An extended abstract is published in *Theory and Applications of Satisfiability Testing - SAT2011.* Further copies of this Research Report may be obtained from the Librarian, Oxford University, Department of Computer Science, Computing Science Group, Wolfson Building, Parks Road, Oxford OX1 3QD, England (Telephone: +44-1865-273837, Email: `library@cs.ox.ac.uk`).

1 Introduction

SAT solvers are now considered to be efficient and practical tools for many kinds of problems [24]. In order to use them to solve constraint satisfaction problems (CSPs), various encodings from CSP to SAT have been developed [3, 20, 23]. In a recent paper we showed that current SAT-solvers will decide the satisfiability of the direct encoding of any CSP instance with *bounded width* in expected polynomial-time [18]. Here we investigate how SAT-solvers perform on CSP instances that are tractable due to the nature of the constraints.

SAT-based constraint solvers performed very well in recent CSP solver competitions [1, 2]. One solver in particular, called Sugar [22], was very efficient and even won in a few categories. Surprisingly, it outperformed standard constraint solvers on many instances involving global constraints, which are supposed to be a particular strength of CSP solvers.

The SAT encoding that the Sugar solver implements is called the *order encoding* [23]. In this paper we show that this encoding, unlike other common encodings, transforms instances of various tractable language classes of the constraint satisfaction problem into easily-recognised tractable instances of SAT. We also show that such instances can then be solved by standard SAT-solvers in polynomial-time.

2 Preliminaries

The (finite domain) constraint satisfaction problem is often defined as follows:

Definition 2.1. *An instance of the* **Constraint Satisfaction Problem** *(CSP) is specified by a triple* (V, D, C)*, where*

- V *is a finite set of* variables*;*
- D *is a finite set of possible* values *for the variables* $v \in V$*, called the domain;*
- C *is a finite set of* constraints*. Each constraint in* C *is a pair* (R_i, S_i) *where*
  - S_i *is an ordered list of* m_i *variables, called the constraint* scope*;*
  - R_i *is a relation over* D *of arity* m_i*, called the constraint* relation*.*

Given any CSP instance (V, D, C), a *partial assignment* is a mapping f from some subset W of V to D. A partial assignment *satisfies the constraints* of the instance if, for all $(R, (v_1, v_2, \ldots, v_m)) \in C$ such that $v_j \in W$ for $j = 1, 2, \ldots, m$, we have $(f(v_1), f(v_2) \ldots, f(v_m)) \in R$. A partial assignment that satisfies the constraints of an instance is called a *partial solution*[1] to that instance. The set of variables on which a partial assignment f is defined is called the domain of f, and denoted $Dom(f)$. A partial solution f' *extends* a partial solution f if $Dom(f') \supseteq Dom(f)$ and $f'(v) = f(v)$ for all $v \in Dom(f)$. A partial solution with domain V is called a solution.

[1]Note that not all partial solutions extend to solutions.

The general CSP is known to be NP-complete, but many different conditions have been identified which are sufficient to ensure that classes of instances satisfying those conditions are tractable, that is, solvable in polynomial-time [7, 8, 9, 10, 12, 17]. This research has focused on two main ways of imposing restrictions on a constraint problem to ensure that it can be tractably solved. The first of these is to consider restrictions on the way in which the constraints overlap - these are sometimes referred to as *structural* restrictions. In particular, if the hypergraph formed by the constraint scopes has a property known as bounded width, then it has been shown that the CSP instance can be solved in polynomial-time [14].

The second standard approach to identifying restrictions on constraint problems which ensure tractability is to restrict the forms of constraint which are allowed, that is, to restrict the choice of the constraint relations. These are sometimes known as constraint *language* restrictions, and we focus on these in this paper.

3 Some tractable CSP languages

3.1 The Boolean case: tractable languages for SAT

The propositional satisfiability problem (SAT) is actually a special type of constraint satisfaction problem where the domain is Boolean (i.e., it contains just two values, true and false), and the constraint relations are all specified by clauses.

To study possible language restrictions for SAT, we consider subproblems of the form SAT(C), where C is a set of relations over the Boolean domain, and SAT(C) consists of all SAT instances whose constraint relations belong to C. Schaefer's well-known dichotomy theorem [21] identifies all the tractable languages for SAT:

Theorem 3.1 ([21])**.** *Let C be a set of Boolean relations. If C satisfies at least one of the conditions below, then SAT(C) is tractable. Otherwise SAT(C) is NP-complete.*

1. *Every relation in C evaluates to $true$ if all arguments are $true$.*
2. *Every relation in C evaluates to $true$ if all arguments are $false$.*
3. *Every relation in C can be expressed as a Horn formula.*
4. *Every relation in C can be expressed as a dual-Horn formula.*
5. *Every relation in C can be expressed as a 2-CNF formula.*
6. *Every relation in C can be expressed as an affine formula.*

A formula is called *Horn* if it is a conjunction of clauses where every clause contains at most one positive literal. It is *dual-Horn*, if it is a conjunction of clauses where every clause has at most one negative literal. A *2-CNF formula* is a conjunction of clauses containing at most two literals each, and an *affine formula* is a conjunction of linear equations over the two-element field.

