Towards a Compact and Efficient SAT-encoding of Finite Linear CSP

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 applicable to finite linear CSP, named *compact order encoding*, which is designed to generate compact (small sized) and also efficient SAT instances. The basic idea of the compact order encoding is the use of a numeric system of base $B \geq 2$. Each integer variable is divided into m digits and each digit is encoded by using the order encoding. Therefore, it is a generalization of the order encoding (when $m = 1$), and the log encoding (when $B = 2$). In the compact order encoding, each binary constraint can be encoded into $O(B \log_B d)$ SAT clauses which is much less than $O(d)$ clauses of the order encoding where d is the maximum domain size. Therefore it enables to solve large problems that can not be solved by the order encoding. The compact order encoding can generate much efficient SAT instance than the log encoding in general because it uses fewer digits and enables faster propagations. We also confirmed these observations through some experimental results.

1 Introduction

A Propositional Satisfiability Testing Problem (SAT) is a combinatorial problem to find a propositional variable assignment which satisfies all given propositional formulas [1]. Recent performance improvement of SAT technologies makes SAT-based approaches applicable for solving hard and practical combinatorial problems, such as planning, scheduling, hardware/software verification, and constraint satisfaction.

A (finite) Constraint Satisfaction Problem (CSP) is a combinatorial problem to find an integer variable assignment which satisfies all given constraints on integers of finite domains. A SAT-based constraint solver is a program which solves a CSP by encoding it to SAT and searching solutions by a SAT solver.

There have been several methods proposed to encode CSP into SAT, such as the direct encoding [2, 3], the support encoding [4, 5], the log encoding [6, 7], and the order encoding [8, 9].

Among them, the order encoding, which was first used to encode job-shop scheduling problems by Crawford and Baker [10] and studied by Inoue *et al.* [11, 12] showed a good performance for a wide variety of problems, including open-shop scheduling problems [9] and two-dimensional strip packing problems [13].

Also a SAT-based CSP solver, named Sugar[3], based on the order encoding won in several categories of the 2008 and 2009 International CSP Solver Competitions.

One main reason of its good performance is due to the realization of fast propagations. In the order encoding, a propositional variable $p(x \le a)$ is used for each integer variable x and domain value a where $p(x \le a)$ is defined as true if and only if the variable x is less than or equal to a. The constraint $x \le y$ is encoded into SAT clauses of $p(x \le a) \vee \neg p(y \le a)$ for all domain value a. Therefore, 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, the size of the generated SAT instance becomes huge when the domain size of the original CSP is large. For example, the number of SAT clauses in encoding 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 uses a bit-wise representation for integer variables, and the size of the generated SAT instance, therefore, is quite small (linear to $\log d$). However, 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*, which is an integration of the order encoding and the log encoding aiming to be compact (small sized) and also efficient.

The basic idea of the compact order encoding is the use of a numeric system of base $B \ge 2$ to make the domain size small, 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.

However, simple replacement of integer variables with summations does not work. For example, the simple replacement of $x \le y$ will be $\sum_{i=0}^{m-1} B^i x_i \le \sum_{i=0}^{m-1} B^i y_i$ which contains $2m$ variables of domain size B and, therefore, requires $O(B^{2m-1}) = O(d^2)$ SAT clauses by the order encoding.

It is possible to introduce new integer variables to reduce the number of variables occurring in the summation, e.g. replacing $Bx_1 + x_0$ with a new variable x' with extra constraint $x' = Bx_1 + x_0$, the domain size of variables are, however, not bounded by B any more.

Therefore, in this paper, we first describe a reduction of a finite linear CSP with ternary (including unary and binary) constraints (called 3ary-CSP) to a Compact 3ary-CSP satisfying: all constraints are ternary, and the domain sizes of all variables are bounded by the base B.

By applying the order encoding to the reduced Compact 3ary-CSP, each constraint of the original 3ary-CSP can be encoded to at most $O(B^2 \log_B d)$ SAT clauses.

Please 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. Therefore, the compact order encoding can be seen as the generalization and integration of both encodings.

