Proposal of a compact and efficient SAT encoding using a numeral system of any base

Tomoya Tanjo[1], Naoyuki Tamura[2], and Mutsunori Banbara[2]

[1] Graduate School of Engineering, Kobe University, JAPAN
[2] Information Science and Technology Center, Kobe University, JAPAN
`tanjo@stu.kobe-u.ac.jp, tamura@kobe-u.ac.jp, banbara@kobe-u.ac.jp`

Abstract. This paper describes a new SAT encoding method, named *compact order encoding*, applicable to finite domain CSP. It is a generalization of log encoding (compact encoding) and order encoding which is adopted by an award-winning SAT-based CSP solver. The basic idea of the compact order encoding is the use of a numeral system of some base. Each integer variable is divided into some digits and each digit is encoded by using the order encoding. It generates much smaller SAT instances than the order encoding and more efficient SAT instances than the log encoding in general because it requires fewer carry propagations when using larger base than two. We confirmed these observations through the experimental results on some benchmarks, such as Open-Shop Scheduling problems. Especially for very large instances, our solver with MiniSat engine outperformed other encodings and the state-of-the-art CSP solvers choco and Mistral.

1 Introduction

A (finite) Constraint Satisfaction Problem (CSP) is a combinatorial problem to find an assignment which satisfies all given constraints on finite domain variables [22]. A SAT-based CSP solver is a program which solves a CSP by encoding it to SAT [21] and searching solutions by SAT solvers. Remarkable improvements in the efficiency of SAT solvers make SAT-based CSP solvers applicable for solving hard and practical problems. A number of SAT encoding methods have been therefore proposed: direct encoding [6, 28, 21], support encoding [18, 12, 21], log encoding [17, 10, 21], log-support encoding [9], and order encoding [24, 25].

Among them, the *order encoding* [25] was first used to encode Job-Shop Scheduling problems by Crawford and Baker [5] and studied by Inoue *et al* [16, 20]. Its effectiveness has been shown by the fact that a SAT-based CSP solver Sugar [3] became a winner in several categories of the 2008 and 2009 International CSP Solver Competitions. It has also showed a good performance for a wide variety of problems, including Open-Shop Scheduling problems [25], two-dimensional strip packing problems [23], and test case generation [1].

The effectiveness of the order encoding is due to the realization of fast propagations. In the order encoding, a propositional variable $p(x \leq a)$ is used for

[3] `http://bach.istc.kobe-u.ac.jp/sugar/`

each integer variable x and its domain value a, where $p(x \le a)$ is defined as true if and only if the comparison $x \le a$ holds. The constraint $x \le y$ is encoded into the clauses of $p(x \le a) \lor \neg p(y \le a)$ for all domain value a. When $p(y \le a)$ becomes true for some a, the Unit Propagation of SAT solver immediately derives $p(x \le a)$. This inference corresponds to the Bounds Propagation in CSP solvers.

However, in the order encoding, the size of SAT-encoded instances becomes huge when the domain size of the original CSP is large. For example, the number of clauses required by n-ary constraint $\sum_{i=1}^{n} a_i x_i \le b$ will be $O(d^{n-1})$ where d is the maximum domain size of x_i's. On the other hand, the *log encoding* [17, 10] uses a bit-wise representation for integer variables. The size of SAT-encoded instances is therefore compact (linear to $\log d$), but its performance is slow in general because it requires many inference steps to "ripple" carries.

In this paper, we propose a new encoding, named *compact order encoding*, aiming to be compact and efficient. The basic idea of the compact order encoding is the use of a numeral system of base $B \ge 2$. That is, each integer variable x is represented by a summation $\sum_{i=0}^{m-1} B^i x_i$ where $m = \lceil \log_B d \rceil$ and $0 \le x_i < B$ for all x_i, and each x_i is encoded by the order encoding. Note that the compact order encoding with base $B = 2$ is equivalent to the log encoding, and the one with base $B \ge d$ is equivalent to the order encoding.

In our experiments, we used Open-Shop Scheduling problems as benchmarks and showed that the compact order encoding can be more efficient than the order encoding, the log encoding, and the state-of-the-art CSP solvers choco and Mistral. Moreover, we showed that the compact order encoding can be highly scalable with the growth of domain size for very large instances not solved by other encodings and CSP solvers.

In encoding design, we need to keep the arity of constraints to be small in addition to limiting the bounds of integer variables because simple replacement of integer variables with summations of digit variables leads to larger SAT instances than the order encoding. Therefore, we first define *3ary-CSP* containing only ternary (including unary and binary) constraints in Section 2. Then, in Section 3, we introduce *Compact 3ary-CSP* which is a 3ary-CSP such that the domain sizes of variables are bounded by the base B. Section 4 presents a reduction of 3ary-CSP to Compact 3ary-CSP. Section 5 shows how to realize the whole SAT encoding of the given 3ary-CSP. Section 6 shows our experimental results. Section 7 presents some related works. The paper is concluded in Section 8. Our first attempt towards the compact order encoding is reported in [26]. In this paper, we present the further development of the compact order encoding including an extension to non-linear constraints and show some new results comparing with other methods.

2 3ary-CSP

Let $\mathbb{Z}$ and $\mathbb{N}$ be the set of integers and non-negative integers respectively. $\mathbb{B} = \{\mathit{false}, \mathit{true}\}$ is used to denote a set of propositional constants. The following is the definition of (finite domain arithmetic) **3ary-CSP** which only consists of the constraints containing at most three non-negative variables.

Definition 1 (3ary-CSP) A (finite domain arithmetic) 3ary-CSP is defined as a tuple (X, u, P, C) satisfying the followings.