3.2 A trivial case: Constant-closed constraints

A CSP instance is *constant-closed* if every constraint in it allows the constant value d to be assigned to all variables in its scope, for some fixed d. Such instances are trivially

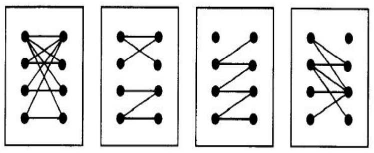

Figure 1: Examples of binary max-closed constraints from [16]. In each rectangle, the circles on the left represent possible values for one variable, and the circles on the right represent possible values for the other variable; both are ordered from bottom to top. Two circles are connected by a line if the constraint allows that combination of values for the two variables.

tractable, because they can be solved by the trivial linear algorithm that assigns the value d to every variable in the instance. Surprisingly, instances with this property do sometimes occur in practice: the majority of the satisfiable binary decision diagram instances used in the third CSP solver competition [1] were later found to have constant solutions.

3.3 Max-closed constraints

A rather more interesting family of tractable constraint satisfaction problems is the class of CSPs whose constraints are all *max-closed*, as defined below:

Definition 3.2 ([16]). *A constraint (R, S) with relation R of arity r over an ordered domain D is said to be* max-closed *if for all tuples $(d_1, \cdots, d_r), (d'_1, \cdots, d'_r) \in R$ we have that the tuple $(max(d_1, d'_1), \cdots, max(d_r, d'_r))$ is also in R. It is* min-closed *if for all such tuples the tuple $(min(d_1, d'_1), \cdots, min(d_r, d'_r))$ is in R.*

Some examples of binary max-closed constraints are shown in Figure 1.
Another broad class of max-closed constraints, of *arbitrary arity*, are constraints on numeric variables that can be represented by inequalities of the form:

$$\sum_{i=1}^{n-1} a_i v_i \geq a_n v_n + c \tag{1}$$

where the v_i's are variables, c is a constant, and the a_i's are non-negative constants [16].

Max-closed constraints can take many other forms. For example, all unary constraints on any ordered domain are max-closed, and certain non-linear numerical constraints, such as $6v_1v_3v_5 \geq 3v_2 + 2$ are max-closed. They all can be solved by a polynomial-time algorithm that enforces a type of local consistency known as arc consistency [7].

Theorem 3.3. *A CSP instance that contains only max-closed constraints can be solved by enforcing arc consistency.*

Proof. Let P be a CSP instance that contains only max-closed constraints. After enforcing arc consistency on P a value d is in the domain of variable v if and only if for every constraint C in P there exists a tuple t satisfying C such that $t[v] = d$. If the domain

$$\begin{pmatrix}
0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&1&1&1&0&0&0\\
0&0&0&1&1&1&1&0&1&0\\
0&1&1&1&1&1&1&0&1&0\\
1&1&1&1&1&1&1&0&1&1\\
0&1&1&1&1&1&1&0&1&0\\
0&0&1&1&1&1&1&0&1&0\\
0&0&1&1&1&1&1&0&1&0\\
0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0
\end{pmatrix}
\begin{pmatrix}
1&1&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&0&0\\
0&0&1&1&1&0&0&0&0&0\\
0&0&1&1&1&0&0&0&0&0\\
0&0&1&1&1&0&0&0&0&0\\
0&0&1&1&1&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1\\
0&0&0&0&0&1&1&1&1&1\\
0&0&0&0&0&1&1&1&0&0\\
0&0&0&0&0&1&1&1&0&0
\end{pmatrix}
\begin{pmatrix}
0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&1&1&1&0\\
0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0\\
1&1&1&1&0&0&0&0&0&0\\
0&1&1&1&0&0&0&0&0&0\\
0&0&1&1&0&0&0&0&0&0
\end{pmatrix}$$

Figure 2: Examples of connected row-convex relations from [8]

.

of any variable is empty, then the instance is unsatisfiable and we are done. Otherwise, let $max(v)$ be the maximum domain value in the arc consistent domain of some variable v. We claim that the tuple $t = (v_1 = max(v_1), v_2 = max(v_2), ..., v_n = max(v_n))$ solves P. Assume, for contradiction, that t is not a solution for P. Then there exists some constraint $C \in P$ that disallows t. Because all the variable domains are arc consistent, for each variable $v_i \in C$ there exists a satisfying tuple t_i such that $t_i[v_i] = max(v_i)$. Applying the max operator to all such tuples produces the tuple t' such that $t'[v_i] = max(v_i)$ for all $v_i \in C$. As C is still max-closed after enforcing arc consistency, t' satisfies C. As $variables(t') = variables(C)$ and $t' \subseteq t$, t satisfies C and we reach a contradiction. Hence, the satisfiability of a max-closed CSP instance P can be established by enforcing arc consistency.

$\square$

Corollary 3.4. *A CSP instance that contains only max-closed constraints can be solved in $O(ed^2)$ time, where e is the number of constraints and d is the domain size.*

3.4 Connected row-convex constraints

Connected row-convex constraints were first defined in [12] using a standard matrix representation of binary relations.

Definition 3.5 ([12])**.** *Let the domain D be the ordered set $\{d_1, d_2, \cdots, d_m\}$, where $d_1 < d_2 < \cdots < d_m$. A binary relation R over D can be represented by a $m \times m$ 0-1 matrix M, by setting $M_{ij} = 1$ if the relation contains the pair (d_i, d_j) and $M_{ij} = 0$ otherwise. A relation is said to be* connected row-convex *if the pattern of 1s in the matrix representation (after removing rows and columns containing only 0s) is connected along each row, along each column, and forms a connected 2-dimensional region (where some of the connections may be diagonal).*

Any binary constraint whose constraint relation is connected row-convex will be called a connected row-convex constraint; some examples are illustrated in Figure 2. It is convenient to also define all unary constraints to be connected row-convex, as they may be combined with binary connected row-convex constraints to obtain a larger tractable class of problems [8].