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

Choosing larger base means smaller number of digits and less carry-ripples, and therefore faster propagations, but larger SAT size. On the other hand, smaller base means smaller SAT size but slower propagations. Therefore, choosing appropriate base will be an important issue to be considered.

In this paper, we describe a choice of $B = \lceil d^{\frac{1}{2}} \rceil$, i.e., all variables are represented by at most two digits, is appropriate for some applications. In this case, each constraint of the 3ary-CSP can be encoded to at most $O(d)$ SAT clauses.

In the rest of the paper, we first define 3ary-CSP and Compact 3ary-CSP, and explain the reduction of 3ary-CSP to Compact 3ary-CSP followed by the explanation on the order encoding of Compact 3ary-CSP. As experimental results, basic propagation performance is measured for various domain size and base values. As a practical application, performance of solving Open-Shop Scheduling Problems in the compact order encoding is compared with those of the order encoding and the log encoding in addition to the comparison with the state-of-the-art constraint solvers choco [14] and Mistral [15].

2 3ary-CSP

First we define some notations. $\mathbb{Z}$ and $\mathbb{N}$ are used to denote a set of integers and non-negative integers respectively. $\mathbb{B} = \{\top, \bot\}$ is used to denote a set of propositional constants ($\top$ and $\bot$ represent “true” and “false” respectively).

The following is the definition of finite linear **3ary-CSP** (ternary CSP) which only consists of the constraints among at most three non-negative variables.

Definition 1 (3ary-CSP) A (finite linear) 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 $p \in P$, $n \in \{1, 2, 3\}$, $x_i \in X$, $a \in \mathbb{N}$, and $\rhd \in \{\leq, \geq, =\}$.

$$C ::= \; p \;\mid\; \neg p \;\mid\; \sum_{i=1}^{n} \pm x_i \rhd a \;\mid\; C \wedge C \;\mid\; C \vee C$$

In the following discussion, we simply call the finite linear 3ary-CSP as 3ary-CSP.

Please note that any finite linear CSP can theoretically be reduced to 3ary-CSP in the following way.

- 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 constant multiplication ax can be replaced by $\sum_{i=1}^{a} x$ when $a > 0$, or $\sum_{i=1}^{-a} -x$ when $a < 0$.

– A linear expression containing more than three variables can be reduced to ternary expressions by replacing partial summations with new variables.

Example 1. Let $w, x, y, z \in \{-1, 0, 1\}$ be integer variables. A constraint $w+2x-3y-4z=0$ can be reduced to $(x'_2 = x'+x') \wedge (y'_2 = y'+y') \wedge (y'_3 = y'_2+y') \wedge (z'_2 = z'+z') \wedge (z'_4 = z'_2+z'_2) \wedge (v = w'+x'_2) \wedge (-v+y'_3+z'_4 = 4)$ by introducing new variables $w' = w+1$, $x' = x+1$, $y' = y+1$, and $z' = z+1$.

An *assignment* of a 3ary-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 $\alpha(x) \le u(x)$ for any $x \in X$, the 3ary-CSP is called *satisfiable* and the assignment is called a *solution* of the 3ary-CSP. We use the following notation to represent the satisfiability.

$$(\alpha, \beta) \models C$$

Sometimes, we use the same notation $(\alpha, \beta) \models C$ for any formula C not limited to 3ary-CSP formulas. We also write $(\alpha, \beta) \models (X, u, P, C)$ when the 3ary-CSP is satisfiable by the assignment.

2.1 Restricted 3ary-CSP

In this subsection, we introduce a restricted form of 3ary-CSP called **Restricted 3ary-CSP**. The aim of introducing Restricted 3ary-CSP is just for the easier discussion on the reduction of 3ary-CSP to Compact 3ary-CSP described later.

Definition 2 (Restricted 3ary-CSP) A Restricted 3ary-CSP is a 3ary-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 $a \in \mathbb{N}$.

$$\begin{aligned} C ::= {} & p \mid \neg p \mid x \le a \mid x \ge a \mid x \le y \mid z = x + a \mid z = x + y \\ & \mid C \wedge C \mid C \vee C \end{aligned}$$

Formulas allowed in Restricted 3ary-CSP is very limited, but any 3ary-CSP formulas can be reduced to the restricted forms as shown by the following Lemma.