- X is a finite set of integer variables.
- u is a mapping from X to $\mathbb{N}$ representing the upper bound of each integer variable in X (the lower bound is fixed to 0).
- P is a finite set of propositional variables.
- C is a formula representing the constraint to be satisfied. The syntax of C is defined as follows, where $p \in P$, $n \in \{1, 2, 3\}$, $x_i, x, y, z \in X$, $a_i \in \mathbb{Z}$, $b \in \mathbb{N}$, and $\rhd \in \{\leq, \geq, =\}$.

$$C ::= \ p \ \mid \ \neg p \ \mid \ \sum_{i=1}^{n} a_i x_i \rhd b \ \mid \ z = xy \ \mid \ C \wedge C \ \mid \ C \vee C$$

Any finite domain CSP can be reduced to 3ary-CSP in the following way.

- Any extensional constraint can be expressed by the above arithmetic constraints in linear size. For example, x and y are variables whose domains are $\{a_1, a_2, a_3\}$ and $R = \{(a_1, a_2), (a_2, a_3)\}$ is an extensional constraint of support tuples on x and y. By assigning integer value i for each a_i, R can be expressed by an arithmetic constraint $(x = 1 \wedge y = 2) \vee (x = 2 \wedge y = 3)$.
- When the lower bound l of an integer variable x is not 0, x can be replaced by a new integer variable x' satisfying $x' = x - l$.
- A constraint $\sum_i a_i \prod_j x_{ij} \rhd b$ is reduced to a summation $\sum_i a_i z_i \rhd b$ and multiplications $z_i = \prod_j x_{ij}$ by introducing new variables z_i.
- A multiplication of more than three variables can be reduced to the multiplications of two variables by replacing partial products with new variables.
- A linear expression containing more than three variables can be reduced to ternary expressions by replacing partial summations with new variables.

An *assignment* of a CSP (X, u, P, C) is a pair (α, β) where α is a mapping from X to $\mathbb{N}$ and β is a mapping from P to $\mathbb{B}$. When there exists an assignment (α, β) which satisfies the formula C and $\alpha(x) \leq u(x)$ for any $x \in X$, the CSP is called *satisfiable* and the assignment is called a *solution* of the CSP. We use the notation $(\alpha, \beta) \models C$ to represent the satisfiability. Sometimes, we use the same notation $(\alpha, \beta) \models C$ for any formula C not limited to CSP formulas. We also write $(\alpha, \beta) \models (X, u, P, C)$ when the CSP is satisfiable by the assignment.

2.1 Restricted 3ary-CSP

We introduce a restricted form of CSP called **Restricted 3ary-CSP** (R-CSP) to make the reduction of CSP to Compact 3ary-CSP described in Section 4 easily understandable.

x_1	x_0	Satisfiable
0–1	0–9	Yes
2	0–6	Yes
2	7–9	No
3–9	0–9	No

Fig. 1. The satisfying values of $(x_1 \leq 2) \wedge (x_1 \leq 1 \vee x_0 \leq 6)$

$$\begin{array}{ccc} & x_1 & x_0 \\ \times & & 9 \\ \hline & v_{01} & v_{00} \\ v_{11} & v_{10} & \\ \hline z_2 & z_1 & z_0 \end{array}$$

Fig. 2. A long multiplication of $100z_2 + 10z_1 + z_0 = 9(10x_1 + x_0)$

Definition 2 (Restricted 3ary-CSP) A Restricted 3ary-CSP (R-CSP) is a CSP (X, u, P, C) where the formula C is restricted in the following forms where $p \in P$, $x, y, z \in X$, and $a \in \mathbb{N}$.

$$\begin{aligned} C ::= p \mid \neg p \mid x \leq a \mid x \geq a \mid x \leq y \mid z = x + a \mid z = x + y \\ \mid z = ax \mid z = xy \mid C \wedge C \mid C \vee C \end{aligned}$$

Formulas allowed in R-CSP are very limited, but any CSP formulas can be reduced to the restricted forms as follows.

Lemma 1 Any CSP (X, u, P, C) can be reduced to an R-CSP.

It is shown by verifying the reductions for all cases of $\sum_{i=1}^{n} a_i x_i \rhd b$ (that is, for all combinations of comparison operators and signs of each a_i).

Any solution of the R-CSP can be translated back to a solution of the original CSP by simply removing the assignments of newly introduced variables.

3 Compact 3ary-CSP

Definition 3 (Compact 3ary-CSP) Let $B \geq 2$ be an integer constant. A Compact 3ary-CSP (C-CSP) $(B; X, P, C)$ is a CSP (X, u, P, C) that satisfies:

- $u(x) = B - 1$ for any integer variable $x \in X$, and
- the formula C is restricted in the following forms where $p \in P$, $n \in \{1, 2, 3\}$, $x_i, x, y, z \in X$, $0 \leq a < B$, $0 \leq b \leq 3(B - 1)$, and $\rhd \in \{\leq, \geq, =\}$.

$$C ::= \ p \ \mid \ \neg p \ \mid \ \sum_{i=1}^{n} \pm x_i \rhd b \ \mid \ By + z = ax \ \mid \ C \wedge C \ \mid \ C \vee C$$

C-CSP contains no nonlinear constraints and the upper bounds of integer variables are fixed to an integer constant $B - 1$ where $B \geq 2$ is called a *base*.

Example 1. Let us consider a C-CSP $(10; \{x_1, x_0\}, \emptyset, C)$ where $C = (x_1 \leq 2) \wedge (x_1 \leq 1 \vee x_0 \leq 6)$. Its satisfiability can be summarized in Fig. 1. Therefore, it is equi-satisfiable to $10x_1 + x_0 \leq 26$.

Example 2. Let us consider a C-CSP $(10; \{x_1, x_0, z_1, z_0\}, \{c\}, C)$ where $C = (c \vee z_1 = x_1 + 2) \wedge (\neg c \vee z_1 = x_1 + 3) \wedge (c \vee z_0 = x_0 + 6) \wedge (\neg c \vee z_0 = x_0 - 4)$. It is easy to confirm it is equi-satisfiable to $10z_1 + z_0 = 10x_1 + x_0 + 26$. When $c = \mathit{false}$, $C = (z_1 = x_1 + 2 \wedge z_0 = x_0 + 6)$ derives $10z_1 + z_0 = 10x_1 + x_0 + 26$. The same conclusion can be derived also when $c = \mathit{true}$. The propositional variable c corresponds to a carry bit of $x_0 + 6$.