It has been known for some time that the class of connected row-convex constraints can be solved in polynomial-time by an algorithm that enforces a type of local consistency known as *path-consistency* [12]. Recently this result has been strengthened by considering another type of consistency known as *singleton arc consistency* [5].

Theorem 3.6 ([6]). *A CSP instance that contains only connected row-convex constraints can be solved by enforcing singleton arc consistency.*

Proof. All connected row-convex relations have a majority polymorphism [7]. Hence, they can be solved by a singleton arc consistency algorithm, by Theorem 26 in [6]. $\square$

Corollary 3.7. *A CSP instance that contains only connected row-convex constraints can be solved in $O(end^3)$ time, where e is the number of constraints, d is the domain size and n is the number of variables.*

4 Standard SAT encodings

A *sparse encoding* of a CSP instance introduces a new Boolean variable, $x^{=}_{v,a}$, for each possible variable assignment, $v = a$. Let P be a CSP instance containing a variable v with domain $\{1, 2, \ldots, d\}$. Its sparse encoding, F, will contain the following set of Boolean variables to represent v: $\{x^{=}_{v,1}, x^{=}_{v,2}, \cdots, x^{=}_{v,d}\}$. Moreover, F will contain clauses that force at least one of these $x^{=}_{v,i}$s to be satisfied, to ensure that v is assigned some value. This restriction is usually represented by the so-called *at-least-one* (ALO) clause, $x^{=}_{v,1} \vee \cdots \vee x^{=}_{v,d}$. Additionally, F must not allow both $x^{=}_{v,i}$ and $x^{=}_{v,j}$ to be true for any $i \neq j$, since a CSP variable can be assigned one value only. This requirement is usually captured by a set of negative binary clauses, $\neg x^{=}_{v,i} \vee \neg x^{=}_{v,j}$, for all $1 \leq i < j \leq d$. These are called the *at-most-one* (AMO) clauses. AMO and ALO clauses form together the so-called *exactly one* (EO) constraint.

Finally, clauses are added to represent the constraints, to rule out any assignment to the Boolean variables that violates a constraint. Examples of sparse encodings include the direct encoding and the support encoding [20].[2]

The *direct encoding* encodes the disallowed variable assignments - the so-called *conflicts* or *no-goods.* It generates a clause $\bigvee_{v \in S} \neg x_{vf(v)}$ for each partial assignment f that does *not* satisfy the constraint $(R, S) \in C$.

Another way of translating constraints into clauses is to encode the allowed variable assignments - the so-called *supports.* This has been used as the basis for an encoding of binary CSP instances, known as the *support encoding.* For each pair of variables v, w in the scope of some constraint, and each value $i \in D$, the support encoding will contain the clause $\neg x_{vi} \vee \bigvee_{j \in A} x_{wj}$, where $A \subseteq D$ is the set of values for the variable w which are compatible with the assignment $v = i$, according to the constraint.

Proposition 4.1. *No sparse encoding of a CSP instance with domain size > 2 belongs to a tractable language class of SAT.*

[2]AMO clauses can be omitted in the direct encoding of a CSP instance. However, we assume here that every sparse encoding will contain clauses that impose the at-most-one condition.

Proof. Let v be a CSP variable with domain $\{1, 2, \ldots, d\}$. A sparse encoding introduces d Boolean variables, $\{x^{=}_{v,1}, x^{=}_{v,2}, \cdots, x^{=}_{v,d}\}$, to represent possible assignments to v. The encoding must also include a suitable set of clauses to enforce the "exactly-one" (EO) constraint on these variables, as in any solution of the CSP instance exactly one of the $x^{=}_{v,i}$ must be assigned true.

However, the EO relation on $d > 2$ Boolean variables does not lie in any of the 6 tractable language classes for SAT [21]. Hence the clauses which encode the EO relation do not all fall into any one of these tractable language classes. $\square$

The *log encoding* introduces a Boolean variable for each *bit* in the value of a CSP variable. For instance, a variable with domain $\{0, 1, 2, 3\}$ will be encoded using two Boolean variables. The AMO and ALO clauses described above are not needed in this encoding, since every bit pattern represents a potential solution. However, unary constraints can be imposed to rule out certain bit patterns, and hence restrict the values of a variable to some subset of the domain. Once again, clauses are added to represent the constraints, by ruling out any assignment that violates a constraint.

Proposition 4.2. *The log encoding of any CSP instance with domain size* > 4 *containing certain unary constraints does not belong to any tractable language class of SAT.*

Proof. Let $\{x_{[1]}, \cdots, x_{[r]}\}$ be the Boolean variables representing a CSP variable v under the log encoding of some instance P. Assume P contains unary constraints that restrict v to values that are a power of 2. The clauses representing these constraints must allow precisely those assignments where exactly one of the $x_{[i]}$s is true. Hence the log encoding of P must enforce the EO constraint on $\{x_{[1]}, \ldots, x_{[r]}\}$. If the size of the domain of v is greater than 4, then $r > 2$ and the EO constraint does not fall into any of the six tractable language classes for SAT [21]. Hence the clauses which encode this unary constraint do not all fall into any one of these tractable language classes. $\square$

5 The order encoding

All of the SAT encodings described thus far use one or more Boolean variables to represent each possible assignment of a CSP variable, $v = a$. The authors of the *order encoding* took a different approach [23], also used in [3, 11]. In this encoding each Boolean variable represents a comparison, $v \leq c$.