Lemma 1 Any 3ary-CSP (X, u, P, C) can be reduced to a Restricted 3ary-CSP.

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

Due to space limitations, we only show the reduction of $x_1 - x_2 - x_3 \ge a$. This formula can be translated into $(z_1 = x_2 + x_3) \wedge (z_2 = z_1 + a) \wedge (z_2 \le x_1)$ by introducing new integer variables z_1 and z_2. □

Any solution of the Restricted 3ary-CSP can be translated back to a solution of the original 3ary-CSP by simply removing the assignments for the newly introduced variables.

3 Compact 3ary-CSP

In this section, we define **Compact 3ary-CSP** which is a 3ary-CSP but the upper bounds of integer variables are fixed to some integer constant $B-1$ where $B \geq 2$ is called a *base*.

Definition 3 (Compact 3ary-CSP) Let $B \geq 2$ be an integer constant. A Compact 3ary-CSP $(B; X, P, C)$ is a 3ary-CSP (X, u, P, C) where $u(x) = B-1$ for any integer variable $x \in X$.

Example 2. Let us consider a Compact 3ary-CSP $(10; \{x_1, x_0\}, \emptyset, C)$ where C is given as follows.

$$C = (x_1 \leq 2) \wedge (x_1 \leq 1 \vee x_0 \leq 6)$$

Its satisfiable assignments can be summarized as follows.

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

Therefore, its satisfiability is equivalent to the satisfiability of $10x_1 + x_0 \leq 26$.

Example 3. Let us consider a Compact 3ary-CSP $(10; \{x_1, x_0, z_1, z_0\}, \{c\}, C)$ where C is given as follows.

$$\begin{aligned} C = {} & (c \vee z_0 = x_0 + 6) \wedge (c \vee z_1 = x_1 + 2) \\ & \wedge (\neg c \vee z_0 = x_0 - 4) \wedge (\neg c \vee z_1 = x_1 + 3) \end{aligned}$$

It is easy to confirm its satisfiability is equivalent to the satisfiability of $10z_1 + z_0 = 10x_1 + x_0 + 26$. When $c = \bot$, $C = (z_0 = x_0 + 6 \wedge z_1 = x_1 + 2)$ derives $10z_1 + z_0 = 10x_1 + x_0 + 26$. The same conclusion can be derived also when $c = \top$. As a matter of fact, the propositional variable c represents a carry bit from the summation of $x_0 + 6$.

4 Reducing 3ary-CSP into Compact 3ary-CSP

In this section, we will explain a method of reducing 3ary-CSP into Compact 3ary-CSP.

As described in the section 2.1, any 3ary-CSP can be reduced to a Restricted 3ary-CSP. So we consider a method to reduce a given Restricted 3ary-CSP (X, u, P, C) into a Compact 3ary-CSP $(B; X', P', C')$ for any base $B \geq 2$.

Before describing our reduction method, we introduce some symbols and notations. First, the *maximum domain size* d and its *order of magnitude* m are defined as follows.

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

To consider the reduction of Restricted 3ary-CSP to Compact 3ary-CSP, we need to introduce a new integer variable of Compact 3ary-CSP for each digit of each integer variable x of the Restricted 3ary-CSP.

So, we define $x^{(i)}$ as a syntactic function which generates a new integer variable symbol of Compact 3ary-CSP from an integer variable symbol x of Restricted 3ary-CSP and an integer i $(0 \le i < m)$. The intention of $x^{(i)}$ is 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 for any integer constant $a \in \mathbb{N}$ defined as follows.

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

Finally, the following is the definition of the notation of $x^{(j,i)}$ where x is either an integer variable or constant.

$$x^{(j,i)} := \sum_{k=i}^{j} B^{k-i} x^{(k)}$$

Remind that the constraint C in Restricted 3ary-CSP was defined as follows:

$$\begin{aligned} C ::= {} & p \mid \neg p \mid x \le a \mid x \ge a \mid x \le y \mid z = x + a \mid z = x + y \\ & \mid C \wedge C \mid C \vee C \end{aligned}$$

In the above definition, the formulas $x \le y$, $x \le a$, and $x \ge a$ can be reduced in a similar way. Also the formulas $z = x + a$ and $z = x + y$ can be reduced in a similar way. So we only focus on the reduction of $x \le y$ and $z = x + y$ in the following subsections.