Example 3. Let us consider a following C-CSP $(10; X, \{c\}, C)$.

$$\begin{aligned}
X &= \{x_1, x_0, z_2, z_1, z_0, v_{11}, v_{10}, v_{01}, v_{00}\} \\
C &= (10v_{11} + v_{10} = 9x_1) \wedge (10v_{01} + v_{00} = 9x_0) \\
&\quad \wedge (c \vee z_2 = v_{11}) \wedge (\neg c \vee z_2 = v_{11} + 1) \\
&\quad \wedge (c \vee z_1 = v_{01} + v_{10}) \wedge (\neg c \vee z_1 = v_{01} + v_{10} - 10) \wedge (z_0 = v_{00})
\end{aligned}$$

By considering a long multiplication as shown in Fig. 2, it is easy to confirm it is equi-satisfiable to $100z_2 + 10z_1 + z_0 = 9(10x_1 + x_0)$. The propositional variable c corresponds to a carry bit of $v_{01} + v_{10}$.

4 Reducing CSP into C-CSP

As described in Section 2.1, any CSP can be reduced to an R-CSP. We consider reducing an R-CSP into a C-CSP for any base $B \geq 2$. Before that, we introduce some notations.

The *maximum domain size* d and its *order of magnitude* m are defined as follows.

$$\begin{aligned}
d &= 1 + \max(\{u(x) \mid x \in X\} \cup \{a \mid a \text{ is an integer constant occurs in } C\}) \\
m &= \lceil \log_B d \rceil
\end{aligned}$$

In the reduction, we need to introduce a new integer variable of C-CSP for each digit of each integer variable x of the R-CSP. We define $x^{(i)}$ as a syntactic function which generates a new integer variable symbol of C-CSP from an integer variable symbol x of R-CSP for each integer i $(0 \leq i < m)$. The function $x^{(i)}$ is intended to represent the i-th digit of x, that is, the following equation is kept in mind.

$$x^{(i)} = (x \operatorname{div} B^i) \bmod B$$

We also use the same notation $a^{(i)}$ for any integer constant $a \in \mathbb{N}$ of R-CSP. The notation of $x^{(j,i)}$ is defined as $\sum_{k=i}^{j} B^{k-i} x^{(k)}$ where x is a variable or constant.

Recall that the constraint C in R-CSP includes the following comparisons: $x \leq a$, $x \geq a$, $x \leq y$, $z = x + a$, and $z = x + y$. We only focus on the reduction of $x \leq y$, $z = x + y$ and $z = xy$ in the remaining subsections since other formulas can be reduced in a similar way.

4.1 Reduction of $x \le y$

Any formula $x \le y$ of R-CSP can be represented as $x^{(m-1,0)} \le y^{(m-1,0)}$ by using integer variables of targeted C-CSP. We show a method to reduce it into a C-CSP formula without changing the upper bounds of the integer variables [26].

Definition 4 (Reduction of $x \le y$) Let x and y be integer variables or constants of an R-CSP. A syntactic translation $\tau(x, y)$ is defined as follows.

$$\begin{aligned}
\tau(x, y) &:= \tau_{m-1}(x, y) \\
\tau_0(x, y) &:= x^{(0)} \le y^{(0)} \\
\tau_i(x, y) &:= x^{(i)} \le y^{(i)} \wedge (x^{(i)} \le y^{(i)} - 1 \vee \tau_{i-1}(x, y)) \qquad (i > 0)
\end{aligned}$$

Proposition 1 The following holds for any integer variables or constants $x^{(i)}$ and $y^{(i)}$ of a C-CSP.

$$x^{(m-1,0)} \le y^{(m-1,0)} \iff \tau(x, y)$$

It can be proved by induction on m.

Example 4. The following is an example of $\tau_2(x, y)$ which is equivalent to $x^{(2,0)} \le y^{(2,0)}$, i.e. $B^2x^{(2)} + Bx^{(1)} + x^{(0)} \le B^2y^{(2)} + By^{(1)} + y^{(0)}$.

$$\begin{aligned}
&\tau_2(x, y) \\
&= x^{(2)} \le y^{(2)} \wedge (x^{(2)} \le y^{(2)} - 1 \vee \tau_1(x, y)) \\
&= x^{(2)} \le y^{(2)} \wedge (x^{(2)} \le y^{(2)} - 1 \vee (x^{(1)} \le y^{(1)} \wedge (x^{(1)} \le y^{(1)} - 1 \vee \tau_0(x, y)))) \\
&= x^{(2)} \le y^{(2)} \wedge (x^{(2)} \le y^{(2)} - 1 \vee (x^{(1)} \le y^{(1)} \wedge (x^{(1)} \le y^{(1)} - 1 \vee x^{(0)} \le y^{(0)})))
\end{aligned}$$

4.2 Reduction of $z = x + y$

Any formula $z = x + y$ of R-CSP can be represented as $z^{(m-1,0)} = x^{(m-1,0)} + y^{(m-1,0)}$ by using integer variables of targeted C-CSP. We show a method to reduce it into an equi-satisfiable ternary constraint without changing the upper bounds of the integer variables [26].

Definition 5 (Reduction of $z = x + y$) Let $B \ge 2$ be a base, z, x and y be integer variables or constants of an R-CSP, and c_i $(0 \le i \le m)$ be propositional variables not occurring in other places. A syntactic translation $\sigma(z, x, y)$ is defined as follows.

$$\begin{aligned}
\sigma(z, x, y) \;:=\; & \neg c_0 \wedge \neg c_m \\
\wedge \; & \bigwedge_{i=0}^{m-1} ((c_i \vee c_{i+1} \vee z^{(i)} = x^{(i)} + y^{(i)}) \\
& \quad \wedge (c_i \vee \neg c_{i+1} \vee z^{(i)} = x^{(i)} + y^{(i)} - B) \\
& \quad \wedge (\neg c_i \vee c_{i+1} \vee z^{(i)} = x^{(i)} + y^{(i)} + 1) \\
& \quad \wedge (\neg c_i \vee \neg c_{i+1} \vee z^{(i)} = x^{(i)} + y^{(i)} + 1 - B))
\end{aligned}$$

Note that $\sigma(z, x, y)$ only contains ternary constraints of C-CSP and its size is linear to m.