Let P be a CSP instance containing a variable v with domain $\{1, 2, \ldots, d\}$. Its order encoding, F, will contain the following set of Boolean variables to represent v: $\{x^{\leq}_{v,1}, x^{\leq}_{v,2}, \cdots, x^{\leq}_{v,d-1}\}$, where each $x^{\leq}_{v,i}$ represents the comparison $v \leq i$. Moreover, F will contain clauses $\neg(x^{\leq}_{v,c-1}) \vee (x^{\leq}_{v,c})$, for $c = 2, 3, \ldots, d-1$, to ensure that these variables are consistently assigned.

Finally, clauses are added to represent the constraints, to rule out any assignment to the Boolean variables that violates a constraint. For example, if a unary constraint on variable v excludes the value i from its domain, then we add the clause $\neg(x^{\leq}_{v,i}) \vee (x^{\leq}_{v,i-1})$, to represent the constraint that $(v > i) \vee (v \leq i - 1)$. Note that an arbitrary unary constraint on a finite domain can be represented as a conjunction of such binary clauses.

Example 5.1. *The order encoding for linear inequality constraints over the integers is discussed in detail in [23].*

Let $P = (V, D, C)$ be a CSP instance, in which D is the set $\{1, 2, \ldots, d\}$, and assume that every constraint of P is of the form $\sum_{i=1}^{n} a_i v_i \leq c$, where a_is are non-zero integer constants, c is an integer constant, and the v_is are mutually distinct integer variables.

The order encoding of P, taking values in D, will be a conjunction of the following clauses, where each inequality of the form $v \leq j$ is represented by a Boolean variable, $x^{\leq}_{v,j}$:

- *for all $v_i \in V$ and $j \in D$ such that $2 \leq j \leq d-1$: $\neg(v_i \leq j-1) \vee (v_i \leq j)$*
- *for all constraints $\sum_{i=1}^{n} a_i v_i \leq c$:*

$$\bigwedge_{(b_1,\ldots,b_n)\in B} \bigvee_{i} \begin{cases} v_i \leq \lfloor \frac{b_i}{a_i} \rfloor & (a_i > 0) \\ \neg(v_i \leq \lceil \frac{b_i}{a_i} \rceil - 1) & (a_i < 0) \end{cases}$$

where B is the set of integer tuples defined by

$$\{(b_1, \ldots, b_n) \mid \sum_{i=1}^{n} b_i = c - n + 1, \bigwedge_{i=1}^{n} min(a_i v_i) - 1 \leq b_i \leq max(a_i v_i)\}$$

It turns out that under the order encoding, unlike the encodings considered earlier, certain tractable CSP classes are translated to tractable language classes of SAT.

Theorem 5.2. *If a CSP instance $P = (V, D, C)$ is constant-closed under the lowest or the highest domain value, then its order encoding will also be constant-closed.*

Proof. Let $P = (V, D, C)$ be a CSP instance defined on domain $D = \{1, \ldots, d\}$. If P allows value 1 to be assigned to every variable $v \in V$, its order encoding will be satisfied by assigning the value *True* to all the Boolean variables $x^{\leq}_{v,i}$. On the other hand, if P allows value d to be assigned to every variable $v \in V$, its order encoding will be satisfied by assigning the value *False* to all the Boolean variables $x^{\leq}_{v,i}$. Hence, the result follows. □

A similar result can be obtained for max-closed constraints, thanks to the following theorem and corollary:

Theorem 5.3 ([16])**.** *A constraint over an ordered domain is max-closed if and only if it is logically equivalent to a conjunction of disjunctions of the following form:*

$$(v_i < a_i) \vee (v_1 > b_1) \vee \cdots \vee (v_n > b_n).$$

where each v_i is a variable (not necessarily distinct), and the a_i, b_i are domain values.

Corollary 5.4 ([16])**.** *If the domain of the variables is $\{True, False\}$, with $False < True$, then a constraint is min-closed if and only if it is logically equivalent to a conjunction of Horn clauses.*

Theorem 5.5. *If a CSP instance P contains only max-closed constraints, then its order encoding will be min-closed.*

Proof. By Theorem 5.3, every max-closed constraint can be represented as a set of constraints of the form $(v_i < a_i) \vee (v_1 > b_1) \vee \cdots \vee (v_n > b_n)$. Hence, the order encoding of a max-closed CSP instance P will produce the following set of clauses:

- for all $v_i \in V$ and $c \in D$ such that $2 \leq c \leq d-1$: $\neg(v_i \leq c-1) \vee (v_i \leq c)$
- for all constraints $(v_i < a_i) \vee (v_1 > b_1) \vee \cdots \vee (v_n > b_n)$:
$$(v_i \leq a_i - 1) \vee \neg(v_1 \leq b_1) \vee \cdots \vee \neg(v_n \leq b_n).$$

As each inequality of the form $v \leq c$ is represented by a Boolean variable, $x^{\leq}_{v,c}$, we end up with a set of Horn clauses, which are min-closed by Corollary 5.4. □

A similar result can be obtained for connected row-convex constraints, thanks to the following theorem, which shows that they have a very simple representation in terms of disjunctions of inequalities.

Theorem 5.6 ([8])**.** *A constraint over an ordered domain is connected row-convex if and only if it is logically equivalent to a conjunction of disjunctions of the following forms:*

$$\begin{array}{c} v \leq d_i \vee w \leq d_j \\ v \leq d_i \vee w \geq d_j \\ v \geq d_i \vee w \leq d_j \\ v \geq d_i \vee w \geq d_j. \end{array} \tag{2}$$

where v and w are variables (not necessarily distinct), and d_i, d_j are domain values.