4.1 Reduction of $x \le y$

The formula $x \le y$ of Restricted 3ary-CSP can be represented as $x^{(m-1,0)} \le y^{(m-1,0)}$ by using integer variables of targeted Compact 3ary-CSP.

In this subsection, we show a method to reduce the formula into Compact 3ary-CSP formula without changing the upper bounds of the integer variables.

First we show the following Lemma.

Lemma 2 For any non-negative integers b, x_i, y_i $(i = 0, 1)$ satisfying $b \ge 2$ and $x_0, y_0 < b$, the following holds.

$$bx_1 + x_0 \le by_1 + y_0 \iff (x_1 \le y_1) \wedge (x_1 \le y_1 - 1 \vee x_0 \le y_0)$$

Proof. $(\Longrightarrow)$ When $bx_1 + x_0 \le by_1 + y_0$, we can easily conclude $(x_1 \le y_1) \wedge (x_1 = y_1 \Rightarrow x_0 \le y_0)$ which is equivalent to the conclusion formula.
$(\Longleftarrow)$ For both cases of $(x_1 \le y_1) \wedge (x_1 \le y_1 - 1)$ and $(x_1 \le y_1) \wedge (x_0 \le y_0)$, we can derive $bx_1 + x_0 \le by_1 + y_0$. □

Now we define the reduction of $x^{(m-1,0)} \le y^{(m-1,0)}$ and prove its correctness.

Definition 4 (Reduction of $x \le y$) Let $B \ge 2$ be a base, x and y be integer variables or constants of a Reduced 3ary-CSP. Then $\tau(x, y)$ is a syntactic translation defined as follows.

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

Proposition 1 Let $B \ge 2$ be a base, $x^{(i)}$ and $y^{(i)}$ be integer variables of a Compact 3ary-CSP. Then the following holds.

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

Proof. It can be easily shown by recursively applying the Lemma 2. □

Please note that $\tau(x, y)$ only contains binary constraints of Compact 3ary-CSP and its size is linear to 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. $B^2x^{(2)} + Bx^{(1)} + x^{(0)} \le B^2y^{(2)} + By^{(1)} + y^{(0)}$.

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

4.2 Reduction of $z = x + y$

The formula $z = x + y$ of Restricted 3ary-CSP can be represented as $z^{(m-1,0)} = x^{(m-1,0)} + y^{(m-1,0)}$ by using integer variables of targeted Compact 3ary-CSP.

As in the case of $x \le y$, we show a method to reduce the formula into ternary formula without changing the upper bounds of the integer variables. However, in this case, we don't have an equivalent formula but an equi-satisfiable formula.

First we show the following Lemma.

Lemma 3 For any non-negative integers b, c, c', x, y, z satisfying $b \ge 2$, $c \le 1$, $c' \le 1$, and $x, y, z < b$, the following holds.

$$\begin{aligned}
bc + z = x + y + c' \iff &(c' = 1 \lor c = 1 \lor z = x + y) \\
\land \quad &(c' = 1 \lor c = 0 \lor z = x + y - b) \\
\land \quad &(c' = 0 \lor c = 1 \lor z = x + y + 1) \\
\land \quad &(c' = 0 \lor c = 0 \lor z = x + y + 1 - b)
\end{aligned}$$

Proof. Easily verified by considering all cases of $c = 0$ or 1, and $c' = 0$ or 1. $\square$

Now we define the reduction of $z^{(m-1,0)} = x^{(m-1,0)} + y^{(m-1,0)}$ and prove its correctness.

Definition 5 (Reduction of $z = x + y$) Let $B \geq 2$ be a base, z, x and y be integer variables or constants of a Restricted 3ary-CSP, and c_i $(0 \leq i \leq m)$ be propositional variables not occurring in other places. Then $\sigma(z, x, y)$ is a syntactic translation defined as follows.