Proposition 2 Let $B \geq 2$ be a base, $z^{(i)}$, $x^{(i)}$ and $y^{(i)}$ be integer variables or constants of a C-CSP. Then the following holds for any α.

$$(\alpha, \emptyset) \models z^{(m-1,0)} = x^{(m-1,0)} + y^{(m-1,0)} \iff \exists\beta.(\alpha, \beta) \models \sigma(z, x, y)$$

Proof. By considering digit-wise addition, it is easy to confirm that $z^{(m-1,0)} = x^{(m-1,0)} + y^{(m-1,0)}$ is equi-satisfiable with $c'_0 = 0 \wedge c'_m = 0 \wedge \bigwedge_{i=0}^{m-1} Bc'_{i+1} + z^{(i)} = x^{(i)} + y^{(i)} + c'_i$ for the same assignments for $z^{(i)}$, $x^{(i)}$, $y^{(i)}$ and some assignments for $c'_i \in \{0, 1\}$ $(0 \leq i \leq m)$. By considering all cases of $c'_i = 0$ or 1 and replacing c'_i with propositional variables c_i, we can show $z^{(m-1,0)} = x^{(m-1,0)} + y^{(m-1,0)}$ is equi-satisfiable with $\sigma(z, x, y)$ for the same assignments for $z^{(i)}$, $x^{(i)}$, and $y^{(i)}$.
□

Example 5. The following is an example of $\sigma(z, x, y)$ when the base $B = 2$ and $u(z) = u(x) = u(y) = 7$. It represents an addition of 3-bits integers.

$$\begin{aligned}
&\sigma(z, x, y) \;= \\
&\quad \neg c_0 \wedge \neg c_3 \\
&\wedge (c_0 \vee c_1 \vee z^{(0)} = x^{(0)} + y^{(0)}) \wedge (c_0 \vee \neg c_1 \vee z^{(0)} = x^{(0)} + y^{(0)} - 2) \\
&\wedge (\neg c_0 \vee c_1 \vee z^{(0)} = x^{(0)} + y^{(0)} + 1) \wedge (\neg c_0 \vee \neg c_1 \vee z^{(0)} = x^{(0)} + y^{(0)} - 1) \\
&\wedge (c_1 \vee c_2 \vee z^{(1)} = x^{(1)} + y^{(1)}) \wedge (c_1 \vee \neg c_2 \vee z^{(1)} = x^{(1)} + y^{(1)} - 2) \\
&\wedge (\neg c_1 \vee c_2 \vee z^{(1)} = x^{(1)} + y^{(1)} + 1) \wedge (\neg c_1 \vee \neg c_2 \vee z^{(1)} = x^{(1)} + y^{(1)} - 1) \\
&\wedge (c_2 \vee c_3 \vee z^{(2)} = x^{(2)} + y^{(2)}) \wedge (c_2 \vee \neg c_3 \vee z^{(2)} = x^{(2)} + y^{(2)} - 2) \\
&\wedge (\neg c_2 \vee c_3 \vee z^{(2)} = x^{(2)} + y^{(2)} + 1) \wedge (\neg c_2 \vee \neg c_3 \vee z^{(2)} = x^{(2)} + y^{(2)} - 1)
\end{aligned}$$

4.3 Reduction of $z = xy$

Any formula $z = xy$ of R-CSP can be represented as $z^{(m-1,0)} = x^{(m-1,0)}y^{(m-1,0)}$ by using integer variables of targeted C-CSP. Before considering the reduction of a general case, we explain the reduction of a special case $z = ay$ $(0 \leq a < B)$.

Reduction of a Special Case $z = ay$ $(0 \leq a < B)$ We decompose $z = ay$ with respect to each digit of y and represent each $ay^{(i)}$ as v_i. Then $z = ay$ can be represented as follows.

$$\bigwedge_{i=0}^{m-1} (v_i = ay^{(i)}) \;\wedge\; z = \sum_{i=0}^{m-1} B^i v_i$$

Each v_i is represented by $B{v_i}^{(1)} + {v_i}^{(0)}$ since $ay^{(i)} < B^2$.

Definition 6 (Reduction of $z = ay$ $(0 \le a < B)$) Let $B > 2$ be a base, z, y be integer variables or constants of an R-CSP, $0 \le a < B$ be an integer constant and v_i $(0 \le i < m)$ be integer variables not occurring in other places. A syntactic translation $\nu(z, a, y)$ is defined as follows.

$$\nu(z, a, y) := \bigwedge_{i=0}^{m-1} (B{v_i}^{(1)} + {v_i}^{(0)} = ay^{(i)}) \wedge \left[z = \sum_{i=0}^{m-1} B^i v_i \right]$$

where $[e]$ is a syntactic function reducing e by iteratively using the Proposition 2 and digitwise left shift operations.

Lemma 2 Let $B > 2$ be a base, $z^{(i)}$ and $y^{(i)}$ be integer variables or constants of a C-CSP, and $0 \le a < B$ be an integer constant. Then the following holds for any α where α' is an assignment containing only new integer variables introduced by ν.

$$(\alpha, \emptyset) \models z^{(m-1,0)} = ay^{(m-1,0)} \iff \exists \alpha'. \exists \beta. (\alpha \cup \alpha', \beta) \models \nu(z, a, y)$$

Proof. We omit the proof because it is easily confirmed.

