Proposal and Evaluation of Hybrid Encoding of CSP to SAT Integrating Order and Log Encodings

Soh, Takehide

Banbara, Mutsunori

Tamura, Naoyuki

(Citation)
International Journal on Artificial Intelligence Tools,26(1):1760005-1760005

(Issue Date)
2017-02

(Resource Type)
journal article

(Version)
Accepted Manuscript

Proposal and Evaluation of Hybrid Encoding of CSP to SAT Integrating Order and Log Encodings*

Takehide Soh

Mutsunori Banbara

Naoyuki Tamura

Information Science and Technology Center, Kobe University,
1-1, Rokko-dai, Nada, Kobe, Hyogo 657-8501, Japan.

{*soh@lion., banbara@, tamura@*}*kobe-u.ac.jp*

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

This paper proposes a new hybrid encoding of finite linear CSP to SAT which integrates order and log encodings. The former maintains bound consistency by unit propagation and works well for constraints consisting of small/middle sized arity and variable domains. The latter generates smaller CNF and works well for constraints consisting of larger sized arity and variable domains but its performance is not good in general because more inference steps are required to ripple carries. This paper describes the first attempt of hybridizing the order and log encodings without channeling constraints. Each variable is encoded by either the order encoding or the log encoding, and each constraint can contain both types of variables. Using the CSP solver competition benchmark consisting of 1458 instances, we made a comparison between the order, log and proposed hybrid encodings. As a result, the hybrid encoding solves the largest number of instances with the shortest CPU time. We also made a comparison with the four stete-of-the-art CSP and SMT solvers Mistral, Opturion CPX, Yices, and z3. In this comparison, the hybrid encoding also shows the best performance. Furthermore, we found that the hybrid encoding is especially superior than other solvers for instances containing disjunctive constraints and global constraints—it indeed solves more instances than the virtual best solver consisting of those four state-of-the-art systems.

Keywords: Order Encoding; Log Encoding; Hybrid;

1. Introduction

Propositional Satisfiability Testing (SAT) is a combinatorial problem to find a variable assignment satisfying the given propositional formula written in Conjunctive Normal Form (CNF)[1]. Recent progress of SAT technologies makes SAT-based ap-

*This paper is an extended version of the paper "A Hybrid Encoding of CSP to SAT Integrating Order and Log Encodings" presented in ICTAI 2015. We improve the previous implementation and all experiments are rebuild from scratch with more benchmarks and other state-of-the-art systems. We also conduct detailed analyses of experimental results from several points of view.

proaches applicable for solving hard and practical combinatorial problems, such as model checking, scheduling, and constraint satisfaction.

In particular, constraint solving is one successful application of SAT-based approaches, in which the given finite linear Constraint Satisfaction Problem (CSP) over integers is encoded to a SAT instance and solved by a SAT solver. Sugar[2], Scarab[3], Azucar[4], BEE[5], Numberjack[6], Picat[7], and meSAT[8] are such SAT-based constraint solvers. Also, there are survey, textbook and related works for the SAT-based constraint solving [9,10,11,12,13]. In the following, we simply call finite linear CSP over integers as CSP.

In most of these SAT-based CSP solvers, the *order encoding* (or its variants) is used to encode CSP variables and constraints, in which a Boolean variable meaning $x \geq a$ is assigned for each integer variable x and its domain value a[14,15,16,17,18,2]. It realizes an efficient encoding of CSP to SAT by maintaining bounds consistency of CSP by unit propagation of SAT solving, and it is known as the only SAT encoding reducing some tractable CSPs to tractable SAT instances[19]. It also showed good performance for a wide range of problems[20,21,22], and particularly, Sugar[a] became a winner in GLOBAL categories at the 2008 and 2009 CSP solver competitions[23].

However, there exists a limitation. The size of the generated SAT instances can be huge when the variable domains and arity of constraints become large. For example, in the order encoding, a linear comparison $\sum_{i=1}^{n} a_i x_i \geq c$ of arity n requires $O(d^{n-1})$ clauses where d is the domain size of the variables x_i's. The number of clauses can be reduced by introducing auxiliary variables but it still can be very large.

On the other hand, the log encoding[24,25] uses a binary representation for integer variables, and the size of the generated SAT instance is much smaller (that is, $O(n \log d)$). However, its performance is not good in general because more inference steps are required to ripple carries.

Therefore, the integration of these two encodings is an important research topic, but, as far as we know, such encoding has not been studied in previous works. It is possible to encode each variable with both encodings, and add extra clauses to channel them. However, such approach loses the merit of the log encoding because we need $O(d)$ clauses for each variable as the channeling constraints.

In this paper, we propose a novel hybrid encoding which integrates the order and log encodings without channeling constraints. It has the following two features. (i) Each variable is encoded by either the order encoding or the log encoding, and each constraint can contain both types of variables. (ii) The "degree" of hybridization can be controlled by classifying the order-encoded and log-encoded variables. We propose to use the *domain product* criterion for that classification.

The way of hybridization in this paper differs from previously proposed hybrid encodings. Direct-order encoding in the literature[17,5,8] uses both of direct and order

[a] *http://bach.istc.kobe-u.ac.jp/sugar/*

encodings for each variable with channeling constraints. Clauses generated by the direct encoding accelerates the speed of solving "=" and "≠" constraints. Log-order encoding in the literature[26] is a mixture of the log and order encodings, that is, one bit is introduced to halve the domain size of each variable. More general idea is presented in the literature[4] as compact order encoding. The literature[27] proposes hybrid encodings for the MinSAT problem that hybridize the minimal support and direct encodings. Also, the literature[28] proposes methods using mixture of log and direct encodings which basically uses the direct encoding but uses the log encoding for at-least-one and at-most-one constraints. However, all of them apply the same encoding method for all variables. On the other hand, the proposed hybrid encoding allows us to use different methods for different variables even if those variables are included in one linear constraint.

We implemented the proposal as the Diet-Sugar system[b]. It is based on the Sugar solver, so it adopts XCSP format input[c] including global constraints [23]. We evaluate its performance with two benchmark sets. The first one consists of our crafted instances to verify synergy effect of the hybridization. We will discuss about problems which are only solved by the proposed encoding. The second one consists of 1458 instances from the 2009 CSP solver competition. On those instances, we made comparisons between the order, log, and hybrid encodings. In addition, we compare the proposed hybrid encoding with the state-of-the-art CSP solvers and SMT solvers over the competition benchmark set. For each experiment, we discuss pros and cons of the proposed hybrid encoding. We also make all experimental data available online[d] so that readers can proceed further study from their own perspective. The comparisons between three encodings shows that the performance of the proposed hybrid encoding is superior to the order and log encodings and is close to their virtual best. In addition, the comparisons with the state-of-the-art solvers show that it outperforms other systems on several types of instances such as i) containing disjunctive constraints (OR), ii) containing global constraints, iii) containing Boolean constraints, and iv) satisfiability.

2. Preliminaries

In this section, we define Constraint Satisfaction Problem (CSP) and its variants: Simple CSP, 0-1 CSP, Pseudo-Boolean Constraint Satisfaction Problem (PB), and Satisfiability Testing Problem (SAT).

In this paper, our interest is in finite linear CSPs which has been encoded to SAT instances[2,3,4,5,6,7,8]. So, we assume CSPs consist of linear expressions on finite-

[b] *http://kix.istc.kobe-u.ac.jp/~soh/dsugar/*

[c] *http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html*

[d] *http://kix.istc.kobe-u.ac.jp/~soh/dsugar/ijait/*

domain integers, and constraints are represented as a disjunction of linear comparisons. Such restriction does not make any limitation practically. Some CSP solvers, such as Sugar, provide a preprocessor translating general CSPs to linearized CSPs.

Let $\mathcal{X}$ be a set of *variables* (symbols such as x, y, x_1, y_1 are used to denote variables). When $x_1, x_2, \ldots, x_n$ are variables, and $a_1, a_2, \ldots, a_n, b$ and c are integer constants, expressions of the form $\sum_{i=1}^{n} a_i x_i + b$ are called *linear summations* (or simply *summations*), and expressions of the form $\sum_{i=1}^{n} a_i x_i \geq c$ are called *linear comparisons* (or simply *comparisons*). The number of variables n is called an *arity* of the summation or the comparison. Without loss of generality, we also assume all a_i's are non-zero, x_i's are all distinct, and the comparison operator is $\geq$ relation. We use S, S_1, ... to denote summations, and E, E_1, ... for comparisons. For simplicity, mathematically equivalent expressions are also used. For example, $x < y$ means $-x+y \geq 1$. Negation operator "$\neg$" is defined as a syntactic function on comparisons: $\neg(S \geq c) = (-S \geq -c+1)$.

A finite set of comparisons is called a *constraint* and denoted by C, C_1, ..., etc. A finite set of constraints is called a *constraint set* and denoted with italic bold face, such as $\boldsymbol{C}$.

Definition 2.1. A tuple $(X, \mathit{Dom}, \boldsymbol{C})$ is called a *Constraint Satisfaction Problem* (CSP) when:

- X is a finite set of variables ($X \subseteq \mathcal{X}$),
- Dom is a function specifying each variables' *domain* (a set of possible values) as a bounded interval of integers,
- $\boldsymbol{C}$ is a constraint set on X.

Constraint set $\boldsymbol{C} = \{C_1, \ldots, C_n\}$ means a conjunction of C_i's, and might be written by $C_1 \wedge \cdots \wedge C_n$ or a sequence of C_i's. Constraint $C = \{E_1, \ldots, E_m\}$ means a disjunction of E_j's, and might be written by $E_1 \vee \cdots \vee E_m$.

$\mathit{Var}(E)$ gives a set of variables occurring in a comparison E. The lower and upper bounds of $\mathit{Dom}(x)$ are denoted by $\mathit{lb}(x)$ and $\mathit{ub}(x)$ respectively.

A variable x is called a *Boolean variable* (denoted by p, q, p_1, q_1, ...) when $\mathit{Dom}(x) \subseteq \{0, 1\}$. A comparison is called a *simple comparison* or a *literal* (denoted by L, L_1, ...) when it does not contain integer variables and consists of at most one Boolean variable. Non-literal comparisons are called *complex*.