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

Please note that $\sigma(z, x, y)$ only contains ternary constraints of Compact 3ary-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 Compact 3ary-CSP. Then the following holds for any assignment $(\alpha, \emptyset)$.

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

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 using Lemma 3 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 $y^{(i)}$. $\square$

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 Whole reduction

Now we define the whole reduction of Restricted 3ary-CSP to Compact 3ary-CSP.

Definition 6 For any base $B \geq 2$ and Restricted 3ary-CSP formula C, we define the function C^* as follows.

$$\begin{aligned}
(p)^* &= p \\
(\neg p)^* &= \neg p \\
(x \leq a)^* &= \tau(x, a) \\
(x \geq a)^* &= \tau(a, x) \\
(x \leq y)^* &= \tau(x, y) \\
(z = x + a)^* &= \sigma(z, x, a) \\
(z = x + y)^* &= \sigma(z, x, y) \\
(C_1 \wedge C_2)^* &= C_1^* \wedge C_2^* \\
(C_1 \vee C_2)^* &= C_1^* \vee C_2^*
\end{aligned}$$

Proposition 3 Let (X, u, P, C) be a Restricted 3ary-CSP. Let $B \geq 2$ be a base and $(B; X', P', C')$ be a Compact 3ary-CSP defined as follows.

$$\begin{aligned}
&X' = \{x^{(i)} \mid x \in X,\ 0 \leq i < m\} \\
&P' = P \cup \{p \mid p \text{ is a new propositional variable introduced to } C'\} \\
&C' = C^* \wedge \bigwedge_{x \in X} (x \leq u(x))^*
\end{aligned}$$

Then the following holds.

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

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

Finally, we conclude that any 3ary-CSP can be reduced to a Compact 3ary-CSP.

Theorem 1 (Reduction of 3ary-CSP to Compact 3ary-CSP) Any 3ary-CSP can be reduced to a Compact 3ary-CSP of any base $B \geq 2$.

Proof. It can be shown from the Lemma 1 and the Proposition 3. □

5 Compact Order Encoding

In this section, we describe the order encoding in general, and the compact order encoding realized by applying the order encoding to the Compact 3ary-CSP reduced from the original 3ary-CSP.

5.1 Order encoding

In the order encoding, a propositional variable $p(x \le a)$ is used for each integer variable x and domain value a where $p(x \le a)$ is defined as true if and only if the variable x is less than or equal to a.

Now we consider the encoding of a 3ary-CSP (X, u, P, C).

We use the following propositional variables for each integer variable $x \in X$.

$$p(x \le 0) \quad p(x \le 1) \quad p(x \le 2) \quad \cdots \quad p(x \le u(x) - 1)$$

Please note that $p(x \le u(x))$ is unnecessary because it is always true. We also assume $p(x \le a) = \bot$ whenever $a < 0$ and $p(x \le a) = \top$ whenever $a \ge u(x)$.

In addition, the following clauses are required for each variable x to specify the relation among these propositional variables.

$$\neg p(x \le a - 1) \vee p(x \le a) \qquad (1 \le a \le u(x) - 1)$$

The following is the Proposition presented in [9] to encode a linear comparison in general where $l(e)$ and $u(e)$ specifies the lower and upper bound of an expression e respectively.

Proposition 4 The following holds for any $x_i \in X$, non-zero integers a_i, and integer $c \ge l(\sum_{i=1}^n a_i\, x_i)$.

$$\sum_{i=1}^{n} a_i\, x_i \le c \iff \bigwedge_{\sum_{i=1}^n b_i = c - n + 1} \bigvee_i (a_i\, x_i \le b_i)^{\#}$$

Parameters b_i's range over $\mathbb{Z}$ satisfying $\sum_{i=1}^n b_i = c - n + 1$ and $l(a_i x_i) - 1 \le b_i \le u(a_i x_i)$ for all i. The translation $()^{\#}$ is defined as follows.

$$(a\, x \le b)^{\#} := \begin{cases} p(x \le \lfloor b/a \rfloor) & (a > 0) \\ \neg p(x \le \lceil b/a \rceil - 1) & (a < 0) \end{cases}$$

Proof. Please refer to [9].