Reduction of a General Case of $z = xy$ We decompose $z = xy$ with respect to each digit of x and represent each $x^{(i)}y$ as w_i. Then $z = xy$ can be represented as follows.

$$\bigwedge_{i=0}^{m-1} (w_i = x^{(i)}y) \ \wedge \ z = \sum_{i=0}^{m-1} B^i w_i$$

Each $w_i = x^{(i)}y$ can be represented by the following formula because each $x^{(i)}$ takes a value from 0 to $B - 1$ and $w_i = ay$ when $x^{(i)} = a$.

$$\bigwedge_{a=0}^{B-1} \Big((x^{(i)} \le a - 1) \vee (x^{(i)} \ge a + 1) \vee (w_i = ay) \Big)$$

The calculation of ay occurs repeatedly for various i. To eliminate the redundant calculations, we introduce a new variable $y_a = ay$ for each a. Then $z = xy$ is represented as follows.

$$\begin{aligned} &\bigwedge_{i=0}^{m-1} \bigwedge_{a=0}^{B-1} \Big((x^{(i)} \le a - 1) \vee (x^{(i)} \ge a + 1) \vee (w_i = y_a) \Big) \\ \wedge\ & z = \sum_{i=0}^{m-1} B^i w_i \wedge \bigwedge_{a=0}^{B-1} y_a = ay \end{aligned}$$

Each $w_i = y_a$ can be reduced to $\bigwedge_{j=0}^{m-1} ({w_i}^{(j)} = {y_a}^{(j)})$, and each $y_a = ay$ can be reduced to $\nu(y_a, a, y)$.

Definition 7 (Reduction of $z = xy$) Let $B > 2$ be a base, z, x and y be integer variables or constants of an R-CSP, and y_a $(0 \le a < B)$ and w_i $(0 \le i < m)$ be integer variables not occurring in other places. A syntactic translation $\mu(z, x, y)$ is defined as follows.

$$\mu(z,x,y) := \bigwedge_{i=0}^{m-1} \bigwedge_{a=0}^{B-1} \left((x^{(i)} \le a-1) \vee (x^{(i)} \ge a+1) \vee \bigwedge_{j=0}^{m-1} ({w_i}^{(j)} = {y_a}^{(j)}) \right)$$
$$\wedge \left[z = \sum_{i=0}^{m-1} B^i w_i \right] \wedge \bigwedge_{a=0}^{B-1} \nu(y_a, a, y)$$

where $[e]$ is a syntactic function reducing e by repeatedly using the Proposition 2 and digitwise left shift operations.

Proposition 3 Let $B > 2$ be a base, $z^{(i)}$, $x^{(i)}$ and $y^{(i)}$ be integer variables or constants of a C-CSP. Then the following holds for any α, where α' is an assignment containing only new integer variables introduced by μ.

$$(\alpha, \emptyset) \models z^{(m-1,0)} = x^{(m-1,0)} y^{(m-1,0)} \iff \exists \alpha'. \exists \beta. (\alpha \cup \alpha', \beta) \models \mu(z, x, y)$$

Proof. We omit the proof because it is easily confirmed.

Example 6. The following is an example of $\mu(z, x, y)$ when the base $B = 2$ and $u(z) = u(x) = u(y) = 3$. It represents a multiplication of 2-digit integers.

$$\begin{aligned} \mu(z,x,y) = {} & ((x^{(1)} \ge 1) \vee ({w_1}^{(0)} = {y_0}^{(0)} \wedge {w_1}^{(1)} = {y_0}^{(1)})) \\ & \wedge\ ((x^{(1)} \le 0) \vee ({w_1}^{(0)} = {y_1}^{(0)} \wedge {w_1}^{(1)} = {y_1}^{(1)})) \\ & \wedge\ ((x^{(0)} \ge 1) \vee ({w_0}^{(0)} = {y_0}^{(0)} \wedge {w_0}^{(1)} = {y_0}^{(1)})) \\ & \wedge\ ((x^{(0)} \le 0) \vee ({w_0}^{(0)} = {y_1}^{(0)} \wedge {w_0}^{(1)} = {y_1}^{(1)})) \\ & \wedge\ [z = Bw_1 + w_0] \wedge \nu(y_1, 1, y) \wedge \nu(y_0, 0, y) \end{aligned}$$

4.4 Whole Reduction

Now we explain the whole reduction to C-CSP.

Definition 8 For any base $B \ge 2$ and R-CSP formula C, we define the function C^* as follows.

$$\begin{array}{rcc} (p)^* = p & (x \le y)^* = \tau(x, y) & (z = xy)^* = \mu(z, x, y) \\ (\neg p)^* = \neg p & (z = x + a)^* = \sigma(z, x, a) & (C_1 \wedge C_2)^* = C_1^* \wedge C_2^* \\ (x \le a)^* = \tau(x, a) & (z = x + y)^* = \sigma(z, x, y) & (C_1 \vee C_2)^* = C_1^* \vee C_2^* \\ (x \ge a)^* = \tau(a, x) & (z = ax)^* = \mu(z, a, x) & \end{array}$$

Proposition 4 Let (X, u, P, C) be an R-CSP. Let $B \ge 2$ be a base and $(B; X', P', C')$ be a C-CSP defined as follows.

$X' = \{x^{(i)} \mid x \in X,\ 0 \le i < m\} \cup \{x \mid x \text{ is a new integer variable introduced to } C'\}$

$P' = P \cup \{p \mid p \text{ is a new propositional variable introduced to } C'\}$

$C' = C^* \wedge \bigwedge_{x \in X} (x \le u(x))^*$

Then the following holds, that is, the R-CSP and the C-CSP are equi-satisfiable.

$$\exists(\alpha,\beta) \models (X,u,P,C) \Longleftrightarrow \exists(\alpha',\beta') \models (B;X',P',C')$$

Proof. It can be shown from the Propositions 1, 2 and 3. □

Finally, we conclude any CSP can be reduced to a C-CSP.

Theorem 1 (Reduction of CSP to C-CSP) Any CSP can be reduced to an equi-satisfiable C-CSP for any base $B \geq 2$.