Corollary 5.7. *If the domain of the variables is $\{True, False\}$, then a constraint is connected row-convex if and only if it is logically equivalent to a conjunction of 2-CNF clauses over literals representing comparisons.*

Theorem 5.8. *If a CSP instance P contains only connected row-convex constraints, then its order encoding will also be connected row-convex.*

Proof. By Theorem 5.6, each connected row-convex constraint can be represented as a set of disjunctions of inequalities of the form (2). The order encoding of a CSP instance P that contains such constraints will produce the following clauses:

- for all $v_i \in V$ and $c \in D$ such that $2 \leq c \leq d-1$: $\neg(v_i \leq c-1) \vee (v_i \leq c)$
- for all constraints C:

$$\begin{array}{lll} v_i \leq d_i \vee v_j \leq d_j & if & C = v_i \leq d_i \vee v_j \leq d_j \\ v_i \leq d_i \vee \neg(v_j \leq d_j - 1) & if & C = v_i \leq d_i \vee v_j \geq d_j \\ \neg(v_i \leq d_i - 1) \vee v_j \leq d_j & if & C = v_i \geq d_i \vee v_j \leq d_j \\ \neg(v_i \leq d_i - 1) \vee \neg(v_j \leq d_j - 1) & if & C = v_i \geq d_i \vee v_j \geq d_j \end{array} \tag{3}$$

As each inequality of the form $v \leq c$ is represented by a Boolean variable, we end up with a set of binary clauses, which are connected row-convex by Corollary 5.7. □

6 Full regular encodings

We note that a sparse encoding introduces a Boolean variable $x^{=}_{v,a}$ representing each assignment $v = a$, but the order encoding introduces a Boolean variable $x^{\leq}_{v,c}$ representing each comparison $v \leq c$. In this section, we will consider another suggested type of encoding that uses a similar idea to the order encoding.

A full regular encoding [3] introduces a Boolean variable $x^{\geq}_{v,a}$ for each inequality of the form $v \geq a$. It can be based on any sparse encoding. For example, the full regular direct encoding is based on the direct encoding. Let $x^{=}_{v,i}$ be an assignment literal in a constraint clause of a sparse encoding; in a full regular encoding it is replaced with $x^{\geq}_{v,i} \wedge \neg x^{\geq}_{v,i+1}$. Similarly, each negative literal $\neg x^{=}_{v,i}$ is replaced with $x^{\geq}_{v,i+1} \vee \neg x^{\geq}_{v,i}$. However, as the next result indicates, once we make these substitutions on a set of support or conflict clauses the resultant SAT formula belongs to a tractable language class of SAT only in the rather trivial constant-closed cases.

Proposition 6.1. *The full regular direct encoding of any CSP instance P with non-unary constraints belongs to a tractable language class of SAT if and only if the constraints of P are constant-closed under the highest or lowest domain value. The same applies to the full regular support encoding.*

Proof. Let $\bigvee(\neg x^{=}_{v,i})$ be a conflict clause in the direct encoding of some non-unary constraint. Under the full regular direct encoding such a clause is translated into the following formula: $\bigvee(x^{\geq}_{v,i+1} \vee \neg x^{\geq}_{v,i})$, which does not belong to any tractable language class of SAT, except the constant-closed classes.

Let $\neg x^{=}_{v,j} \vee \bigvee x^{=}_{w,i}$ be a support clause in the support encoding of some non-unary constraint. Under the full regular support encoding such a clause is translated into a conjunction of clauses of the form $(x^{\geq}_{v,j+1} \vee \neg x^{\geq}_{v,j} \vee x^{\geq}_{w,i+1}) \wedge (x^{\geq}_{v,j+1} \vee \neg x^{\geq}_{v,j} \vee \neg x^{\geq}_{w,i})$, which does not belong to any tractable language class of SAT except the constant-closed classes.

If a CSP instance is closed under the highest or lowest domain value, a full regular direct encoding of such an instance will be constant-closed for the value False or True, respectively, but in all other cases there will be a constraint which is violated by assigning all variables the highest value, and a constraint which is violated by assigning all variables the lowest value. These constraints must be represented by clauses which are not all members of either of the constant-closed classes. $\square$

7 Performance of DPLL-based SAT-solvers on tractable CSPs

It has been shown experimentally in [23] that the order encoding gives better solver performance compared with the direct and support encodings on instances of the graph colouring and open-shop scheduling problems. Our theoretical results above suggest that the order encoding will also be a better choice of encoding than a sparse or log encoding for all CSP instances over the tractable constraint languages we have considered. In this section we investigate to what extent this is true in practice with a current SAT-solver.

```
(int v1 1 5)
(int v2 1 5)
(int v3 1 5)
(<= -7 (+ (mul 5 v1) (mul -7 v3) ) )
(<= -12 (+ (mul 10 v1) (mul -7 v2) (mul -10 v3) ) )
(<= -4 (+ (mul -8 v1) (mul 2 v2) (mul 4 v3) ) )
(<= 6 (+ (mul 5 v1) (mul 8 v2) (mul 2 v3) ) )
```

Figure 3: A constant-closed CSP instance in $*.csp$ format [22] that is satisfied by the assignment $(v1 = 1, v2 = 1, v3 = 1)$. For example, the fourth line of the instance represents the constraint $-7 \leq 5v_1 - 7v_3$.