Definition 2.2. A constraint is called a *simple constraint* when it contains at most one complex (non-literal) comparison. A CSP only consisting of simple constraints is called a *simple CSP*.

Definition 2.3. A constraint (or a comparison) is called a *0-1 constraint* (or *0-1 comparison*) when all of its variables are Boolean. A CSP only consisting of 0-1 constraints is called a *0-1 CSP*.

Definition 2.4. A 0-1 constraint is called a *PB constraint* (Pseudo-Boolean constraint) when it consists of one comparison with $\geq$ relation. A CSP only consisting

of PB constraints is called a *PB* (Pseudo-Boolean constraint satisfaction problem).

Definition 2.5. A constraint is called a *clause* when all of its comparisons are literals. A CSP only consisting of clauses is called a *SAT*.

Any PB constraint is a 0-1 constraint, and any PB is a 0-1 CSP. Any clause is a simple 0-1 constraint, and any SAT is a simple 0-1 CSP.

Example 2.1. Let p_i's be Boolean variables, and x_i's be non-Boolean variables. $p_1 \geq 1$ is a literal equivalent to p_1, and $p_1 < 1$ is a literal equivalent to $\neg p_1$. Constraint $\{p_1, \neg p_2, x_1 - x_2 \geq 1\}$ is simple since it contains only one complex (non-literal) comparison $x_1 - x_2 \geq 1$. $\{p_1 - p_2 \geq 0, p_2 - p_3 \geq 0\}$ is a 0-1 constraint since it only contains Boolean variables, but it is not simple since there are two complex comparisons. $\{p_1 + p_2 \geq 1\}$ is a PB constraint since it is a 0-1 constraint and contains only one comparison with $\geq$ relation. $\{p_1, p_2\}$ is a clause since it only consists of literals, but it is not a PB constraint in our definition, although it is equivalent to $\{p_1 + p_2 \geq 1\}$.

We here briefly explain encodings and translations described in the following sections. By Tseitin translation described in Section 3.1, a CSP is translated to a simple 0-1 CSP. Then, by the order, log, and their hybrid encodings, a simple CSP is translated to a simple 0-1 CSP, which only contains Boolean variables. The simple 0-1 CSP is translated to a PB by well-known big-M method[29]. We can utilize several methods[30,31,32,33] for encoding PB to SAT.

3. Log and Order Encodings as Partial Encodings

In the former part of this section, we give the definition of a framework called *standard Boolean encoding*. This framework enables the hybridization of SAT encodings without channeling by applying each encoding partially, that is, we apply each encoding for each independent set of variables. In the latter part, we reformulate the log and order encodings as such partial encodings.

3.1. *Standard Boolean Encoding*

A *translation* (or *encoding*) τ of CSPs is called *equi-satisfiable* when the satisfiability of an original CSP and a translated CSP is equivalent. Also, when τ_1 and τ_2 are equi-satisfiable, their composition $\tau_1 \circ \tau_2$ is also equi-satisfiable[e].

To define the log and order encodings as partial encodings in terms of independent sets of variables, we introduce the following *standard Boolean encoding*.

Definition 3.1. Let P be a CSP $P = (X, Dom, \boldsymbol{C})$ and V be a subset of variables such that $V \subseteq X$ which is the target set of variables for the partial encoding. A

[e]Here, the composition is defined by: $(f \circ g)(x) = g(f(x))$.

translation of the CSP P is called a *standard Boolean encoding* in terms of V given by $(U, \boldsymbol{A}, T)$ when:

- U is a set of Boolean variables to be added such that $X \cap U = \emptyset$,
- $\boldsymbol{A}$ is a set of constraints to be added, and
- T is a function translating a comparison to a constraint set defining the translation.

Then, using this definition, a CSP $P' = (X', Dom', \boldsymbol{C}')$ obtained by the standard Boolean encoding is defined by:

$$\begin{aligned} X' &= (X \setminus V) \cup U \\ Dom'(x) &= \begin{cases} Dom(x) & (x \in X \setminus V) \\ \{0,1\} & (x \in U) \end{cases} \\ \boldsymbol{C}' &= \boldsymbol{A} \cup \textstyle\bigwedge_{C \in \boldsymbol{C}} \bigvee_{E \in C} T(E) \end{aligned}$$

Note that $\bigvee_{E \in C} T(E)$ sometimes cause the exponential size explosion. When a constraint $C = \{E_1, ..., E_n\}$ is given and all comparisons are complex linear comparisons, the constraint C is then encoded to $\prod_{i=1}^{n} |T(E_i)|$ of constraints. However, this can be avoided by introducing Tseitin translation [34], which can translate a CSP to a simple CSP.

Example 3.1. Let $C = \{E_1, E_2\}$ be a constraint and both comparisons are complex linear comparisons. We can obtain an equi-satisfiable and a set of simple constraints $\{\{p, q\}, \{\neg p, E_1\}, \{\neg q, E_2\}\}$ via Tseitin translation. Then, the number of encoded constraints can be reduced from $|T(E_1)| \times |T(E_2)|$ to $|T(E_1)| + |T(E_2)| + 1$.

3.2. *Log Encoding*

Log encoding (or binary encoding) [24,25] is a well-known equi-satisfiable translation of CSPs to SAT problems. Here, we define it as a translation to 0-1 CSPs by replacing each variable x by its binary representation.

Let $P = (X, Dom, \boldsymbol{C})$ be a CSP, and $V \subseteq X$ be a set of variables to be encoded. Then, we can define a translation τ_V^{L} (the superscript "L" stands for log encoding) as a standard Boolean encoding of V given by $(U_V^{\mathrm{L}}, \boldsymbol{A}_V^{\mathrm{L}}, T_V^{\mathrm{L}})$ where $p_{x,i}$'s are new Boolean variables, and $m_x = \lfloor \log_2(ub(x) - lb(x)) \rfloor$.

$$\begin{aligned} U_V^{\mathrm{L}} &= \{p_{x,i} \mid x \in V,\ 0 \le i \le m_x\} \\ \boldsymbol{A}_V^{\mathrm{L}} &= \{\{x\sigma \le ub(x)\} \mid x \in V\} \\ T_V^{\mathrm{L}}(E) &= \{\{E\sigma\}\} \\ \sigma &= \{x \mapsto lb(x) + \textstyle\sum_{i=0}^{m_x} 2^i p_{x,i} \mid x \in V\} \end{aligned}$$

where the symbol $\mapsto$ denotes a mapping from variables to summations. $E\sigma$ denotes the result of the substitution σ replacing each occurrence of a variable x by $\sigma(x)$ in a comparison E. It is easy to confirm τ_V^{L} is equi-satisfiable.

Example 3.2. Consider a CSP $(X, Dom, \boldsymbol{C})$, where $X = \{x, y\}$, $Dom(x) = \{1, 2, 3\}$, $\boldsymbol{C} = \{\{E\}\} = \{\{x - y \geq 1\}\}$, and $V = \{x\}$. By τ_V^{L}, the followings are obtained.

$$\begin{aligned} U_V^{\mathrm{L}} &= \{p_{x,0}, p_{x,1}\} \\ \boldsymbol{A}_V^{\mathrm{L}} &= \{\{1 + p_{x,0} + 2p_{x,1} \leq 3\}\} \\ T_V^{\mathrm{L}}(\{E\}) &= \{\{1 + p_{x,0} + 2p_{x,1} - y \geq 1\}\} \end{aligned}$$

3.3. *Order Encoding*

Order encoding [14,2,33] translates CSPs to SAT problems by using new Boolean variables $p_{x \geq d}$'s meaning $x \geq d$ for each variable x and value $d \in Dom(x) \setminus \{lb(x)\}$[f].

For example, consider the order encoding of variables x and y when $Dom(x) = Dom(y) = \{1 \,..\, 5\}$. Four Boolean variables $p_{x \geq 2}$, $p_{x \geq 3}$, $p_{x \geq 4}$, and $p_{x \geq 5}$ are introduced for x ($p_{x \geq 1}$ is unnecessary because $x \geq 1$ is always true), and also for y. Three clauses $\{p_{x \geq 2}, \neg p_{x \geq 3}\}$, $\{p_{x \geq 3}, \neg p_{x \geq 4}\}$, and $\{p_{x \geq 4}, \neg p_{x \geq 5}\}$ are introduced for x, and also for y. Then, a comparison $x - y \geq 0$ can be translated to $\{\{p_{x \geq d}, \neg p_{y \geq d}\} \mid 1 < d \leq 5\}$.

Let $P = (X, Dom, \boldsymbol{C})$ be a CSP, and $V \subseteq X$ be a set of variables to be encoded. Then, we can define a translation τ_V^{O} (the superscript "O" stands for order encoding) as a standard Boolean encoding of V given by $(U_V^{\mathrm{O}}, \boldsymbol{A}_V^{\mathrm{O}}, T_V^{\mathrm{O}})$ where $p_{x \geq d}$'s are new Boolean variables.

$$\begin{aligned} U_V^{\mathrm{O}} &= \{p_{x \geq d} \mid x \in V,\ lb(x) < d \leq ub(x)\} \\ \boldsymbol{A}_V^{\mathrm{O}} &= \{\{p_{x \geq d}, \neg p_{x \geq d+1}\} \mid x \in V,\ lb(x) < d < ub(x)\} \\ T_V^{\mathrm{O}}(E) &= \begin{cases} \{\{E\}\} & (Var(S) \cap V = \emptyset) \\ T'(E) & (Var(S) \cap V \neq \emptyset) \end{cases} \end{aligned}$$

where $E = \sum_{i=1}^{n} a_i x_i \geq c$ and $T'(E)$ is defined as follows.

$$T'(E) = \begin{cases} \{\{p_{x_1 \geq \lceil c/a_1 \rceil}\}\} & (n = 1,\ x_1 \in V,\ a_1 > 0) \\ \{\{\neg p_{x_1 \geq \lfloor c/a_1 \rfloor + 1}\}\} & (n = 1,\ x_1 \in V,\ a_1 < 0) \\ \bigwedge_{d \in Dom(x_i)} (\{\{p_{x_i \geq d+1}\}\} \vee T_V^{\mathrm{O}}(\sum_{j \neq i} a_j x_j \geq c - a_i d)) & \\ \qquad (n > 1,\ x_i \in V,\ a_i > 0) & \\ \bigwedge_{d \in Dom(x_i)} (\{\{\neg p_{x_i \geq d}\}\} \vee T_V^{\mathrm{O}}(\sum_{j \neq i} a_j x_j \geq c - a_i d)) & \\ \qquad (n > 1,\ x_i \in V,\ a_i < 0) & \end{cases}$$

In the above definition, we let $p_{x \geq d} = 1$ when $d \leq lb(x)$, and $p_{x \geq d} = 0$ when $d > ub(x)$. This translation algorithm is based on the Proposition 1 in [33]. It is easy to confirm τ_V^{O} is equi-satisfiable.