Proof. It can be shown from the Lemma 1 and the Proposition 4. □

It is possible to optimize the reduction for some special cases. For example, a formula $z \geq x + y$ can be straightforwardly reduced to a constraint of C-CSP representing a digitwise comparison and we can obtain smaller representation than the usual reduction of $z \geq x + y$ using $w = x + y \land w \leq z$.

5 Compact Order Encoding

We describe the compact order encoding realized by applying the order encoding to the C-CSP reduced from the original CSP. In the order encoding, each integer variable of domain size d is represented by using $O(d)$ propositional variables and each n-ary constraint $\sum_{i=1}^{n} a_i x_i \triangleright b$ is encoded into $O(d^{n-1})$ clauses where $\triangleright \in \{\leq, \geq, =\}$.

As described in the previous section, any CSP can be reduced to a C-CSP for any base B. Remind that each integer variable of a C-CSP is bound to $B-1$ and each constraint is at most ternary.

Each $x \leq y$ is encoded into $O(B \log_B d)$ clauses because $\tau(x,y)$ contains $O(\log_B d)$ comparisons and each comparison is encoded into $O(B)$ clauses. Similarly, each $z = x + y$ is encoded into $O(B^2 \log_B d)$ clauses. Each $z = ay$ $(0 \leq a < B)$ is encoded into $O(B^2 \log_B d)$ clauses in general because $\nu(z,a,y)$ contains $O(\log_B d)$ multiplications/additions and each one is encoded into $O(B^2)$ clauses. Especially when $B \geq d$, it can be encoded into $O(d)$ clauses. Each $z = xy$ is encoded into $O(B^3 \log_B d + B^2 \log_B^2 d)$ clauses in general because $\mu(z,x,y)$ contains $O(B)$ applications of ν, each ν is encoded into $O(B^2 \log_B d)$ clauses, and $O(\log_B d)$ additions, each addition is encoded into $O(B^2 \log_B d)$ clauses. Especially when $B \geq d$, it is encoded into $O(d^2)$ clauses because each ν is encoded into $O(d)$ clauses. Fig.3 summarizes the number of clauses required to encode each constraint where $m = \lceil \log_B d \rceil$. Note that $z = xy$ is encoded into $O(d^2)$ clauses by using the compact order encoding when $B \geq d$.

6 Experimental Results

In the compact order encoding, there is a trade-off between the size of SAT-encoded instances and the number of carry propagations. For example, choosing

Constraint	Order Encoding	Compact Order Encoding	Log Encoding
$x \le a$	$O(1)$	$O(m)$	$O(\log_2 d)$
$x \le y$	$O(d)$	$O(mB)$	$O(\log_2 d)$
$z = x + a$	$O(d)$	$O(mB)$	$O(\log_2 d)$
$z = x + y$	$O(d^2)$	$O(mB^2)$	$O(\log_2 d)$
$z = xy$	$O(d^2)$	$O(mB^3 + m^2B^2)$	$O(\log_2^2 d)$

Fig. 3. Comparison of different encodings on the number of SAT-encoded clauses

larger base means smaller number of digits and fewer carry propagations, but larger SAT-encoded instance. We made a choice of $B = \lceil d^{\frac{1}{2}} \rceil$, i.e., all variables are represented by at most two digits, in our experiments. In this case, each ternary constraint of addition and multiplication can be encoded into at most $O(d)$ and $O(d^{\frac{3}{2}})$ clauses respectively.

To confirm the effectiveness of this choice, we measured the basic performance of the compact order encoding for various bases in our previous work [26]. We used a handmade problem which consists of a sequence of $d + 1$ variables to be arranged in the range of size d. As the result, only the compact order encoding with $B = \lceil d^{\frac{1}{2}} \rceil$ solved all the given instances of $5000 \le d \le 30000$. The order encoding could not solve the large instances because of memory limitation. The log encoding could not solve the large instances within 2 hours.

To evaluate the effectiveness of our encoding, Open-Shop Scheduling (OSS) problems are used. We selected the most difficult "j7" and "j8" series (17 instances) of the benchmark set by Brucker *et al* [4]. These include three hard instances that were open until 2006: j7-per0-0, j8-per0-1, and j8-per10-2.

We compare four different encodings: the compact order encoding with $m \in \{2, 3\}$, the order encoding, and the log encoding. For each instance, we set its makespan to the optimum value minus one and then encode it into SAT. Such SAT-encoded instances are unsatisfiable. We use the MiniSat solver [7] as a high-performance SAT solver to solve them. We also compare our system with the state-of-the-art CSP solvers: choco 2.11 [27] and Mistral 1.550 [14].

Fig. 4 shows the times in seconds including both the CPU times of MiniSat for solving SAT-encoded instances and the encoding times. It also shows the CPU times in seconds of choco and Mistral. The column "Size" indicates the number of jobs and machines. All times were collected on a Linux machine with Intel Xeon 3.0 GHz, 16GB Memory. We set a timeout ("T.O.") to 3600 seconds for each instance. The row "#Solved" indicates the total number of solved instances. We highlight the best time for each instance and the best number of solved instances.

The compact order encoding, the order encoding, and Mistral solved the most instances. The compact order encoding with $m = 2$ solved 8 instances with the best CPU times, while choco was 5 and Mistral was 1. Moreover, the compact order encoding with $m = 2$ succeeded in solving the hard instance j8-per10-2 not solved by others.