Linear comparisons $\sum_{i=1}^n a_i\, x_i \ge c$ and $\sum_{i=1}^n a_i\, x_i = c$ can be encoded by using $\sum_{i=1}^n -a_i\, x_i \le -c$ and $\sum_{i=1}^n a_i\, x_i \le c \wedge \sum_{i=1}^n a_i\, x_i \ge c$ respectively.

Example 6. A linear comparison $x + y \le 7$ can be encoded to $p(y \le 5) \wedge (p(x \le 2) \vee p(y \le 4)) \wedge (p(x \le 3) \vee p(y \le 3)) \wedge (p(x \le 4) \vee p(y \le 2)) \wedge p(x \le 5)$ when $l(x) = l(y) = 2$ and $u(x) = u(y) = 6$.

To generate a SAT instance, the formula should finally be converted to a Conjunctive Normal Form (CNF). Tseitin transformation can be used to convert the formula into an equi-satisfiable CNF formula in linear time by introducing a linear number of new propositional variables [16].

5.2 Compact Order Encoding of 3ary-CSP

As described in the previous section, any 3ary-CSP can be reduced to a Compact 3ary-CSP for any base B.

Since the Compact 3ary-CSP contains only ternary constraints and the domain size of variables are bounded by B, each constraint of the original 3ary-CSP can be encoded $O(B^2 \log_B d)$ SAT clauses in the worst case.

The following table summarizes the number of SAT clauses required to encode each constraint.

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

When choosing $B = \lceil d^{\frac{1}{m}} \rceil$ for some $m > 0$, that is, when representing in m-digits, the above table can be rewritten as follows.

	Order	Compact Order	Log
Constraint	Encoding	Encoding	Encoding
$x \le a$	$O(1)$	$O(m)$	$O(\log_2 d)$
$x \le y$	$O(d)$	$O(md^{\frac{1}{m}})$	$O(\log_2 d)$
$z = x + a$	$O(d)$	$O(md^{\frac{1}{m}})$	$O(\log_2 d)$
$z = x + y$	$O(d^2)$	$O(md^{\frac{2}{m}})$	$O(\log_2 d)$

6 Experimental Results

In order to compare the basic performance of the compact order encoding with those of the order and log encodings, we use a handmade problem named a sequence problem which consists of a sequence of $n+1$ variables to be arranged in a given range.

As for a practical application, the performance of the compact order encoding for solving Open-Shop Scheduling Problems is compared with those of the order and log encodings.

6.1 Sequence Problems

A *sequence problem* of length n is defined as the following 3ary-CSP $(X, u, \emptyset, C)$.

- $X = \{x_i \mid 0 \le i \le n\}$
- $u(x) = n - 1$ for each $x \in X$
- $C = \bigwedge_{i=0}^{n-1} x_i + 1 \le x_{i+1}$

This problem is unsatisfiable for any n since there are $n+1$ variables to be arranged in the range of size n.

We compare the compact order encoding with the order and log encodings in the size of the generated SAT instance, the memory consumption, and the CPU time to prove the unsatisfiability of the instance by a SAT solver for this problem.

$\lceil d^{\frac{1}{m}} \rceil$ $(1 \leq m \leq 4)$ and 2 are chosen as a base B for the compact order encoding. Please note that it is equivalent to the order encoding when $m = 1$, and it is equivalent to the log encoding when $B = 2$. The length n is varied within 5000, 8000, 10000, 20000, and 30000.

Each constraint of the sequence problem of length n is reduced to $O(m)$ constraints of the Compact 3ary-CSP, and then reduced Compact 3ary-CSP is encoded to SAT by using the order encoding.