[f] In [2], $x \leq d$ is used instead of $x \geq d$.

Example 3.3. Consider a CSP $(X, Dom, \boldsymbol{C})$, where $X = \{x, y\}$, $Dom(x) = \{0, 1, 2\}$, $\boldsymbol{C} = \{\{E\}\} = \{\{x + y \geq 0\}\}$, and $V = \{x\}$. By τ_V^{O}, the followings are obtained.

$$\begin{aligned} U_V^{\mathrm{O}} &= \{p_{x\geq 1}, p_{x\geq 2}\} \\ \boldsymbol{A}_V^{\mathrm{O}} &= \{\{p_{x\geq 1}, \neg p_{x\geq 2}\}\} \\ T_V^{\mathrm{O}}(\{E\}) &= \{\{p_{x\geq 1}, y \geq 0\}, \{p_{x\geq 2}, y \geq -1\}, \{y \geq -2\}\} \end{aligned}$$

3.4. *Translation to SAT*

By the above encodings, a CSP is translated to a 0-1 CSP, which only contains Boolean variables. And it can be translated to a simple 0-1 CSP by Tseitin translation described in Section 3.1. The simple 0-1 CSP can be translated to a PB by well-known big-M method [29].

For example, consider a constraint $\{p, S \geq c\}$ where p is a Boolean variable and S is a summation. It can be translated to $(c - lb(S))p + S \geq c$, where $lb(S)$ denotes the lower bound of the summation. It holds immediately when $p = 1$, and becomes equivalent to $S \geq c$ when $p = 0$.

There are many works proposed to encode PB to SAT [30,31,32,33], and we can use any of them. An alternative way is the direct use of PB solvers [35,36,37,38,30] without encoding to SAT.

4. Proposal of a Concrete Hybrid Encoding Integrating Order and Log Encodings

We can realize the variable-wise hybridization for any encodings represented in the standard Boolean encoding framework. In this section, we propose the hybridization of the order and log encodings.

4.1. *Hybrid Encoding*

Consider a CSP $P = (X, Dom, \boldsymbol{C})$ and $\{V_O, V_L\}$ which is a partition of the set X of CSP variables. By using the partial encodings in the previous section, the hybrid encoding $\tau_X^{\mathrm{H}} = \tau_{V_L}^{\mathrm{L}} \circ \tau_{V_O}^{\mathrm{O}}$ is simply defined as follows:

$$\begin{aligned} U_X^{\mathrm{H}} &= U_{V_L}^{\mathrm{L}} \cup U_{V_O}^{\mathrm{O}} \\ A_X^{\mathrm{H}} &= A_{V_L}^{\mathrm{L}} \cup A_{V_O}^{\mathrm{O}} \\ T_X^{\mathrm{H}}(E) &= T_{V_O}^{\mathrm{O}}(T_{V_L}^{\mathrm{L}}(E)) \end{aligned}$$

Note that the composition is commutative for the log and order encodings, i.e., $\tau_{V_L}^{\mathrm{L}} \circ \tau_{V_O}^{\mathrm{O}} = \tau_{V_O}^{\mathrm{O}} \circ \tau_{V_L}^{\mathrm{L}}$.

Example 4.1. Consider a CSP $(X, Dom, \boldsymbol{C})$, where $X = \{x, y, z\}$, $Dom(\cdot) = \{0, 1, 2\}$, $\boldsymbol{C} = \{\{E\}\} = \{\{x + y + z \geq 3\}\}$. $V_L = \{x, y\}$ and $V_O = \{z\}$ are also

given. By τ_X^{H}, the followings are obtained.

$$\begin{aligned}
U_X^{\mathrm{H}} &= \{p_{x,0}, p_{x,1}, p_{y,0}, p_{y,1}\} \cup \{p_{z\geq 1}, p_{z\geq 2}\} \\
A_X^{\mathrm{H}} &= \{\{2p_{x,1} + p_{x,0} \leq 2\}, \{2p_{y,1} + p_{y,0} \leq 2\}\} \cup \\
&\quad \{\{p_{z\geq 1}, \neg p_{z\geq 2}\}\} \\
T_X^{\mathrm{H}}(E) &= T_{V_O}^{\mathrm{O}}(T_{V_L}^{\mathrm{L}}(E)) \\
&= T_{V_O}^{\mathrm{O}}(\{2p_{x,1} + p_{x,0} + 2p_{y,1} + p_{y,0} + z \geq 3\}) \\
&= \{\{2p_{x,1} + p_{x,0} + 2p_{y,1} + p_{y,0} \geq 3, p_{z\geq 1}\}, \\
&\quad \{2p_{x,1} + p_{x,0} + 2p_{y,1} + p_{y,0} \geq 2, p_{z\geq 2}\}, \\
&\quad \{2p_{x,1} + p_{x,0} + 2p_{y,1} + p_{y,0} \geq 1\}\}
\end{aligned}$$

4.2. *Classification of Variables*

To develop a concrete hybrid encoding, we need to classify V_L and V_O from X. Our basic idea is using the order encoding as much as possible and the log encoding only for variables included in constraints which is encoded to the large number of SAT clauses.

We consider two ways to detect such log encoding variables. One uses only variable domains and another one uses both variable domains and the arity of constraints. The former is an intuitive way but it sometimes overlooks the increase of clauses in the order encoding.

For instance, consider three integer variables $x, y \in \{1, \ldots, 1000\}$, $z \in \{1, \ldots, 10000\}$, and two constraints: $x \leq z$ and $x + y \leq z$. If we use domain sizes as the classification criterion and define the threshold as 1000, x and y are classified to the order encoding, and z is classified to the log encoding. The numbers of clauses generated by the order encoding are 12993 for the first constraint and 1011993 for the second constraint including axiom clauses, which is almost limit of modern SAT solvers. As is shown by this example, the number of clauses rapidly increases when the arity of constraint becomes larger. To detect such explosion, it is necessary to consider not only domain sizes but also the arity of constraints.

In this paper, we propose to use a product of domain sizes, named *domain product*, as the classification criterion. Let E be a linear comparison $\sum_{i=1}^{n} a_i\, x_i \geq c$, the domain product $DP(E)$ is defined as follows.

$$DP(E) = \prod_{i=1}^{n-1} |Dom(x_i)|$$

Here, we assume variables are sorted by increasing order of domain sizes. The upper bound of the number of generated clauses in the order encoding is $O(d^{n-1})$ where d is domains size and n is the arity of constraint, and thus, the domain product can be seen a rough estimation of the number of generated clauses by the order encoding. In addition to the domain product, it is reasonable to encode variables in PB constraints by the log encoding.

Finally, we define the proposed criterion. Given a CSP $(X, Dom, \boldsymbol{C})$ and threshold θ, the set of the log encoding variables in the hybrid encoding is defined as follows:

$$V_L = \{v \in Var(E) \mid DP(E) > \theta \text{ or } E \text{ is a 0-1 comparison}, \ E \in c, \ c \in \boldsymbol{C}\}$$

The proposed hybrid encoding behaves as follows. When $V_L = \emptyset$, it is equivalent to the order encoding. When $V_L = X$, it is equivalent to the log encoding. When $\emptyset \subsetneq V_L \subsetneq X$, it hybridizes the order and log encodings.

5. Implementation

5.1. Diet-Sugar*: SAT-based CSP Solver using Hybrid Encoding*

We implemented the proposed hybrid encoding as a SAT-based CSP solver named Diet-Sugar, which is based on Sugar[2] written in Java. It accepts all XCSP format [23] including global constraints like *alldifferent*. It also accepts the original input format of Sugar. Diet-Sugar solves CSPs as follows:

(1) Convert a given CSP into the form of the CSP of Definition 2.1, and obtain a simple CSP by Tseitin translation described in Section 3.1.
(2) Encode the simple CSP into a simple 0-1 CSP by the proposed hybrid encoding in Section 4.
(3) Convert the simple 0-1 CSP into a SAT instance as is described in Section 3.4.

For the step 1, we use Sugar as a pre-processor to decompose global constraints into linear comparisons. In this paper, as the backend of Diet-Sugar, we use the latest version of minisat+ (ver. 1.1) and its default SAT solver minisat (ver. 2.2) both of which are available in github[g, h]. Diet-Sugar can also use any SAT solvers[39,40,36,41,35] by using a SAT-based PB solver minisat+ as a CNF generator.

minisat+ provides four translation methods to encode PB constraints into SAT: BDD, Sorter, Adder and their hybridization[33]. We use the BDD translation in Diet-Sugar which is a representative PB to SAT translation and maintains arc consistency. Also, it shows the best performance among those four translation in the preliminary experiments.

5.2. *Improvements to* minisat+

In authors' previous paper[42], we used the original minisat+ without any changes. However, we observed that there is not so small performance difference between the order encoding of Diet-Sugar and Sugar: the former solved 1134 instances out of 1458

[g] *https://github.com/niklasso*

[h] We also tested glucose 4.0 as a backend SAT solver by rebuilding minisat+ with it. For 1458 benchmark instances used in the paper, minisat (ver. 2.2) solved 1216 and glucose (ver. 4.0) solved 1215 instances, respectively.

instances used in Section 6, while the latter solved 1164 instances. It is not negligible even if there is an overhead caused by the classification process and implementation difference. After investigation, we found the overhead mainly comes from the PB normalization pre-processing in minisat+. This pre-processing is executed for each PB constraint even if a constraint is obviously clausal. So, we add about 20 line modifications to minisat+ so that it directly passes the input constraints to SAT solvers when the input is clausal. Thanks to this fix, the order encoding of Diet-Sugar is able to solve 1161 instances and the difference is almost negligible.