Instance	Size	Order Encoding	Compact Order Encoding		Log Encoding	choco	Mistral
			$m = 2$	$m = 3$			
j7-per0-0	7x7	T.O.	T.O.	T.O.	T.O.	T.O.	T.O.
j7-per0-1	7x7	56.16	**11.18**	16.06	119.52	T.O.	27.10
j7-per0-2	7x7	36.15	**8.35**	11.78	85.39	T.O.	49.92
j7-per10-0	7x7	56.01	**15.47**	17.88	100.07	T.O.	76.81
j7-per10-1	7x7	24.98	7.74	6.82	66.32	**0.53**	0.97
j7-per10-2	7x7	497.15	298.91	**253.07**	2804.06	T.O.	546.06
j7-per20-0	7x7	4.43	4.17	3.09	5.18	0.54	**0.12**
j7-per20-1	7x7	13.38	**5.54**	7.54	19.80	T.O.	16.82
j7-per20-2	7x7	24.38	**7.91**	9.61	32.37	T.O.	26.76
j8-per0-1	8x8	T.O.	T.O.	T.O.	T.O.	T.O.	T.O.
j8-per0-2	8x8	161.73	**42.61**	119.35	478.10	T.O.	142.14
j8-per10-0	8x8	345.09	**157.60**	276.06	T.O.	T.O.	308.73
j8-per10-1	8x8	10.20	T.O.	17.76	10.65	**0.69**	1.38
j8-per10-2	8x8	T.O.	**2305.73**	T.O.	T.O.	T.O.	T.O.
j8-per20-0	8x8	3.14	3.06	14.16	13.83	**0.69**	1.53
j8-per20-1	8x8	3.05	15.35	7.75	21.64	**0.70**	1.30
j8-per20-2	8x8	3.83	2.95	22.43	17.61	**0.68**	1.43
#Solved		**14**	**14**	**14**	13	6	**14**

Fig. 4. Benchmark results of different encodings, choco, and Mistral on the "`j7`" and "`j8`" series of OSS benchmark set by Brucker *et al.*

To evaluate the scalability of our encoding, we used 102 OSS instances with very large domain sizes, which are generated from "`j7`" and "`j8`" by multiplying the process times by some constant factor c. The factor c is varied within 1, 10, 20, 100, 200, and 1000. For example, when $c = 1000$, the maximum domain size d becomes about 10^6. We then compare four different encodings, choco, and Mistral in the same way as our previous experiments.

Fig. 5 shows the number of solved instances by four encodings with MiniSat, choco and Mistral. "Domain Size d" indicates the approximate average of domain size of integer variables. We highlight the best number of solved instances.

The compact order encoding solved the most instances for any factor c and totally 80 out of 102 instances rather than 75 by the log encoding, 55 by Mistral, 51 by the order encoding, and 38 by choco. The compact order encoding with $m = 2$ can be highly scalable with the growth of c compared with the order encoding and Mistral. For example, when $c = 1000$, it solved 14 out of 17 instances (82%), while only 4 (24%) by Mistral and none (0%) by the order encoding.

Fig. 6 shows the cactus plot of benchmark results in which the number of solved instances is on the x-axis and the CPU time is on the y-axis. The compact order encoding solved the most instances for almost any CPU time limit. But surprisingly, the compact order encoding with $m = 3$ is slightly faster than that with $m = 2$.

Finally, we explain some details of our experiments. As mentioned in Section 5, in the compact order encoding, the number of clauses required is much

Factor c	Domain Size d	#Instances	Order Encoding	Compact Order Encoding		Log Encoding	choco	Mistral
				$m = 2$	$m = 3$			
1	$d \approx 10^3$	17	**14**	**14**	**14**	13	6	**14**
10	$d \approx 10^4$	17	12	**13**	**13**	**13**	6	12
20		17	12	13	**14**	13	6	12
100	$d \approx 10^5$	17	8	**13**	**13**	12	7	7
200		17	5	**13**	**13**	12	6	6
1000	$d \approx 10^6$	17	0	**14**	13	12	7	4
	Total	102	51	**80**	**80**	75	38	55

Fig. 5. Benchmark results of different encodings, choco, and Mistral on the number of solved instances for OSS benchmark set by Brucker *et al.* with multiplication factor c.

smaller than the order encoding. For example, in the case of j8-per10-2 with $c = 1000$, it requires 4.49 million clauses (file size: 132MB) even when $m = 2$, but the order encoding requires 900 million clauses (file size: 25GB). For the encoding time of the compact order encoding with $m = 2$, it takes less than 8 seconds including reduction time for every instances in Fig. 4 and 5. For Mistral and choco, we used the same command line options as used in the 2009 International CSP Solver Competition.

7 Related Works

FznTini [15] and SAT4J CSP [19] are SAT-based CSP solvers. FznTini uses the log encoding and SAT4J CSP uses the direct encoding and a variant of the support encoding. Eén and Sörensson proposed a SAT encoding of pseudo-Boolean constraints which are special cases of CSP constraints on 0-1 variables [8]. In their paper, each integer is represented in a numeral system of base B and each digit is represented in a unary representation which is similar to the order encoding. They also proposed to use a generalized numeral system with varying bases.

The use of propositional variables $p(x \leq a)$ are described in [11, 2, 13, 3], but the encoding of arithmetic constraints to them is first described in [5] and extended to any linear constraints in [24, 25].

8 Conclusion

We presented a new SAT encoding, named compact order encoding, aiming to be compact and efficient. The basic idea of the compact order encoding is the use of a numeral system of some base. Each integer variable is divided into some digits and each digit is encoded by using the order encoding.