	Order Encoding	Compact Order Encoding			Log Encoding
n		$m = 2$	$m = 3$	$m = 4$	
5000	1006	56	29	21	17
8000	2644	123	52	38	27
10000	4156	174	72	48	45
20000	17956	509	201	119	82
30000	40954	978	352	227	127

Fig. 1. Comparison of the file sizes of the generated SAT instances for sequence problems

Fig. 1 shows the file sizes of the generated SAT instances in mega bytes. Compared with the order encoding, the compact order encoding generates much smaller SAT instances even when $m = 2$. Especially, when $n = 30000$, the size of the order encoding is more than 40 giga bytes while the size of the compact order encoding is less than 1 giga bytes.

Fig. 2 and Fig. 3 show the memory consumptions in mega bytes and the CPU times in seconds for solving the generated SAT instances. We used MiniSat 2.0 core solver [17] as a backend SAT solver on Intel Xeon 3.0 GHz, 16GB Memory machine with time limit of 2 hours (7200 seconds). In the figure, "M.O." means the out of memory, and "T.O." means that the problem could not be solved in 2 hours (7200 seconds).

Only the compact order encoding solved all given instances. The order encoding could not solve the instances due to the out of memory when $n \geq 10000$. The log encoding could not solve the instances in 2 hours when $n \geq 10000$.

When $n \leq 8000$, the order encoding solved the instances without any decisions, but the compact order and log encodings need some decisions because of the carry-ripples.

As a summary, choosing the order of magnitude $m = 2$ (i.e. $B = \lceil d^{\frac{1}{2}} \rceil$) is the most effective choice for this problem.

	Order Encoding	Compact Order Encoding			Log Encoding
n		$m = 2$	$m = 3$	$m = 4$	
5000	4827.61	231.84	121.57	104.86	200.80
8000	13073.18	435.35	221.14	194.59	502.07
10000	M.O.	622.10	377.71	261.69	T.O.
20000	M.O.	1795.87	1028.27	1035.88	T.O.
30000	M.O.	3220.83	T.O.	T.O.	T.O.

Fig. 2. Comparison of the memory consumptions for sequence problems

	Order Encoding	Compact Order Encoding			Log Encoding
n		$m = 2$	$m = 3$	$m = 4$	
5000	14.29	64.78	76.58	103.33	596.80
8000	47.02	189.03	212.21	384.93	2611.44
10000	M.O.	382.95	650.58	526.52	T.O.
20000	M.O.	1527.46	4889.55	6311.37	T.O.
30000	M.O.	4631.40	T.O.	T.O.	T.O.

Fig. 3. Comparison of the CPU times for sequence problems

6.2 Open-Shop Scheduling Problems

An Open-Shop Scheduling (OSS) Problem consists of n jobs J_1, J_2, …, J_n and n machines M_1, M_2, …, M_n. Each job J_i consists of n operations O_{i1}, O_{i2}, …, O_{in}. An operation O_{ij} is performed at machine M_j and has a process time p_{ij} (positive integer). Any two different operations O_{ij} and O_{ik} of the same job J_i can not be processed at the same time. Any two different operations O_{ij} and O_{kj} of the same machine M_j can not be processed at the same time. The objective of OSS as a decision problem is to decide whether the all jobs can be completed within the given deadline, called makespan.

As an OSS instance, "`j6-per0-0`" by Brucker *et al.* [18] is chosen (its optimum makespan is 1056). It consists of 6 jobs and 6 machines. To obtain larger domain problems, we also use instances generated from "`j6-per0-0`" by multiplying the process times by some constant factor c. The factor c is varied within 1, 10, 50, 100, 200, and 1000. Their optimum makespan values are 1056, 10560, 52800, 105600, 211200, and 1056000 respectively. For each instance, the makespan is set to the optimum value minus one which is the most difficult case.

We compare the compact order encoding of $m = 2$ (i.e. $B = \lceil d^{\frac{1}{2}} \rceil$) with the order and log encodings in the CPU time to solve (or to prove the unsatisfiability of) the instance by a SAT solver for this problem. The comparison is also made with the state-of-the-art constraint solvers choco 2.11 [14] and Mistral [15].

Fig. 4 shows the CPU times in seconds spent by MiniSat 2.0 core solver on Intel Xeon 3.0 GHz, 16GB Memory machine with time limit of 1 hours (3600 seconds). In the figure, "T.O." means that the problem could not be solved in 1 hours, and "M.O." means the out of memory.