Throughout this section when we refer to a clause-learning SAT solver we mean a *Conflict-Driven Clause Learning* SAT-solver that implements the *DPLL* algorithm and an *asserting learning scheme*, as defined in Chapter 4 of [20].

We generated various instances of the three tractable CSP classes discussed in this paper. All the instances generated are specified by 3 parameters: the number of variables, the maximum domain value, and the number of constraints. The domain for every variable in our instances is $1..d$, where d is the maximum domain value.

All the instances generated were then encoded using the direct, log and order encodings. For the order encoding we used Sugar's built-in CSP-to-SAT translator, but without the optimizations described in [22] that introduce new variables. For the other encodings we wrote custom translators.

We ran all the encoded instances on the state-of-the-art clause-learning SAT-solver, MiniSAT [19] version 2-070721-8. For all of the results, the times given are elapsed times on a Lenovo 3000 N200 laptop with an Intel Core 2 Duo processor running at 1.66GHz with 2GB of RAM. For each set of parameters we generated four instances and report the average timings.

7.1 Constant-closed constraints

Proposition 7.1. *If a CSP instance P allows the lowest (or highest) domain value to be assigned to all its variables, then a DPLL-based SAT-solver with an appropriate value order will decide the satisfiability of the order encoding of P in linear time.*

Proof. Each clause of the order encoding of a constant-closed constraint under the lowest domain value has at least one positive literal. Hence, a DPLL-based SAT-solver, that assigns the value $True$ first to each variable, will not need to backtrack, and hence will decide its satisfiability in linear time in the size of the instance. Similarly, the satisfiability of a constant-closed instance under the highest domain value will be decided by a DPLL-based SAT-solver with the opposite value order in linear time. $\square$

In order to generate instances that were constant-closed under the lowest (or highest) domain value, we generated random inequalities and then selected those that satisfied the required constant solution. An example is shown in Figure 3.

As predicted by Theorem 5.2, for all constraints that were constant-closed under the lowest domain value, the formulas generated by the Sugar solver contained at least one positive literal in each clause. Moreover, the instances that were satisfied by assigning the highest domain value to each variable contained at least one negative literal in each clause. The runtimes of MiniSAT on these two families of instances[3] are presented in Table 1.

number of CSP variables	number of CSP values	number of constraints	MiniSAT (sec.) direct encoding	MiniSAT (sec.) log encoding	MiniSAT (sec.) order encoding
instances satisfied by assigning the lowest domain value to every variable					
3	10	100	0.01	0.07	**0.00**
6	6	10	22.55	489.60	**0.10**
9	3	10	7.52	39.90	**0.08**
9	3	100	7.70	39.84	**1.76**
9	4	10	$>$ 16 min	284.37	**4.64**
10	3	10	130.34	837.75	**0.39**
instances satisfied by assigning the highest domain value to every variable					
3	10	100	0.02	0.07	**0.00**
6	6	10	17.90	348.19	**0.06**
9	3	10	6.16	28.47	**0.06**
9	3	100	7.60	40.60	**0.90**
9	4	10	$>$ 16 min	263.47	**5.04**
10	3	10	119.39	754.83	**0.61**

Table 1: Performance of MiniSAT on the direct, log and order encoding of constant-closed CSP instances of the form shown in Figure 3.

It is clear from these results that the order encoding is much better than the others for solving these kinds of instances. The order encoding usually produced fewer clauses than the other encodings for instances of this type. However, Mini-SAT's performance was still much better on the clauses produced by the order encoding even in those cases where this encoding produced a larger set of clauses than the other encodings. For instance, for parameters $(9, 3, 100)$ the direct and log encodings produced around 19 700 clauses each, whereas the order encoding produced an average of 46 957 clauses.

7.2 Max-closed constraints

Proposition 7.2. *If a CSP instance P contains only max-closed constraints, then a DPLL-based SAT-solver which assigns False before True will decide the satisfiability of the order encoding of P in linear time.*

Proof. Let F be the order encoding of P. As long as F contains a unit clause, unit propagation takes place. Next, either the solver terminates after discovering an empty

[3]For the instances satisfied by assigning the lowest domain value to every variable, we changed the value ordering to assign True before False. In all other cases we used MiniSAT's default value ordering, which assigns False first.

```
(int v1 1 5)
(int v2 1 5)
(int v3 1 5)
(<= 10 (+ (mul -6 v1) (mul 9 v2) (mul 4 v3) ) )
(<= -3 (+ (mul -10 v1) (mul 2 v2) (mul 4 v3) ) )
(<= 16 (+ (mul 7 v1) (mul 1 v2) (mul 10 v3) ) )
(<= -1 (+ (mul 7 v1) (mul -6 v2) (mul 6 v3) ) )
```

Figure 4: An example of a satisfiable max-closed CSP instance.

clause, or a set of Horn clauses of size > 1 remain. Each such clause contains at least one negative literal. Hence, if a solver then decides to assign value *False* to each variable, every clause will be satisfied and a solution will be found. As unit propagation takes linear time [13, 20, 24], the result follows. □

To obtain satisfiable CSP instances with max-closed constraints, we generated inequality constraints of the form shown in Equation 1, and then selected only those that satisfied some fixed random solution. An example instance is shown in Figure 4.

To generate unsatisfiable instances, we used the same technique as before and then added two further inequalities of the following form:

$$\begin{array}{l} \left(\sum_{i\in\{1,\dots,n\}\setminus\{k\}} v_i\right) - v_k \geq d*(n-1)-1 \\ \left(\sum_{i\in\{1,\dots,n\}\setminus\{j\}} v_i\right) - v_j \geq d*(n-1)-1 \end{array} \tag{4}$$

for some $j \neq k$, where n is the number of variables in the instance and d is the maximum domain value.