6. Experimental Evaluation for Hybrid Encoding

This section evaluates the performance of the proposed hybrid encoding by comparing it with the order and log encodings. For a fair comparison, all three encodings are switched at the step 2) of Section 5 in our implementation Diet-Sugar. All experiments are carried out on OSX equipped with Intel Xeon E5 of 3.7GHz with 32GB RAM.

Two benchmark sets are used. The first one is our crafted benchmark set to verify synergy effect of the hybridization. The second one consists of 1458 instances from the 2009 CSP solver competition to have a comprehensive evaluation on wide variety of problems.

In the following experiments, we use the threshold $\theta = 4096$ for the variable classification described in Section 4.2, which shows the best performance among 1024, 2048, 4096, and 8192 in preliminary experiments. The detailed results for all instances are available in the supplemental web page of Diet-Sugar [i].

6.1. *Evaluation on Crafted Benchmark Set*

6.1.1. *Problem Definition of Crafted Instances*

In this section, we consider a problem that includes both kinds of constraints—those are suitable for the order encoding and those are suitable for the log encoding—to test the synergy effect in the proposed hybrid encoding. Given n and d, let x_i, y_i $(1 \leq i \leq n)$ be integer variables whose domains are $\{0, \ldots, d\}$. Consider a CSP consisting of the following constraints:

$$\bigwedge_{i=1}^{n} x_i \geq y_i \ \wedge \ \sum_{i=1}^{n} x_i < \sum_{i=1}^{n} y_i$$

This problem is unsatisfiable for any n and d since the sum of x's must be greater than or equal to the sum of y's by the left hand side constraints. When we generate benchmark instances of this problem, $x_1 + \cdots + x_n$ are recursively split in half (similarly, for y's) until we get ternary constraints and replace them with newly introduced auxiliary variables. For instance, when $n = 4$ the problem is as follows:

[i] *http://kix.istc.kobe-u.ac.jp/ soh/dsugar/ijait/*

Table 1. CPU Time, Numbers of Clauses and Conflicts on Crafted Benchmark

n	d	CPU Time (sec.)			#Clauses			#Conflicts		
		Order	Log	Hybrid	Order	Log	Hybrid	Order	Log	Hybrid
4	50	3.34	0.97	4.73	63,179	1,088	33,644	50,026	67,154	7,644
4	100	15.14	1.79	1.78	246,379	1,283	1,283	156,794	168,198	168,198
4	150	79.24	2.28	2.28	549,579	1,482	1,482	422,925	218,686	218,686
4	200	253.4	4.03	4.02	972,779	1,470	1,470	935,133	436,850	436,850
4	250	566.73	2.4	2.41	1,515,979	1,438	1,438	1,390,979	249,755	249,755
4	300	TO	6.76	6.78	2,179,179	1,677	1,677	—	716,163	716,163
8	10	0.98	1.17	0.99	12,923	1,691	12,923	8,508	61,529	8,508
8	20	3.51	31.54	3.74	48,283	2,118	29,774	64,066	2,339,531	13,388
8	30	35.22	87.96	7.52	106,043	1,998	62,986	536,078	6,498,426	28,201
8	40	44.17	964.11	12.77	186,203	2,529	45,177	500,552	52,695,464	99,141
8	50	132.55	705.88	21.1	288,763	2,487	67,599	1,150,698	41,806,507	147,876
8	60	344.96	762.02	54.36	413,723	2,441	94,633	1,851,177	47,414,606	454,942

$$
\begin{array}{llll}
x_1 \geq y_1 & x_1 + x_2 = u_1 & u_1 + u_2 = s & s < t \\
x_2 \geq y_2 & x_3 + x_4 = u_2 & & \\
x_3 \geq y_3 & y_1 + y_2 = v_1 & v_1 + v_2 = t & \\
x_4 \geq y_4 & y_3 + y_4 = v_2 & &
\end{array}
$$

where u's, v's are auxiliary variables representing partial sum, s and t represent total sum of $\sum_{i=1}^{n} x_i$ and $\sum_{i=1}^{n} y_i$ respectively.

This problem and translation are a good test-bed to check the basic performance of three encodings by the following reasons. It provides us constraints of several domain products in one instance since variable domain sizes in each partial sum are varied by their depth, e.g., given $d = 10$ in the above example, $DP(x_1 + x_2 = u_1)$ is 100 while $DP(u_1 + u_2 = s)$ is 10000. Besides that, we can change the characteristic of instances by changing n and d. For instance, if d is larger, more clauses are necessary to encode this problem, which would be fit for the log encoding. If n is larger, more propagation is necessary to solve this problem, which would be fit for the order encoding, since a value assignment to x_i must be propagated to s via at least $\lfloor \log_2 n \rfloor$ auxiliary variables.

The following two sets of parameters are used to generate benchmark instances, and all experiments are performed with the 1000 seconds timelimit.

- The first set of parameters consists of $n = 4$ and d varying as 50, 100, 150, 200, 250, 300.
- The second set of parameters consists of $n = 8$ and d varying as 10, 20, 30, 40, 50, 60.

6.1.2. *Experimental Results on Crafted Instances*

Table 1 shows the CPU time, the number of clauses generated by each encoding, and the number of conflicts during SAT solving.

The log encoding generates very compact CNF and solves all instances of $n = 4$ within 10 seconds even if d becomes 300. Nevertheless, the log encoding is not good

for instances of $n = 8$. A reason is that, to ripple carries, the log encoding requires more inference steps than the other encoding, and thus it is hard to solve instances of $n = 8$ that further requires propagation.

In contrast, the order encoding solves more instances of $n = 8$ with less conflicts and CPU time than the log encoding. However, it cannot solve the biggest instance of $n = 4$ since the number of generated clauses becomes over 2 millions, which is three orders of magnitude larger than ones of the log encoding.

Our proposed hybrid encoding correctly inherits the advantages of both encodings. Furthermore, it solves instances of $n = 8$ with much less conflicts and CPU time than both encodings, which demonstrate the synergy effect of the hybridization. In particular, the instance of $n = 8$ and $d = 60$ is solved by the hybrid encoding with only 454942 conflicts.

6.2. *Evaluation on the CSP Solver Competition Benchmark*

6.2.1. *Benchmark Instances*

We carry out experiments on 1458 instances from the CSP solver competition 2009, which is the latest competition using XCSP instances. We use all instances satisfying the condition "intentional category having sequential domain". Instances of intentional category include only intentional constraints (not extensional ones). Though both intensional and extensional benchmarks are important, in this paper, we deal with intensional benchmarks since real world problems are often described by intensional constraints.

In those 1458 instances, maximum domain sizes are up to 20001 and maximum arities are up to 3861. All experiments are performed with the 1000 seconds time-limit.

6.2.2. *Evaluation of the Variable Classification by Domain Product*

This section provides an empirical evaluation for the variable classification method using the domain product described in Section 4.2. Table 2 shows the numbers of instances solved by each encoding. Each row denotes the interval of the domain product. The second column denotes the numbers of instances whose maximum domain products are in the intervals. The remaining columns denotes the numbers of solved instances and their percentages to the subtotal number of instances.

This table tells us that a better encoding is switched while the domain product varying from small to large. The order encoding solved more instances than the log encoding when the domain product is small. In contrast, the log encoding solved more instances when the domain product becomes large. This fact supports that the use of the domain product is a good baseline to classify variables to the order and log encodings. Consequently, the hybrid encoding inherits the advantages of both encodings and solved the largest number of instances 1216, while the order and log encodings solved 1161 and 1170.

Table 2. Number of instances solved by each encoding. Each row denotes the interval of the domain product (DP). The best figures are shown in bold face. Numbers in parentheses denote the percentage of the subtotal number of instances in each row.

DP Interval	#Ins.	Order	Log	Proposal
$1 < DP \leq 128$	447	**344** (77%)	330 (74%)	343 (77%)
$128 < DP \leq 256$	88	**80** (91%)	**80** (91%)	**80** (91%)
$256 < DP \leq 512$	58	**51** (88%)	**51** (88%)	**51** (88%)
$512 < DP \leq 1024$	101	**74** (73%)	56 (55%)	73 (72%)
$1024 < DP \leq 2048$	59	**54** (92%)	45 (76%)	**54** (92%)
$2048 < DP \leq 4096$	22	**20** (91%)	16 (73%)	**20** (91%)
$4096 < DP \leq 8192$	23	21 (91%)	**22** (96%)	**22** (96%)
$8192 < DP \leq 16384$	20	19 (95%)	**20** (100%)	**20** (100%)
$16384 < DP$	640	498 (78%)	550 (86%)	**553** (86%)
Total	1458	1161 (80%)	1170 (80%)	**1216** (83%)

Another question is that the above turnover is observed when we use other PB solvers which are not SAT-encoding based. We carried out experiments using two distinct PB solvers both of which are different from minisat+ in the sense that they directly solve PB instances and do not encode into SAT. Sat4j-pb[43] is an extension from SAT to PB which is done by replacing the resolution performed between clauses by cutting planes, following the spirit of PBChaff[44] or Galena[45]. clasp[46,47] supports counter-based native PB constraints handling in CDCL solving. Table 3 shows the results. Although those two PB solvers take different approaches, a better encoding is switched while the domain product varying from small to large.

In addition, we need more investigation about the threshold of the domain product. This time we use 4096 and empirical results support this value. However, the value may depend on benchmark instances. Studying how to decide the threshold is an important research subject.

Table 3. Number of instances solved by each encoding using other PB solvers Sat4j-pb and clasp. Table format is the same as Table 2.