Multiplication factor c	makespan	Order Encoding	Compact Order Encoding	Log Encoding	choco	Mistral
1	1055	124.55	19.22	381.27	975.85	110.47
10	10559	1351.15	57.99	1055.78	2213.90	22.49
50	52799	M.O.	88.18	820.77	3165.83	15.25
100	105599	M.O.	117.37	698.89	T.O.	T.O.
200	211199	M.O.	161.00	770.69	T.O.	T.O.
1000	1055999	M.O.	338.29	1250.69	2320.69	T.O.

Fig. 4. Comparison of the CPU times for `j6-per0-0` with multiplication factor c

Compared with other encodings, the compact order encoding is the fastest for any c. The CPU time in the compact order encoding is much shorter than the CPU time in the order encoding. The log encoding is the second fastest. But the compact order encoding is over 4 times faster than the log encoding. The order encoding uses much memory than the compact order encoding and it takes considerably longer time when the multiplication factor c becomes larger.

Compared with CSP solvers, only the compact order and log encodings solved the all given instances.

7 Conclusion

In this paper, we proposed a new encoding method, named compact order encoding, which is an integration of the order encoding and the log encoding aiming to be compact (small sized) and also efficient.

The compact order encoding is realized by two steps. In the first step, a finite linear CSP is reduced to a Compact 3ary-CSP in which the domain sizes of all variables are bounded by some $B \geq 2$. In the next step, the obtained Compact 3ary-CSP is encoded into SAT by using the order encoding.

The features of the compact order encoding can be summarized as follows.

– It is a generalization of the order and log encodings.
– It is compact. Each ternary constraint of 3ary-CSP is encoded into at most $O(B^2 \log_B d)$ SAT clauses where B is a base and d is the domain size. It is much less than $O(d^2)$ SAT clauses of the order encoding.
– It is efficient. The compact order encoding is considered to be more efficient than the log encoding in general because it requires less carry-ripples.

We also confirmed these observations through some experimental results.

References

1. Biere, A., Heule, M., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. Volume 185 of Frontiers in Artificial Intelligence and Applications (FAIA). IOS Press (2009)

2. de Kleer, J.: A comparison of ATMS and CSP techniques. In: Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI 1989). (1989) 290–296
3. Walsh, T.: SAT v CSP. In: Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP 2000). (2000) 441–456
4. Kasif, S.: On the parallel complexity of discrete relaxation in constraint satisfaction networks. Artificial Intelligence **45**(3) (1990) 275–286
5. Gent, I.P.: Arc consistency in SAT. In: Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002). (2002) 121–125
6. Iwama, K., Miyazaki, S.: SAT-variable complexity of hard combinatorial problems. In: Proceedings of the IFIP 13th World Computer Congress. (1994) 253–258
7. Gelder, A.V.: Another look at graph coloring via propositional satisfiability. Discrete Applied Mathematics **156**(2) (2008) 230–243
8. 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. (2006) 590–603
9. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling finite linear CSP into SAT. Constraints **14**(2) (2009) 254–272
10. 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). (1994) 1092–1097
11. 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) (2006) 2291–2306
12. 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). (2006) 103–112
13. Soh, T., Inoue, K., Tamura, N., Banbara, M., Nabeshima, H.: A SAT-based method for solving the two-dimensional strip packing problem. Journal of Algorithms in Cognition, Informatics and Logic (2009)
14. The choco team: choco: an open source Java constraint programming library. In: Proceedings of the 3rd International CSP Solver Competition. (2008) 7–13
15. Hebrard, E.: Mistral, a constraint satisfaction library. In: Proceedings of the 3rd International CSP Solver Competition. (2008) 31–39
16. Prestwich, S.D.: CNF encodings. In: Handbook of Satisfiability. IOS Press (2009) 75–97
17. 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. (2003) 502–518
18. Brucker, P., Hurink, J., Jurisch, B., Wöstmann, B.: A branch & bound algorithm for the open-shop problem. Discrete Applied Mathematics **76**(1–3) (1997) 43–59