We encoded the generated instances using the direct, log and order encodings. As predicted by Theorem 5.5, the order encodings of these max-closed instances contained Horn clauses only. The runtimes of MiniSAT on the various encodings are presented in Table 2.

The results show that the order encoding outperforms the other two encodings, as expected. Moreover, in the unsatisfiable case MiniSAT detects the unsatisfiability purely by unit propagation - no variable is picked for branching. Hence, it is even quicker in solving unsatisfiable instances than in solving the satisfiable ones. The opposite seems to be true for the other two encodings.

7.3 Connected row-convex constraints

Proposition 7.3. *If a CSP instance P contains only connected row-convex constraints, then a clause-learning SAT-solver will decide the satisfiability of the order encoding of P after a linear number of conflicts.*

Proof. Let F be the order encoding of P. By Theorem 5.8, we know that F is in 2-CNF. Consider the case when after unit propagation some binary clause C is falsified. The literals of C have been assigned at the current decision level (since a previous assignment of one of its literals would cause the clause to be either satisfied or become unit and hence

number of CSP variables	number of CSP values	number of constraints	MiniSAT (sec.) direct encoding	MiniSAT (sec.) log encoding	MiniSAT (sec.) order encoding
satisfiable instances					
3	10	100	0.01	0.06	**0.00**
6	6	10	7.42	95.97	**0.07**
9	3	10	1.30	4.99	**0.01**
9	3	100	5.15	23.27	**0.59**
9	4	10	$>$ 16 min	176.16	**0.73**
10	3	10	20.08	107.38	**0.11**
unsatisfiable instances					
3	10	100	0.02	0.06	**0.00**
6	6	10	26.64	515.90	**0.01**
9	3	10	7.85	41.12	**0.00**
9	3	100	7.87	40.15	**0.02**
9	4	10	$>$ 16 min	299.60	**0.02**
10	3	10	140.68	907.51	**0.01**

Table 2: Performance of MiniSAT on the direct, log and order encoding of max-closed CSP instances of the form shown in Figure 4.

trigger further unit propagation). Each clause that caused literals of C to be assigned will also contain literals that have been assigned at the current decision level. As an asserting learning scheme adds a clause that contains only one variable that has been assigned at the current decision level, it will backtrack to a unit clause and add its negation to the clause set. As at most n unit clauses can be added, where n is the number of variables in P, a clause-learning SAT-solver will terminate after at most n conflicts. □

We wrote a program to generate instances of the form shown in Equation 2. For the satisfiable case we ensured that each constraint in an instance satisfied some fixed random solution. For the unsatisfiable case we added four constraints imposed on two randomly picked variables of the form in Equation 5.

$$\begin{array}{l} v_i \leq a \vee v_j \leq b \\ v_i \geq a+1 \vee v_j \leq b \\ v_i \leq a \vee v_j \geq b+1 \\ v_i \geq a+1 \vee v_j \geq b+1 \end{array} \tag{5}$$

An example instance is shown in Figure 5.

As predicted by Theorem 5.8, the order encoding of such instances produced binary clauses only. Comparison with the direct and log encoding is shown in Table 3.

All three encodings performed fairly well on the connected row-convex class, although the log encoding seems to be significantly worse than the others. Since the constraints are all binary, the direct encoding actually generates a set of 2-CNF clauses together with at-least-one clauses, which grow with domain size.

```
(int v1 1 5)
(int v2 1 5)
(int v3 1 5)
(or (<= v1 1) (<= v2 4) )
(or (<= v2 2) (>= v3 3) )
(or (>= v3 3) (>= v1 1) )
(or (<= v3 1) (<= v2 2) )
(or (>= v3 2) (<= v2 2) )
(or (<= v3 1) (>= v2 3) )
(or (>= v3 2) (>= v2 3) )
```

Figure 5: An example of an unsatisfiable connected row-convex CSP instance. The last four constraints ensure that the instance is trivially unsatisfiable.

number of CSP variables	number of CSP values	number of constraints	MiniSAT (sec.) direct encoding	MiniSAT (sec.) log encoding	MiniSAT (sec.) order encoding
satisfiable instances					
100	10	100	0.01	0.01	**0.00**
10	100	10	0.31	6.14	**0.00**
100	100	10	2.85	11.76	**0.02**
10	100	100	15.74	645.36	**0.00**
10	110	100	15.75	696.07	**0.00**
10	200	100	226.75	$>$ 16 min	**0.00**
unsatisfiable instances					
100	10	100	0.01	0.02	**0.00**
10	100	10	0.83	14.97	**0.00**
100	100	10	3.21	16.67	**0.01**
10	100	100	5.78	368.79	**0.01**
10	110	100	9.72	$>$ 16 min	**0.01**
10	200	100	83.33	$>$ 16 min	**0.00**

Table 3: Performance of MiniSAT on the direct, log and order encoding of connected row-convex CSP instances of the form shown in Figure 5.

8 Conclusions

The increasing efficiency of SAT-solvers has led to the development of SAT-based constraint solvers and various SAT encodings for CSPs have now been developed. However, most previous comparison between such encodings has been purely empirical. In this paper we gave a theory-based argument to prefer the order encoding for certain families of constraint satisfaction problems. In particular, we showed that translating such instances using a sparse encoding or the log encoding results in SAT instances which do not fall into known tractable classes. However, translating such instances using the order encoding results in SAT instances that do fall into known tractable classes. Moreover, standard SAT-solvers will then solve them efficiently. We also provided experimental evidence that, as predicted, the order encoding is a considerably better choice in practice, with current SAT-solvers, for max-closed, connected row-convex and certain constant-closed CSP classes than either the direct encoding or the log encoding.