DP Interval	#Ins.	Sat4j-pb[43]			clasp[46,47]		
		Order	Log	Proposal	Order	Log	Proposal
$1 < DP \leq 128$	447	335 (75%)	321 (72%)	**336** (75%)	**341** (76%)	325 (73%)	**341** (76%)
$128 < DP \leq 256$	88	**80** (91%)	77 (88%)	**80** (91%)	**81** (92%)	77 (88%)	**81** (92%)
$256 < DP \leq 512$	58	**50** (86%)	49 (84%)	**50** (86%)	**51** (88%)	50 (86%)	**51** (88%)
$512 < DP \leq 1024$	101	**67** (66%)	45 (45%)	**67** (66%)	**72** (71%)	47 (47%)	**72** (71%)
$1024 < DP \leq 2048$	59	**55** (93%)	44 (75%)	54 (92%)	**55** (93%)	46 (78%)	**55** (93%)
$2048 < DP \leq 4096$	22	15 (68%)	**16** (73%)	15 (68%)	**19** (86%)	17 (77%)	**19** (86%)
$4096 < DP \leq 8192$	23	19 (83%)	**21** (91%)	**21** (91%)	20 (87%)	**21** (91%)	**21** (91%)
$8192 < DP \leq 16384$	20	**19** (95%)	**19** (95%)	18 (90%)	19 (95%)	**20** (100%)	**20** (100%)
$16384 < DP$	640	480 (75%)	535 (84%)	**536** (84%)	491 (77%)	545 (85%)	**548** (86%)
Total	1458	1120 (77%)	1127 (77%)	**1177** (81%)	1149 (79%)	1148 (79%)	**1208** (83%)

6.2.3. *Relations between Instances Solved by the Order, Log, and Hybrid Encodings*

This section evaluates how comprehensively the hybrid encoding covers the advantages of the order and log encodings. In addition, we also discuss its current limitation.

The inclusion relation of instances solved by the three encodings is shown in Fig. 1. The number 1106 in the center shows the number of commonly solved instances by three encodings. The number $48 + 7$ in the left shows the number that the order encoding can solve but the log encoding cannot, and $61 + 3$ shows the opposite number. The hybrid encoding indeed covered 92% (109/119) of those instances and solved 1 more instances which cannot be solved by both. In this sense, for those 1458 instances, the proposed hybrid encoding realizes the integration of the order and log encodings in a high level. In fact, even if we assume the virtual best encoding consists of the order and log encodings, the hybrid encoding solved almost the same number of instances, i.e., only 10 less out of 1216.

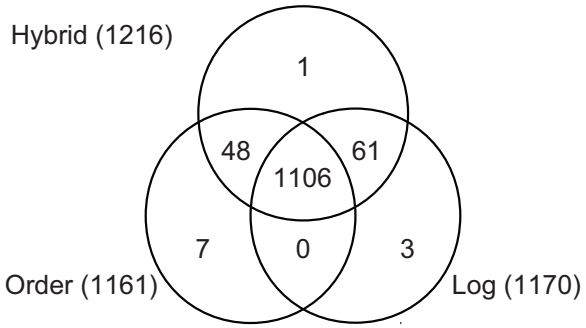

Fig. 1. Number of instances solved by each encoding and their inclusion relations.

Table 4. The 10 instances that the hybrid encoding did not solve. Asterisk * denotes that the instance is a pure PB instance. The columns DP denotes the minimum and maximum domain product, S/U denotes the satisfiability, Enc. denotes the encoding solved the instance with its CPU time, and Proposal denotes the encoding selected by the proposed hybrid encoding.

Instance Name	#Var.	#Con.	DP	S/U	Enc. (CPU)	Proposal
squares-35-35	2450	2978	(1, $> 2^{14}$)	SAT	Order (451.35)	Hybrid
bibd-15-75-40-8-20_glb*	9000	8070	(1, $> 2^{14}$)	SAT	Order (516.64)	Log
ft10x2-837	201	210	(1, $> 2^{14}$)	UNSAT	Order (634.65)	Hybrid
ppp-1-9-16-19*	4626	30921	(4, $> 2^{14}$)	SAT	Order (543.47)	Log
ppp-1-11-19-21*	4602	30353	(4, $> 2^{14}$)	SAT	Order (618.75)	Log
par-32-3*	6352	13332	(1, 4)	SAT	Order (144.67)	Log
pigeon-clauses-15-10*	150	1065	(2, 512)	UNSAT	Order (887.48)	Log
domino-2000-2000	2000	2000	(1, 2000)	SAT	Log (5.09)	Order
knights-50-25	25	300	(50, 2500)	UNSAT	Log (60.6)	Order
lei450-15d-14	450	16750	(14, 14)	UNSAT	Log (470.09)	Order

In turn, let us focus on what cannot be covered by the current hybrid encoding, i.e., the difference to the virtual best of the order and log encodings. First 7 instances in Table 4 corresponds to the 7 instances only solved by the order encoding in Fig.1, and following 3 instances are ones by the log encoding.

Those 10 instances could not be simultaneously turned "solved" by naively changing the threshold. However, we can improve the hybrid encoding by adding some classification rules.

For instance, after further investigation, we found that even if a given instance is a PB problem, if the maximum domain product of instances is less than the current threshold 4096 then the set of instances solved by the order encoding is a superset of the one by the log encoding. Thus, by adding the rule "if the maximum domain product is less than the threshold then uses the order encoding even for PB instances" we can solve 2 more instances, i.e., "par-32-3" and "pigeon-clauses-15-10" without any sub effect.

In this section, we discussed how comprehensively the hybrid encoding covers the advantages of the order and log encodings. It indeed covers 92% (109/119) of instances. We also discussed the missing 10 instances and possible improvements. In the next section, we show the total performance of the proposed hybrid encoding.

6.2.4. *Total Performance Comparison between the Order, Log, and Hybrid Encodings.*

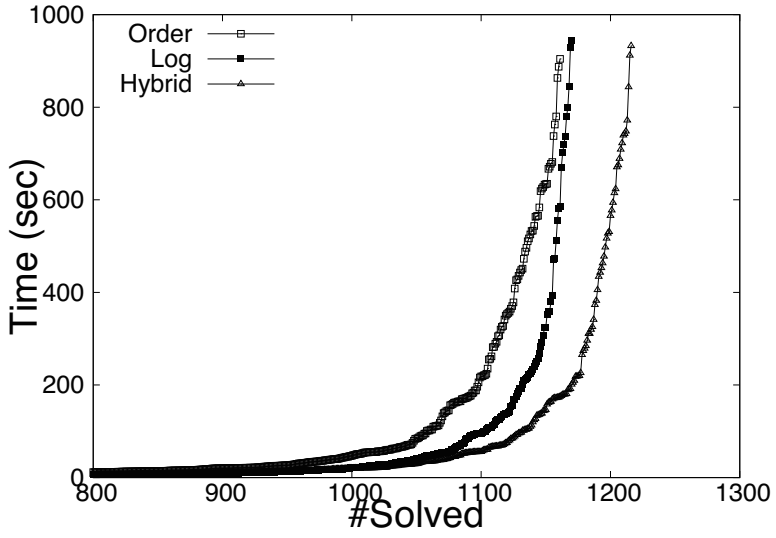

Fig. 2. Cumulative number of instances solved versus CPU time by three encodings

Fig. 2 shows the cactus plot for three encodings and Table 5 shows the number

Table 5. Number of instances solved by each encoding. Each row denotes the series of instances. The best figures are shown in bold face. Series shown in bold face have constraints of multiple domain products over the threshold, i.e., the proposed hybrid encoding uses both of the order and log encodings. Numbers in parentheses denote the percentage of the subtotal number of instances in each row.

Series	#Ins.	Order	Log	Proposal
2D Strip Packing	20	**8** (40%)	7 (35%)	**8** (40%)
All Interval Series	15	**9** (60%)	**9** (60%)	**9** (60%)
BIBD	83	**80** (96%)	79 (95%)	79 (95%)
BMC	15	**15** (100%)	**15** (100%)	**15** (100%)
Chessboard Coloration	15	**12** (80%)	**12** (80%)	**12** (80%)
Cumulative Job-Shop	10	**5** (50%)	4 (40%)	4 (40%)
Domino	10	9 (90%)	**10** (100%)	9 (90%)
Fischer	25	21 (84%)	**24** (96%)	**24** (96%)
Golomb Ruler	28	**23** (82%)	18 (64%)	**23** (82%)
Graph Coloring	141	**98** (70%)	97 (69%)	**98** (70%)
Haystacks	15	**3** (20%)	1 (7%)	**3** (20%)
Job-Shop	76	**67** (88%)	62 (82%)	**67** (88%)
Knights	10	7 (70%)	**10** (100%)	9 (90%)
Langford	43	**27** (63%)	24 (56%)	**27** (63%)
Latin Square	10	**9** (90%)	5 (50%)	**9** (90%)
Magic Square	18	10 (56%)	**13** (72%)	**13** (72%)
Multi Knapsack	6	4 (67%)	**6** (100%)	**6** (100%)
NengFa	7	**7** (100%)	6 (86%)	**7** (100%)
Open-Shop	75	**71** (95%)	61 (81%)	**71** (95%)
Perfect Square Packing	74	**58** (78%)	56 (76%)	**58** (78%)
Pigeons	29	**23** (79%)	**23** (79%)	**23** (79%)
Primes	76	45 (59%)	**70** (92%)	**70** (92%)
Pseudo-Boolean	363	290 (80%)	**312** (86%)	**312** (86%)
Quasigroup Existence	5	**5** (100%)	**5** (100%)	**5** (100%)
Queens	15	**11** (73%)	**11** (73%)	**11** (73%)
Queens-Knights	10	**10** (100%)	**10** (100%)	**10** (100%)
RCPSP	78	**78** (100%)	**78** (100%)	**78** (100%)
Rader Surveillance	65	**65** (100%)	64 (98%)	**65** (100%)
Ramsey	16	**10** (63%)	**10** (63%)	**10** (63%)
Schurr's Lemma	10	**9** (90%)	8 (80%)	**9** (90%)
Social Golfers	10	**6** (60%)	**6** (60%)	**6** (60%)
Super-solutions	85	**66** (78%)	54 (64%)	**66** (78%)
Total	1458	1161 (80%)	1170 (80%)	**1216** (83%)

of instances solved by three encodings, which is organized series-by-series.

We first discuss the result given by Table 5. Series shown in bold face have constraints of multiple domain products over the threshold, i.e., the proposed hybrid encoding uses both of the order and log encodings for each instance on those series.

Since there are various kinds of instance series, the both order and log encodings have pros and cons. The hybrid encoding appropriately covers those series with the help of both encodings by using the domain product. In addition, the synergy effect of the order and log encodings are observed in "Fischer" and "Primes". Although the log and hybrid encodings solve the largest number of instances, the cumulative CPU times of the hybrid encoding is better than the log encoding: 2865 sec. vs.