To evaluate the effectiveness and scalability of our encoding, we used Open-Shop Scheduling problems as benchmarks and showed that the compact order encoding can be more efficient than the order encoding, the log encoding, and the state-of-the-art CSP solvers choco and Mistral. Moreover, we showed that

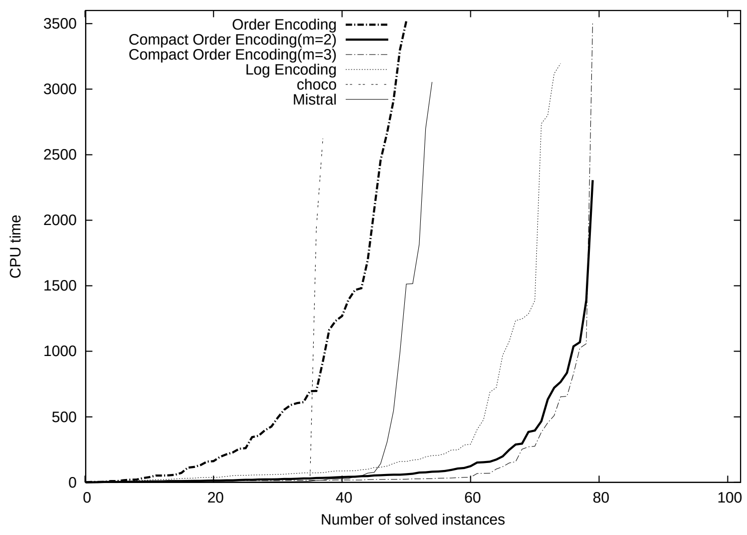

Fig. 6. Cactus plot of different encodings, choco, and Mistral for 102 OSS instances.

it can be highly scalable with the growth of domain size by solving very large instances which can not be solved by other solvers.

SAT encodings have an essential role in developing efficient SAT-based CSP solvers. There are several future topics for enhancing our encoding. Among them, it is very important to investigate the choice of appropriate base for solving a wide variety of problems.

References

1. Banbara, M., Matsunaka, H., Tamura, N., Inoue, K.: 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. pp. 112–126 (2010)
2. Beckert, B., Hähnle, R., Manyà, F.: Transformations between signed and classical clause logic. In: In Proc. 29th International Symposium on Multiple-Valued Logics (ISMVL ' 99. pp. 248–255. IEEE Press (1999)
3. Béjar, R., Hähnle, R., Manyà, F.: A modular reduction of regular logic to classical logic. In: In Proceedings if the 31st International Symposium on Multiple-valued Logic (ISMVL ' 01. pp. 221–226. IEEE Press (2001)
4. Brucker, P., Hurink, J., Jurisch, B., Wöstmann, B.: A branch & bound algorithm for the open-shop problem. Discrete Applied Mathematics 76(1–3), 43–59 (1997)
5. Crawford, J.M., Baker, A.B.: Experimental results on the application of satisfiability algorithms to scheduling problems. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994). pp. 1092–1097 (1994)
6. de Kleer, J.: A comparison of ATMS and CSP techniques. In: Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI 1989). pp. 290–296 (1989)

7. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), LNCS 2919. pp. 502–518 (2003)
8. Eén, N., Sörensson, N.: Translating pseudo-Boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation 2(1-4), 1–26 (2006)
9. Gavanelli, M.: The log-support encoding of CSP into SAT. In: Proceedings of the 13th International Joint Conference on Principles and Practice of Constraint Programming (CP 2007), LNCS 4741. pp. 815–822 (2007)
10. Gelder, A.V.: Another look at graph coloring via propositional satisfiability. Discrete Applied Mathematics 156(2), 230–243 (2008)
11. Gent, I.P., Nightingale, P.: A new encoding of alldifferent into SAT. In: Proceedings of the 3rd International Workshop on Modelling and Reformulating Constraint Satisfaction Problems (2004)
12. Gent, I.P.: Arc consistency in SAT. In: Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002). pp. 121–125 (2002)
13. Hähnle, R.: Uniform notation of tableau rules for multiple-valued logics. In: ISMVL. pp. 238–245 (1991)
14. Hebrard, E.: Mistral, a constraint satisfaction library. In: Proceedings of the 3rd International CSP Solver Competition. pp. 31–39 (2008)
15. Huang, J.: Universal Booleanization of constraint models. In: Proceedings of the 14th International Joint Conference on Principles and Practice of Constraint Programming (CP 2008), LNCS 5202. pp. 144–158 (2008)
16. Inoue, K., Soh, T., Ueda, S., Sasaura, Y., Banbara, M., Tamura, N.: A competitive and cooperative approach to propositional satisfiability. Discrete Applied Mathematics 154(16), 2291–2306 (2006)
17. Iwama, K., Miyazaki, S.: SAT-variable complexity of hard combinatorial problems. In: Proceedings of the IFIP 13th World Computer Congress. pp. 253–258 (1994)
18. Kasif, S.: On the parallel complexity of discrete relaxation in constraint satisfaction networks. Artificial Intelligence 45(3), 275–286 (1990)
19. Le Berre, D., Lynce, I.: CSP2SAT4J: A simple CSP to SAT translator. In: Proceedings of the second CSP Competition. pp. 43–54 (2008)
20. Nabeshima, H., Soh, T., Inoue, K., Iwanuma, K.: Lemma reusing for SAT based planning and scheduling. In: Proceedings of the International Conference on Automated Planning and Scheduling 2006 (ICAPS 2006). pp. 103–112 (2006)
21. Prestwich, S.D.: CNF encodings. In: Handbook of Satisfiability, pp. 75–97. IOS Press (2009)
22. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming. Elsevier Science Inc., New York, NY, USA (2006)
23. Soh, T., Inoue, K., Tamura, N., Banbara, M., Nabeshima, H.: A SAT-based method for solving the two-dimensional strip packing problem. Fundamenta Informaticae 102(3-4), 467–487 (2010), http://portal.acm.org/citation.cfm?id=1890507.1890516
24. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling finite linear CSP into SAT. In: Proceedings of the 12th International Joint Conference on Principles and Practice of Constraint Programming (CP 2006), LNCS 4204. pp. 590–603 (2006)
25. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling finite linear CSP into SAT. Constraints 14(2), 254–272 (2009)
26. Tanjo, T., Tamura, N., Banbara, M.: Towards a compact and efficient SAT-encoding of finite linear CSP. In: Proceedings of the 9th International Workshop on Constraint Modelling and Reformulation (ModRef 2010) (2010)
27. The choco team: choco: an open source Java constraint programming library. In: Proceedings of the 3rd International CSP Solver Competition. pp. 7–13 (2008)
28. Walsh, T.: SAT v CSP. In: Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP 2000). pp. 441–456 (2000)