We have shown that using the order encoding to translate a CSP instance that is constant-closed for the lowest domain value gives a set of clauses satisfying the first condition of Schaefer's Dichotomy Theorem. Similarly, constraints that are constant-closed under the highest domain value translate under the order encoding to clauses that satisfy the second condition of the theorem. Max-closed constraints translate to clauses satisfying the third condition of the theorem. By symmetry between min-closed and max-closed constraints, min-closed constraints translate to clauses satisfying the fourth condition of the theorem. Connected row-convex constraints translate to clauses satisfying the fifth condition. The final, sixth, condition in Schaefer's Theorem can never be satisfied using the order encoding, since (for all domains with 3 or more elements) it is already broken by the consistency clauses, $\neg(x^{\leq}_{v,c-1}) \vee (x^{\leq}_{v,c})$. Hence we have given a complete list of all constraint types which are encoded to tractable language classes for SAT using the order encoding.

It would be interesting to see if other encodings could be developed which would allow other tractable CSP languages to be translated to tractable languages for SAT, and hence solved efficiently by SAT-based solvers.

9 Acknowledgements

The provision of an EPSRC Doctoral Training Award to Justyna Petke is gratefully acknowledged. We would also like to thank the reviewers of SAT2011 and Andras Salamon for their valuable comments.

References

[1] 3rd internat. CSP solver competition, http://www.cril.univ-artois.fr/CPAI08/.

[2] 4th internat. CSP solver competition, http://www.cril.univ-artois.fr/CPAI09/.

[3] C. Anstegui and F. Manyà. Mapping Problems with Finite-Domain Variables into Problems with Boolean Variables. *Theory and Applications of Satisfiability Testing - SAT 2004*, vol. 3542, pages 1–15, Springer 2005.

[4] A. Atserias, J. K. Fichte, and M. Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. In *Journal of Artificial Intelligence Research (JAIR)*, volume 40, pages 353–373, JAIR, 2011.

[5] C. Bessiere and R. Debruyne. Theoretical analysis of singleton arc consistency and its extensions. in *Artificial Intelligence Journal*, volume 172:1, pages 29–41, AI 2008.

[6] H. Chen, V. Dalmau and B. Grußien. Arc Consistency and Friends. In *The Computing Research Repository (CoRR)*, volume abs/1104.4993, CoRR 2011.

[7] D. Cohen and P. Jeavons. The complexity of constraint languages. *Handbook of Constraint Programming*, Chapter 8, pages 245–280, Elsevier 2006.

[8] D. Cohen, P. Jeavons, P. Jonsson and M. Koubarakis. Building tractable disjunctive constraints. *Journal of the ACM*, vol. 47, pages 826–853, ACM 2000.

[9] D. Cohen, P. Jeavons, and M. Gyssens A unified theory of structural tractability for constraint satisfaction problems. *Journal of Computer and System Sciences*, vol. 74, pages 721–743, Elsevier 2007.

[10] M.C. Cooper, D. Cohen, and P. Jeavons. Characterising tractable constraints. *Artificial Intelligence Journal*, vol. 65, pages 347–361, Elsevier 1994.

[11] J.M. Crawford and A.B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the twelfth national conference on Artificial intelligence - AAAI'94*, volume 2, AAAI 1994.

[12] Y. Deville, O. Barette, and P. van Hentenryck. Constraint satisfaction over connected row convex constraints. *Proceedings of IJCAI'97*, pages 405–411, IJCAI 1997.

[13] W. Dowling and J. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, vol. 1, pages 267–284, Elsevier 1984.

[14] M. Grohe. The Complexity of Homomorphism and Constraint Satisfaction Problems Seen from the Other Side. *Journal of the ACM*, vol. 54, pages 1–24, ACM 2007.

[15] H.H. Hoos. SAT-encodings, search space structure, and local search performance. *Proceedings of IJCAI'99*, pages 296–302, IJCAI 1999.

[16] P. Jeavons and M.C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence Journal*, pages 327–339, Elsevier 1995.

[17] P. Jeavons, D. Cohen, and M. Gyssens. Closure Properties of Constraints. *Journal of the ACM*, vol. 44, pages 527–548, ACM 1997.

[18] P. Jeavons and J. Petke. Local consistency and SAT-solvers. *Principles and Practice of Constraint Programming - CP'10*, pages 398–413, Springer 2010.

[19] The MiniSat page. http://minisat.se/.

[20] S.D. Prestwich. CNF encodings. *Handbook of Satisfiability*, Chapter 2, pages 75–97, IOS Press 2009.

[21] T.J. Schaefer. The Complexity of Satisfiability Problems. *Proceedings of the 10th ACM Symposium on Theory of Computing - STOC'78*, pages 216–226, ACM 1978.

[22] Sugar - a SAT-based constraint solver. http://bach.istc.kobe-u.ac.jp/sugar/.

[23] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear CSP into SAT. *Constraints Journal*, vol. 14, pages 254–272, Springer 2009.

[24] L. Zhang and S. Malik. The quest for efficient Boolean satisfiability solvers. *Computer Aided Verification*, vol. 2404, pages 641–653, Springer 2002.