3344 sec. in "Fischer" and 1918 sec. vs. 2303 sec. in "Primes".

At last, we conclude this section with the overall cactus plot—the relation between the cumulative number of solved instances and the CPU time for three encodings. Fig. 2 shows that the hybrid encoding solved the largest number of instances in any cut-off time.

7. Comparison with State-of-the-art CSP and SMT Solvers

The purposes of this section are twofold. One is to check which types of instances are suitably solved by the proposed hybrid encoding when we compare it with other systems. Another is to check whether the hybrid encoding has a comparable performance with other systems. For the comparisons, it is difficult to compare it with comprehensive systems due to the limitation of computational resource and time. However, we collect four state-of-the-art CSP and SMT solvers[j].

7.1. *Description of other Solvers used.*

The followings CSP and SMT solvers are used in the experiments. We also add the virtual best solver consists of those four solvers to discuss the difference of our proposed system to other systems.

- Mistral (version 1.550)[48] is a pure CSP solver which equips an AC3-like algorithm as a constraint propagator. In particular, for binary relations it uses the technique called AC3-bitset described in the literature [49]. It won the first prize of "n-ary constraints in intension" and other categories in the last CSP solver competition (CSC 2009).
- Opturion CPX (version 1.0.2) is a CSP solver branched from G12/CPX solver which is a part of the G12 project[50]. It is a hybrid solver combining CSP solver and SAT solver, and won the first prize of "Fixed/Free" category in the last MiniZinc challenge (Minizinc 2015).
- Yices (version 2.4.2)[51] is an SMT solver which includes both Boolean satisfiability solver and theory solvers. Yices won the first prize of the category of "Quantified Free Linear Integer Arithmetic (QF_LIA)" in the last International Satisfiability Modulo Theories Competition (SMT-COMP 2015). The theory solver QF_LIA is used in the following experiments.
- Z3 (version 4.3.2)[52] is also an SMT solver and it also won the second prize of "QF_LIA" in SMT-COMP 2015. As same as Yices, the theory solver QF_LIA is used in the following experiments.
- $\mathrm{VBS}_{\mathrm{moyz}}$ is a virtual best solver which consists of the best solver for each instance from Mistral, Opturion CPX, Yices, and Z3. The subscript $\mathrm{VBS}_{\mathrm{moyz}}$ means the initial letters of the four solvers.

[j] Sugar is not used here since the performance difference between the order encoding of Diet-Sugar and Sugar is negligible as is described in Section 5.2.

7.2. *Experimental Condition*

We use the same 1458 instances described in Section 6.2.1. All experiments are carried out on OSX equipped with Intel Xeon E5 of 3.7GHz with 32GB RAM. We make all execution logs available in the web page[k] to be sure the reproducibility. In addition, all solvers we compared can be downloaded from the web page of each solver: Yices[l], Z3[m], Opturion CPX[n], Mistral[o]. Since the input formats of each solver are different, we prepare the benchmark instances in three formats. Note that, of course, all CPU time of the following conversion is excluded from the results in the latter section.

- For Mistral, it can directly handle XCSP format benchmark instances which can be downloaded from the CSP solver competition web page[p].
- For SMT solvers, we use the translation function of Sugar, which convert XCSP format into SMT-LIB standard 2[53]. Since SMT-LIB does not support global constraints except Alldifferent, other global constraints are decomposed to linear arithmetic as same as Diet-Sugar.
- For Opturion CPX, at first, we convert XCSP format into MiniZinc by using the tool xcsp2mzn[q]. Next, we convert MiniZinc format to FlatZinc by using the MiniZinc library appended to the package of Opturion CPX.

7.3. *Experimental Results Compared with State-of-the-art Solvers*

7.3.1. *Which types of instances are suitably solved by the hybrid encoding?*

Comparing with other systems, this section discusses which types of instances are suitably solved by the hybrid encoding. We use the following perspectives: SAT or UNSAT, with OR (disjunction) or not, the maximum domain sizes, and the categories of the CSP solver competition.

Table 6 shows the number of solved instances organized by their satisfiability. Among 1458 instances, the 782 instances are SAT, the 529 instances are UNSAT, and the remains are unknown, i.e., any solvers cannot solve them. Comparing with each of other systems, the hybrid encoding solved the largest number of instances

[k] *http://kix.istc.kobe-u.ac.jp/~soh/dsugar/ijait/*

[l] *http://yices.csl.sri.com*

[m] *https://github.com/Z3Prover/z3/wiki*

[n] *http://www.opturion.com*

[o] *http://homepages.laas.fr/ehebrard/mistral.html*

[p] *http://www.cril.univ-artois.fr/CSC09/benchs/CSC09.tar*

[q] *https://github.com/CP-Unibo/mzn2feat/tree/master/xcsp2mzn*

763 and 453 for both SAT and UNSAT instances. In particular, for SAT instances, it solves more instances than the virtual best of other solvers VBS_{moyz} which solved 762. Furthermore, the percentage of solved instances is close to 100%. For those 1458 instances, we can say that the proposed hybrid encoding is particularly good at solving SAT instances.

Table 6. SAT/UNSAT based Comparison: the numbers of instances solved by each system. Numbers in parentheses denote the percentage of the subtotal number of each row. Each row denotes sets of instances which are SAT or UNSAT. The unsolved instances are denoted as "Unknown".

Satisfiability	#Ins.	Proposal	Other CSP/SMT Solvers			
		(Hybrid)	Mistral	Opturion	Yices	Z3
SAT	782	**766** (98%)	684 (87%)	695 (89%)	618 (79%)	579 (74%)
UNSAT	523	**450** (86%)	431 (82%)	376 (72%)	366 (70%)	362 (69%)
Unknown	153	**0** (0%)	**0** (0%)	**0** (0%)	**0** (0%)	**0** (0%)
Total	1458	**1216** (83%)	1115 (76%)	1071 (73%)	984 (67%)	941 (65%)

Table 7 shows the number of solved instances organized by whether instances include disjunctive (OR) constraints or not. Among 1458 instances, the 413 instances include OR and the remaining 1045 instances do not include OR. Comparing with each of other systems, the hybrid encoding solved the largest number of instances for both "with OR" and "without OR". The "with OR" instances consist of highly combinatorial problem series such as "Job-Shop", "Open-Shop", "2D Strip Packing", "Perfect Square Packing" and "Ramsey".

In particular, for "with OR" instances, it solves more instances than the virtual best solver of other systems VBS_{moyz} which solves 339 instances. A reason is that the proposed system is a pure SAT-based system and SAT solvers are good at handling such disjunctive constraints.

Table 7. OR based Comparison: each row denotes sets of instances which contains disjunctive constraints or not. Table format is the same as Table 6.

Constraints	#Ins.	Proposal	Other CSP/SMT Solvers			
		(Hybrid)	Mistral	Opturion	Yices	Z3
with OR	413	**346** (84%)	276 (67%)	321 (78%)	305 (74%)	283 (69%)
without OR	1045	**870** (83%)	839 (80%)	750 (72%)	679 (65%)	658 (63%)
Total	1458	**1216** (83%)	1115 (76%)	1071 (73%)	984 (67%)	941 (65%)

Table 8 shows the number of solved instances organized by the interval of max domain sizes of instances. For instance, the first row represents that the set of instances whose maximum domain sizes are from 1 to 2, i.e., those instances consists of Boolean variables. The hybrid encoding solved the largest number of instances for small domains and large domains. This result well demonstrates the goodness of the proposed hybrid encoding. The both log and order encodings contribute for small domains, and the log encoding contributes for such large domains.

From the other point of view, the hybrid encoding is inferior to Mistral in the

interval $8 < MD \leq 64$. We found that the main reason is that the series of "Graph Coloring": the difference of Mistral and the hybrid encoding is indeed 22 instances. We will discuss it in Section 7.3.2.

Table 8. Maximum Domain based Comparison: each row denotes the interval of the maximum domain sizes. Table format is the same as Table 6.

Maximum Domain Size	#Ins.	Proposal (Hybrid)	Other CSP/SMT Solvers			
			Mistral	Opturion	Yices	Z3
$0 < \text{MD} \leq 2$	474	**415** (88%)	345 (73%)	344 (73%)	333 (70%)	335 (71%)
$2 < \text{MD} \leq 4$	114	**101** (89%)	94 (82%)	97 (85%)	92 (81%)	95 (83%)
$4 < \text{MD} \leq 8$	86	**82** (95%)	81 (94%)	77 (90%)	72 (84%)	71 (83%)
$8 < \text{MD} \leq 16$	115	103 (90%)	**108** (94%)	81 (70%)	57 (50%)	57 (50%)
$16 < \text{MD} \leq 32$	142	113 (80%)	**138** (97%)	108 (76%)	105 (74%)	99 (70%)
$32 < \text{MD} \leq 64$	107	74 (69%)	**83** (78%)	63 (59%)	56 (52%)	51 (48%)
$64 < \text{MD} \leq 128$	141	**102** (72%)	91 (65%)	94 (67%)	81 (57%)	62 (44%)
$128 < \text{MD} \leq 1024$	161	**134** (83%)	127 (79%)	125 (78%)	120 (75%)	113 (70%)
$1024 < \text{MD} \leq 8192$	90	**65** (72%)	31 (34%)	59 (66%)	43 (48%)	32 (36%)
$8192 < \text{MD}$	28	**27** (96%)	17 (61%)	23 (82%)	25 (89%)	26 (93%)
Total	1458	**1216** (83%)	1115 (76%)	1071 (73%)	984 (67%)	941 (65%)

Table 9 shows the number of solved instances organized by the categories of the CSP solver competition. The proposed hybrid encoding solved the largest number of instances for all category except "2-ARY-INT".

Interesting point is "Global" categories. Although the proposed hybrid encoding does not have special propagators for them, the result shows that the proposal is superior to other systems including the CSP solver Mistral. In addition, the virtual best solver VBS_{moyz} solves 27, 171, and 141 instances for Global1, 2, and 3. Thus, in total, the proposal solves more instances than VBS_{moyz} for those global categories. For "N-ARY-INT" category, a reason of the goodness of the hybrid encoding is the log encoding since longer constraints tend to have large domain products.

As same as the comparison using maximum domain sizes, the reason of the difference to Mistral comes from instances of "Graph Coloring". We will discuss it in Section 7.3.2.

Table 9. Category based Comparison: Each row denotes the categories of the CSP solver competition. Note that Global1, 2, and 3 denotes categories containing global constrains: 1) Alldiff, 2) Alldiff+Elt+Wsum, 3) Alldiff+Cumul+Elt+Wsum. Table format is the same as Table 6.

Category	#Ins.	Proposal (Hybrid)	Other CSP/SMT Solvers			
			Mistral	Opturion	Yices	Z3
2-ARY-INT	494	379 (77%)	**384** (78%)	342 (69%)	301 (61%)	281 (57%)
GLOBAL1	31	**30** (97%)	27 (87%)	14 (45%)	13 (42%)	11 (35%)
GLOBAL2	206	**179** (87%)	142 (69%)	110 (53%)	122 (59%)	101 (49%)
GLOBAL3	162	**140** (86%)	121 (75%)	137 (85%)	138 (85%)	133 (82%)
N-ARY-INT	565	**488** (86%)	441 (78%)	468 (83%)	410 (73%)	415 (73%)
Total	1458	**1216** (83%)	1115 (76%)	1071 (73%)	984 (67%)	941 (65%)

Table 10. Number of instances solved by each system. Each row denotes the series of instances. Asterisk * denotes the series that contain instances consisting of disjunctions (OR). The best figures are shown in bold face. Numbers in parentheses denote the percentage of the subtotal number of instances in each row.

Series	#Ins.	Proposal	Other CSP/SMT Solvers			
		(Hybrid)	Mistral	Opturion	Yices	Z3
2D Strip Packing	20	**8** (40%)	5 (25%)	4 (20%)	6 (30%)	6 (30%)
All Interval Series	15	9 (60%)	**15** (100%)	8 (53%)	7 (47%)	7 (47%)
BIBD	83	**79** (95%)	72 (87%)	24 (29%)	66 (80%)	32 (39%)
BMC	15	**15** (100%)	**15** (100%)	**15** (100%)	**15** (100%)	**15** (100%)
Chessboard Coloration	15	**12** (80%)	11 (73%)	10 (67%)	**12** (80%)	**12** (80%)
Cumulative Job-Shop	10	**4** (40%)	1 (10%)	3 (30%)	**4** (40%)	**4** (40%)
Domino	10	9 (90%)	**10** (100%)	**10** (100%)	**10** (100%)	**10** (100%)
Fischer	25	**24** (96%)	14 (56%)	21 (84%)	23 (92%)	23 (92%)
Golomb Ruler	28	**23** (82%)	**23** (82%)	20 (71%)	15 (54%)	16 (57%)
Graph Coloring	141	98 (70%)	**122** (87%)	81 (57%)	62 (44%)	60 (43%)
Haystacks	15	3 (20%)	**5** (33%)	0 (0%)	0 (0%)	0 (0%)
Job-Shop	76	**67** (88%)	51 (67%)	64 (84%)	61 (80%)	53 (70%)
Knights	10	**9** (90%)	7 (70%)	4 (40%)	**9** (90%)	**9** (90%)
Langford	43	27 (63%)	**32** (74%)	23 (53%)	24 (56%)	25 (58%)
Latin Square	10	**9** (90%)	6 (60%)	5 (50%)	5 (50%)	5 (50%)
Magic Square	18	**13** (72%)	8 (44%)	9 (50%)	3 (17%)	3 (17%)
Multi Knapsack	6	**6** (100%)	**6** (100%)	**6** (100%)	**6** (100%)	5 (83%)
NengFa	7	**7** (100%)	**7** (100%)	6 (86%)	6 (86%)	4 (57%)
Open-Shop	75	**71** (95%)	70 (93%)	**71** (95%)	62 (83%)	55 (73%)
Perfect Square Packing	74	**58** (78%)	42 (57%)	56 (76%)	57 (77%)	51 (69%)
Pigeons	29	23 (79%)	**29** (100%)	11 (38%)	9 (31%)	8 (28%)
Primes	76	70 (92%)	70 (92%)	**71** (93%)	53 (70%)	33 (43%)
Pseudo-Boolean	363	**312** (86%)	249 (69%)	297 (82%)	248 (68%)	282 (78%)
Quasigroup Existence	5	**5** (100%)	**5** (100%)	**5** (100%)	**5** (100%)	**5** (100%)
Queens	15	11 (73%)	**12** (80%)	9 (60%)	7 (47%)	6 (40%)
Queens-Knights	10	**10** (100%)	8 (80%)	**10** (100%)	9 (90%)	**10** (100%)
RCPSP	78	**78** (100%)	**78** (100%)	**78** (100%)	77 (99%)	**78** (100%)
Rader Surveillance	65	**65** (100%)	**65** (100%)	**65** (100%)	64 (98%)	**65** (100%)
Ramsey	16	**10** (63%)	**10** (63%)	9 (56%)	4 (25%)	5 (31%)
Schurr's Lemma	10	**9** (90%)	8 (80%)	8 (80%)	3 (30%)	2 (20%)
Social Golfers	10	**6** (60%)	**6** (60%)	**6** (60%)	1 (10%)	4 (40%)
Super-solutions	85	**66** (78%)	53 (62%)	62 (73%)	51 (60%)	48 (56%)
Total	1458	**1216** (83%)	1115 (76%)	1071 (73%)	984 (67%)	941 (65%)

7.3.2. *Total Performance Comparison between the Hybrid Encoding and other Systems*

Table 10 shows the number of instances solved by each solver, which is organized series-by-series.

The difference between the proposed hybrid encoding and other systems are made in the series of "Job-Shop", "Open-Shop", "Perfect Square Packing", "Super-solution" which contain many disjunctive constraints. In addition, the hybrid encoding is superior in the series of "Pseudo-Boolean". Since the good performance in those two types of series is the unique feature of the hybrid encoding and not

achieved by each of order/log encodings. We can say that the proposed hybridaization contributes to make pure SAT-based approaches competitive with those state-of-the art solvers.

On the other hand, Mistral solves the largest number of instances of "Graph Coloring". The difference made in UNSAT instances and the hybrid encoding solved 63 UNSAT instances while Mistral solved 86 UNSAT instances. It is considered that AC3-bitset[49] in Mistral is effective for those instances.

In order to improve the hybrid encoding to solve more instances of "Graph Coloring", an idea is to hybridize the direct encoding[54,55] since those instances consist of a lot of $\neq$ constraints and the direct encoding is known to be good at handling them. In addition, our proposed hybrid encoding framework is capable to hybridize it.

At last, we conclude this section with the overall evaluation using a cactus plot—the relation between the cumulative number of solved instances and the CPU time over all solvers Fig. 3 shows that the hybrid encoding solved the largest number of instances in any cut-off time.

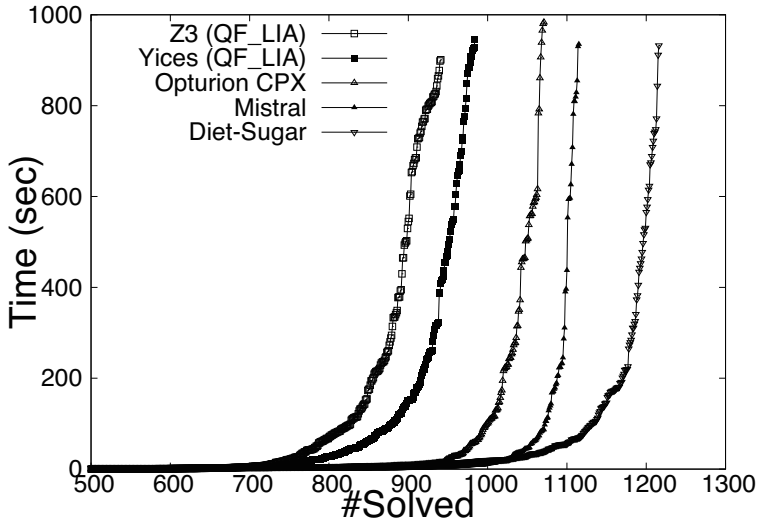

Fig. 3. Cumulative number of instances solved versus CPU time by state-of-the-art systems.

8. Conclusion

This paper proposes a new hybrid encoding of CSP to SAT. Contributions of this paper can be summarized as follows.

- Proposal of a new hybrid encoding integrating the order and log encodings:

The proposed encoding is able to take the advantages of the both encodings and has the following two features. First, it is channeling constraint free which enables us to effectively hybridize both encodings. Second, it can encode each constraint using both the order-encoded and log-encoded variables (Section 3 and Section 4.1).
- Proposal of the domain product as a criterion of the hybridization: The "degree" of hybridization can be controlled by classifying the order-encoded and log-encoded variables. We propose to use the domain product criterion for that classification which is superior than the domain size criterion when constraints consist of large sized arity (Section 4.2).

We implemented the proposed hybrid encoding as the Diet-Sugar system (Section 5) and confirmed the effect of the proposal by the following two evaluations (Section 6 and Section 7).

- Evaluation of the effectiveness of the proposed hybridization: We made the comparison with three encodings. Using 1458 competition benchmarks, we confirm that the hybrid encoding is superior than both the order and log encodings. It indeed solves the largest number of instances 1216 with the shortest CPU time while the order encoding solves 1161 and the log encoding solves 1170 instances (Fig. 1 and Fig. 2). We also confirmed that the better encoding is switched from the order encoding to the log encoding while the domain product varies from small to large, which demonstrates the appropriateness of the domain product criterion (Table. 2 and Table. 3).
- Evaluation of our implementation Diet-Sugar through the comparison with other state-of-the-art CSP and SMT solvers: We confirmed that Diet-Sugar solves the largest number of instances 1216 with the shortest CPU time while Mistral, Opturion CPX, Yices, and Z3 solve 1115, 1071, 984, and 941 instances, respectively (Fig. 3). Diet-Sugar is especially superior than other solvers for instances containing disjunctive constraints and global constraints—it solves more instances than the virtual best solver consisting of the four state-of-the-art systems (Table 7 and Table 9).

Our framework in Section 3.1 could be extended to hybridize other existing SAT encodings and hybrid SAT encodings. Proposing and comparing such encodings are our future work. Comparisons using benchmarks including extensional constraints are also important research topics. In addition, using state-of-the-art PB solvers such as winning solvers in the latest pseudo-Boolean competition would be promising to improve our approach.

All implementations and data of experimental results are available in the supplemental web page.

References

1. A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications (FAIA), vol. 185. IOS Press, 2009.
2. N. Tamura, A. Taga, S. Kitagawa, and M. Banbara, "Compiling finite linear CSP into SAT," *Constraints*, vol. 14, no. 2, pp. 254–272, 2009.
3. T. Soh, N. Tamura, and M. Banbara, "Scarab: A rapid prototyping tool for SAT-based constraint programming systems," in *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT 2013), LNCS 7962*, Jul. 2013, pp. 429–436.
4. T. Tanjo, N. Tamura, and M. Banbara, "Azucar: A SAT-based CSP solver using compact order encoding — (tool presentation)," in *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT 2012), LNCS 7317*. Springer, Jun. 2012, pp. 456–462.
5. A. Metodi and M. Codish, "Compiling finite domain constraints to SAT with BEE," *Theory and Practice of Logic Programming*, vol. 12, no. 4-5, pp. 465–483, 2012.
6. E. Hebrard, E. O'Mahony, and B. O'Sullivan, "Constraint programming and combinatorial optimisation in Numberjack," in *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2010), LNCS 6140*, 2010, pp. 181–185.
7. N. Zhou, "Combinatorial search with picat," *CoRR*, vol. abs/1405.2538, 2014.
8. M. Stojadinovic and F. Maric, "meSAT: multiple encodings of CSP to SAT," *Constraints*, vol. 19, no. 4, pp. 380–403, 2014.
9. S. D. Prestwich, "CNF encodings," in *Handbook of Satisfiability*, 2009, pp. 75–97.
10. D. E. Knuth, "Satisfiability," in *Fascicle 6 of The Art of Computer Programming*. Addison-Wesley Professional, 2015, vol. 4.
11. M. Cadoli, G. Ianni, L. Palopoli, A. Schaerf, and D. Vasile, "NP-SPEC: an executable specification language for solving all problems in NP," *Comput. Lang.*, vol. 26, no. 2-4, pp. 165–195, 2000.
12. O. Ohrimenko, P. J. Stuckey, and M. Codish, "Propagation via lazy clause generation," *Constraints*, vol. 14, no. 3, pp. 357–391, 2009.
13. I. Abío and P. J. Stuckey, "Encoding linear constraints into SAT," in *Proceedings of the 20th International Joint Conference on Principles and Practice of Constraint Programming (CP 2014), LNCS 8656*, 2014, pp. 75–91.
14. J. M. Crawford and A. B. Baker, "Experimental results on the application of satisfiability algorithms to scheduling problems," in *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994)*, 1994, pp. 1092–1097.
15. O. Bailleux and Y. Boufkhad, "Efficient CNF encoding of Boolean cardinality constraints," in *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP 2003), LNCS 2833*, 2003, pp. 108–122.
16. C. Ansótegui and F. Manyà, "Mapping problems with finite-domain variables into problems with boolean variables," in *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*, 2004.
17. I. P. Gent and P. Nightingale, "A new encoding of alldifferent into SAT," in *Proceedings of the 3rd International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, 2004.
18. C. Sinz, "Towards an optimal CNF encoding of Boolean cardinality constraints," in *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP 2005), LNCS 3709*, 2005, pp. 827–831.

19. J. Petke and P. Jeavons, "The order encoding: From tractable csp to tractable sat," in *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT 2011), LNCS 6695*, 2011, pp. 371–372.
20. T. Soh, K. Inoue, N. Tamura, M. Banbara, and H. Nabeshima, "A SAT-based method for solving the two-dimensional strip packing problem," *Fundamenta Informaticae*, vol. 102, no. 3-4, pp. 467–487, 2010.
21. M. Banbara, H. Matsunaka, N. Tamura, and K. Inoue, "Generating combinatorial test cases by efficient SAT encodings suitable for CDCL SAT solvers," in *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-17), LNCS 6397*, 2010, pp. 112–126.
22. M. Heule and S. Szeider, "A SAT approach to clique-width," in *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, 2013, pp. 318–334.
23. C. Lecoutre, O. Roussel, and M. R. C. van Dongen, "Promoting robust black-box solvers through competitions," *Constraints*, vol. 15, no. 3, pp. 317–326, 2010.
24. K. Iwama and S. Miyazaki, "SAT-variable complexity of hard combinatorial problems," in *Proceedings of the IFIP 13th World Computer Congress*, 1994, pp. 253–258.
25. A. V. Gelder, "Another look at graph coloring via propositional satisfiability," *Discrete Applied Mathematics*, vol. 156, no. 2, pp. 230–243, 2008.
26. V. Nguyen, M. N. Velev, and P. Barahona, "Application of hierarchical hybrid encodings to efficient translation of CSPs to SAT," in *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2013), IEEE*, 2013, pp. 1028–1035.
27. J. Argelich, C. M. Li, F. Manyà, and Z. Zhu, "Many-valued minsat solving," in *IEEE 44th International Symposium on Multiple-Valued Logic, ISMVL 2014, Bremen, Germany, May 19-21, 2014*, 2014, pp. 32–37.
28. A. M. Frisch and T. J. Peugniez, "Solving non-boolean satisfiability problems with stochastic local search," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, 2001, pp. 282–290.
29. I. Griva, S. G. Nash, and A. Sofer, *Linear and Nonlinear Optimization*. Society for Industrial Mathematics, 2009.
30. N. Eén and N. Sörensson, "Translating pseudo-Boolean constraints into SAT," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, no. 1-4, pp. 1–26, 2006.
31. O. Bailleux, Y. Boufkhad, and O. Roussel, "A translation of pseudo Boolean constraints to SAT," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, no. 1-4, pp. 191–200, 2006.
32. I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and V. Mayer-Eichberger, "A new look at BDDs for pseudo-Boolean constraints," *Journal of Artificial Intelligence Research*, vol. 45, pp. 443–480, 2012.
33. N. Tamura, M. Banbara, and T. Soh, "PBSugar: Compiling pseudo-boolean constraints to SAT with order encoding," in *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2013), IEEE*, Nov. 2013, pp. 1020–1027.
34. G. S. Tseitin, "On the complexity of derivations in the propositional calculus," *Studies in Mathematics and Mathematical Logic Part II*, pp. 115–125, 1968.
35. D. L. Berre and A. Parrain, "The Sat4j library, release 2.2," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, no. 2-3, pp. 59–64, 2010.
36. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "Conflict-driven answer set solving," in *Proceedings of the 20th International Joint Conference on Artificial Intel-*

ligence (IJCAI 2007), 2007, pp. 386–392.
37. V. M. Manquinho and J. P. M. Silva, "On using cutting planes in pseudo-Boolean optimization," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, no. 1-4, pp. 209–219, 2006.
38. V. M. Manquinho, R. Martins, and I. Lynce, "Improving unsatisfiability-based algorithms for Boolean optimization," in *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT 2010), LNCS 6175*, 2010, pp. 181–193.
39. G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 2009, pp. 399–404.
40. A. Biere, "Lingeling essentials, A tutorial on design and implementation aspects of the the SAT solver lingeling," in *POS-14. Fifth Pragmatics of SAT workshop, a workshop of the SAT 2014 conference, part of FLoC 2014 during the Vienna Summer of Logic, July 13, 2014, Vienna, Austria*, 2014, p. 88.
41. H. Nabeshima, K. Iwanuma, and K. Inoue, "Glueminisat 2.2.8," in *Proceedings of SAT Competition 2014*, 2014, pp. 35–36.
42. T. Soh, M. Banbara, and N. Tamura, "A hybrid encoding of CSP to SAT integrating order and log encodings," in *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*, 2015, pp. 421–428.
43. D. L. Berre and A. Parrain, "On extending sat solvers for pb problems," in *In 14th RCRA workshop Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA 07*, 2007.
44. H. Dixon, "Automating pseudo-boolean inference within a dpll framework," Ph.D. dissertation, University of Oregon, Eugene, OR, USA, 2004, aAI3153782.
45. D. Chai and A. Kuehlmann, "A fast pseudo-boolean constraint solver," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 24, no. 3, pp. 305–317, 2005. [Online]. Available: *http://dx.doi.org/10.1109/TCAD.2004.842808*
46. *M. Gebser, B. Kaufmann, and T. Schaub, "Conflict-driven answer set solving: From theory to practice,"* Artif. Intell.*, vol. 187, pp. 52–89, 2012.*
47. *M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "On the implementation of weight constraint rules in conflict-driven ASP solvers," in* Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings*, 2009, pp. 250–264.*
48. *E. Hebrard, "Mistral, a constraint satisfaction library," 2008, pp. 31–39.*
49. *C. Lecoutre and J. Vion, "Enforcing arc consistency using bitwise operations,"* Constraint Programming Letters*, vol. 2, pp. 21–35, 2012.*
50. *P. J. Stuckey, M. J. G. de la Banda, M. J. Maher, K. Marriott, J. K. Slaney, Z. Somogyi, M. Wallace, and T. Walsh, "The G12 project: Mapping solver independent models to efficient solutions," in* Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, 2005, pp. 13–16.*
51. *B. Dutertre, "Yices 2.2," in* Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, 2014, pp. 737–744.*
52. *L. M. de Moura and N. Bjørner, "Z3: an efficient SMT solver," in* Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings,

2008, pp. 337–340.

53. *C. Barrett, A. Stump, and C. Tinelli, "The SMT-LIB standard: Version 2.0," in* Satisfiability Modulo Theories (SMT 2010)*, A. Gupta and D. Kroening, Eds., Edinburgh, UK, 2010, http://www.SMT-LIB.org*

    .

54. *J. de Kleer, "A comparison of ATMS and CSP techniques," in* Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI 1989)*, 1989, pp. 290–296.*
55. *T. Walsh, "SAT v CSP," in* Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP 2000)*, 2000, pp. 441–456.*